TU Delft

Delft University of Technology

Leveraging Data in Algorithm Design
for Problems in Bilevel Optimization, Adaptable Robust Optimization, and Phylogenetics

Julien, E.A.T.

DOI
10.4233/uuid:3df57b28-1976-4d62-a483-2c513e062d89

Publication date
2025

Document Version
Final published version

Citation (APA)
Julien, E. A. T. (2025). *Leveraging Data in Algorithm Design: for Problems in Bilevel Optimization, Adaptable Robust Optimization, and Phylogenetics*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:3df57b28-1976-4d62-a483-2c513e062d89

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

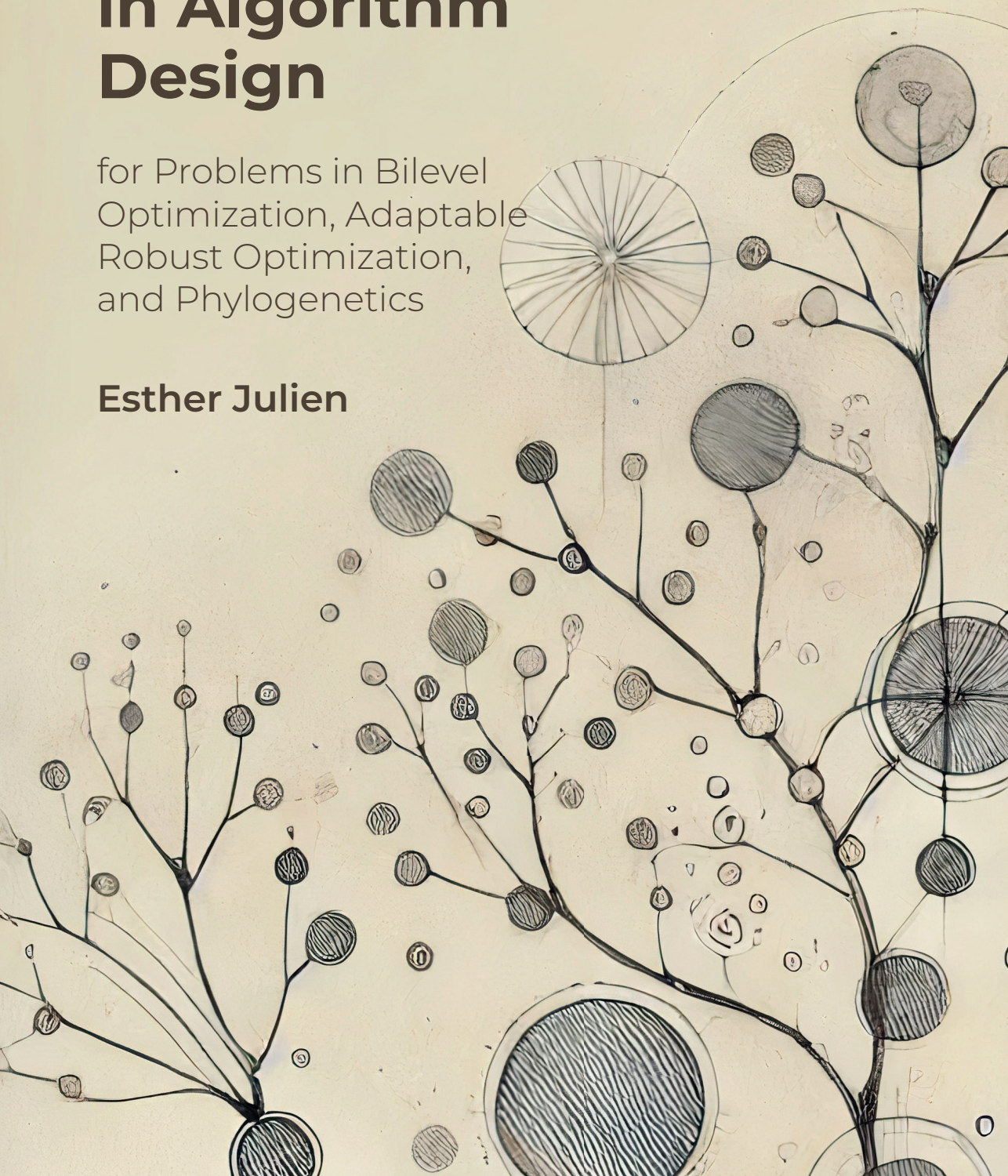
Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Leveraging Data in Algorithm Design

for Problems in Bilevel Optimization, Adaptable Robust Optimization, and Phylogenetics

Esther Julien



Leveraging Data in Algorithm Design

for Problems in Bilevel Optimization, Adaptable Robust Optimization, and Phylogenetics

LEVERAGING DATA IN ALGORITHM DESIGN

FOR PROBLEMS IN BILEVEL OPTIMIZATION, ADAPTABLE ROBUST OPTIMIZATION, AND PHYLOGENETICS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, Prof.dr.ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on Wednesday 17 September 2025 at 10:00 o'clock

by

Esther Anna Theresia JULIEN

Master of Science in Econometrics and Management Science,
Erasmus University Rotterdam, the Netherlands
born in Geldrop, the Netherlands.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Dr.ir. L.J.J. van Iersel,	Delft University of Technology, promotor
Prof.dr. L. Stougie,	Vrije Universiteit Amsterdam, promotor
Independent members:	
Dr. M. Bodur,	The University of Edinburgh, Scotland
Dr. S. Linz,	The University of Auckland, New Zealand
Prof.dr. W. Romeijnders,	The University of Groningen
Prof.dr. M.M. de Weerdt	Delft University of Technology
Prof.dr. D.C. Gijswijt	Delft University of Technology, reserve member
Other members:	
Prof.dr. Ş.İ. Birbil,	University of Amsterdam

This research was partly funded by the Dutch Research Council (NWO), under project OCENW.GROOT.2019.015.

Keywords:	Discrete Optimization, Machine Learning, Robust Optimization, Bilevel Optimization, Phylogenetics
Printed by:	Proefschriften.nl
Front & Back:	Proefschriften.nl

Copyright © 2025 by E.A.T. Julien

ISBN 978-94-6473-887-2

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Contents

SUMMARY

This thesis explores the integration of machine learning (ML) into algorithm design for solving complex combinatorial optimization problems. We focus on solving problems that arise in *bilevel* optimization, *two-stage robust* optimization, and *phylogenetics*. Although these problem classes seem vastly different, they share the common characteristic of being extremely challenging to solve. Current solution methods for these problems are computationally heavy and their solving duration is often impractical for already small-sized instances. Adding a component of machine learning also adds complexity. We primarily attempt to mitigate this effect by proposing learning methods that utilize efficient training data generation schemes that rely on solving sub-problems or are even based on a simple procedure, instead of learning from optimal solutions of many instances similar to the target problem.

In the first part of the thesis, data-driven algorithms for bilevel and two-stage robust optimization are introduced. By utilizing techniques like neural network representations (as a mixed-integer linear program), bilevel problems are transformed into single-level formulations. Column-and-constraint generation, a well-known algorithm that requires iterative steps to solve two-stage robust optimization problems, is simplified by using similar neural network representations. These proposed methods, respectively NEUR2BILO and NEUR2RO, can find similar quality solutions as state-of-the-art methods within a fraction of their runtimes. For another algorithm that solves two-stage robust optimization problems (approximately), K-adaptability branch-and-bound, an ML-based node selection strategy, is proposed to more efficiently traverse within the branch-and-bound search tree. Similar quality solutions as obtained by using the default random strategy are achieved up to 90% faster with the ML-based strategy. All three proposed methods require a data generation scheme based on solving single-level problems.

In the second part, we focus on a problem that arises in phylogenetics: the reconstruction of phylogenetic networks based on multilocus data, i.e., data based on different parts of the species' genomes, giving rise to different phylogenetic trees. The combinatorial problem that emerges is then how we can construct a phylogenetic network such that all trees are displayed in the network. This problem is already NP-hard for two bifurcating trees. In this thesis, a class of heuristics is proposed based on the mathematical characterization of *cherry picking* and machine-learning techniques. We demonstrate the practicality of these heuristics through experiments on multifurcating tree sets with missing leaves, which show good overall performance and particularly strong results on large instances with up to 100 trees and 100 leaves. Applying these heuristics creates a so-called orchard network. We show that a related theoretical problem, which is also biologically motivated, is NP-hard: how many auxiliary leaves should be added to an arbitrary network to make it orchard?

SAMENVATTING

Dit proefschrift focust zich op het gebruik van machine learning (ML) voor de ontwikkeling van algoritmes die gebruikt worden om complexe combinatorische optimalisering problemen op te lossen. In het bijzonder worden deze methoden toegepast op problemen die voorkomen in *bilevel* optimalisering, *two-stage robuuste* optimalisering en *fylogenetica*. Hoewel deze probleemklassen sterk van elkaar lijken te verschillen, zijn ze alle drie uiterst moeilijk om op te lossen. Huidige oplossingsmethoden kunnen alleen erg kleine instanties binnen een redelijke tijd oplossen. Het toevoegen van een ML-component brengt zelf ook extra complexiteit met zich mee. Dit effect proberen we te beperken door ML methodes te introduceren die gebruik maken van efficiënte datageneratie methoden. Die methoden baseren zich op de oplossingen van sub-problemen of zelfs op simpele processen, in plaats van de optimale oplossingen van vele vergelijkbare probleem instanties.

In het eerste gedeelte van het proefschrift worden algoritmes geïntroduceerd voor bilevel en two-stage robuuste optimalisering waarin het gebruik van data centraal staat. Zo worden neurale netwerk representaties (als een gemengd geheeltallig lineair programma) gebruikt om van een bilevel probleem een enkel-level probleem te maken. Daarnaast wordt een bekend algoritme voor het oplossen van two-stage robuuste optimalisering problemen, genaamd column-and-constraint generation, vereenvoudigd door vergelijkbare neurale netwerk representaties toe te passen. De twee ontwikkelde methoden, NEUR2BILO en NEUR2RO, kunnen beide kwalitatief vergelijkbare oplossingen vinden als de huidige beste methoden, in een fractie van hun tijdsduur. Voor een algoritme dat een lichtelijk versimpelde versie van two-stage robuuste optimalisering oplost, genaamd K-adaptability branch-and-bound, wordt een op ML gebaseerde knoopselectiestrategie voorgesteld, om efficiënter door de branch-and-bound zoekboom heen te gaan. Kwalitatief vergelijkbare oplossingen worden tot 90% sneller gevonden wanneer de knoopselectiestrategie op ML is gebaseerd, vergeleken met de standaard kansgestuurde strategie. Ieder van de drie bovengenoemde technieken leert op basis van oplossingen van enkel-level problemen.

In het tweede gedeelte van het proefschrift focussen we ons op problemen die voorkomen in fylogenetica. Namelijk, op het reconstrueren van fylogenetische netwerken, gebaseerd op multilocus data, d.w.z., data gebaseerd op verschillende delen van de genomen van meerdere organismen, wat resulteert in verschillende fylogenetische bomen. Het combinatorische probleem wat vervolgens ontstaat is hoe we een fylogenetisch netwerk kunnen reconstrueren zodat alle bomen erin worden weergegeven. Dit probleem is al NP-hard voor twee binaire bomen. In dit proefschrift wordt een klasse van heuristieken geïntroduceerd die zijn gebaseerd op de wiskundige karakterisatie van *cherry picking* en ML technieken. Door deze heuristieken te gebruiken, wordt een zogenaamd

orchard netwerk gereconstrueerd. We bewijzen dat het volgende gerelateerde theoretische probleem, dat ook biologisch gemotiveerd is, NP-hard is: hoeveel bijkomende 'bladeren' moeten aan een arbitrair netwerk worden toegevoegd om het orchard te maken?

1

INTRODUCTION

1.1. MACHINE LEARNING FOR OPTIMIZATION

This thesis focuses on leveraging machine-learning (ML) methods to solve combinatorial optimization problems. In the following, we discuss the applicability of machine learning in optimization and the types of roles it can take. The problems studied in this thesis are then presented alongside the goals.

ROLES OF MACHINE LEARNING IN OPTIMIZATION

There are many different reasons for adopting ML techniques to solve optimization problems. For instance, *the problem description is known, but (parts of) the parameters are not, which can be predicted.* The parameters define an instance of a problem, e.g., for routing problems, the duration times of traversing over roads can be uncertain in advance. A straightforward way to deal with this uncertainty is simply predicting these parameters by using any predictor that best suits the application; this is an initial step after which the predicted values are used as input for an existing solution algorithm. Whether this approach is advised is another issue. Some might suggest that considering the uncertainty while solving the problem leads to better, more frequently feasible, solutions. Robust and stochastic optimization can then be adopted. These problem structures are unfortunately very complex and solving them takes substantially longer than their deterministic counterparts. Another approach for predicting unknown problem parameters lies in the 'end-to-end predict-then-optimize' paradigm [69, 194]. The predictor is trained not for the accuracy of the uncertain parameters it aims to predict, but for the optimal solution that corresponds to the predictions. This idea stems from the observation that the model cannot always predict accurately. As we mainly care about the decisions of the solution and not the correct values of the parameters, this alternative objective is assumed for training the prediction model.

Another setting where ML can be utilized is the following. Often when solving a problem, one can derive the correct formulations of the mixed-integer programming (MIP) [158] problem relatively easily. As is the case with problems that resemble classic problem structures such as that of knapsack, machine scheduling, or routing problems.

However, sometimes the problem description is unknown and ML can help define it: *the objective value function or constraints of an MIP problem can be predicted.* A model is trained on historical (or simulated) data that, for example, can classify good and/or feasible decisions based on some features of problem instances [71]. This usually entails that the machine-learning model needs to be integrated into the solution algorithm, which is a challenging setting. Decision trees and neural networks can be formulated as mixed-integer linear programming variables and constraints, which can then be added to the original MIP [75, 145]. These added formulations serve as a proxy for the missing elements.

Assumed setting in this thesis. Most problems studied in combinatorial optimization are very complex: the runtime does not have a polynomial relationship with the problem size but grows much faster. This brings us to the application of machine learning in optimization considered in this thesis. Namely, when the problem is known, but a

data-driven algorithm design that leads to better performance

is proposed. Algorithms typically involve iterative steps where the choice made per step shapes the path forward. This influences the duration of the algorithm and/or the quality of the solution, making effective decisions critical. Machine learning can be used in each iteration to predict which choice leads to the most efficient or best solution. In some cases, existing algorithms can be improved by adding ML for these choices. In other cases, new algorithms are designed specifically for the integration of ML embedding. Simulated or historical data that describe the states and best actions of each step can then be used as expert data for practical applications. This is the *supervised* learning approach. *Reinforcement learning* can also be applied [147], where no historical data is required. This is an *unsupervised* learning approach.

Problems studied in this thesis. We focus on data-driven algorithm design for problems of the following types. There is a natural divide that can be made between the problems, which gives rise to the thesis being split into two parts:

I. *Hierarchical Optimization*: Bilevel and two-stage robust optimization,

II. *Phylogenetics*: Reconstructing phylogenetic networks describing evolutionary histories.

For the methods we introduce, *supervised* learning techniques are used where data is necessary for training the learning model. We focus on developing an effective solution algorithm for unseen problem instances. To achieve this, we need solutions from many similar problem instances. Sometimes the necessary data is not available. The challenge is then

to generate training data efficiently,

which can be considered as another goal of this thesis. A potentially ineffective approach would be to solve many training instances, that are similar in complexity as the target problem, to *optimality*. Various alternatives are presented throughout this thesis: (a) solving not the entire problem but instead subproblems and (b) generating problem instances from solutions; option (a) is used in Part I, and option (b) in Part II.

HIGH-LEVEL GOALS OF THIS THESIS

The first main goal of this thesis is to design data-driven algorithms that lead to improved performance for solving bilevel optimization problems, two-stage robust optimization problems, and reconstructing a phylogenetic network. To minimize the burden of applying machine learning to an algorithm, we have considered the added complexity introduced and tried to mitigate these effects. This can be achieved by choosing ML models that are well-suited to the application while prioritizing those that use fewer parameters. Another way is by designing efficient training data generation methods, which can be viewed as the second goal of the thesis. The two goals are intertwined: the efficiency of data generation goes hand-in-hand with the design of the algorithm, as the learning task depends on its algorithmic application.

1.2. THESIS OUTLINE

As previously mentioned, this thesis is divided into two parts: Part I for hierarchical optimization and Part II for phylogenetics. An introduction of each part follows in this chapter. We start Part I with preliminaries in Chapter 2. This chapter explains neural network representations as mixed-integer linear programming models, used in the two subsequent chapters. We then move on to the chapters introducing how machine learning can be integrated into algorithm design for hierarchical optimization problems. In Chapter 3, a learning-based method for bilevel optimization is presented. In Chapter 4, a method using machine-learning techniques is proposed for the existing algorithm *column-and-constraint generation* to solve two-stage robust optimization problems more efficiently. In Chapter 5, we present a learning-based node selection strategy for another algorithm that solves two-stage robust optimization problems: the *K-adaptability branch-and-bound* algorithm.

We then proceed to Part II, where we apply machine learning to problems in phylogenetics and examine related theoretical questions specific to phylogenetics. In Chapter 6, a learning-based heuristic for the hybridization problem on binary phylogenetic trees is introduced. Then, in Chapter 7, this heuristic is adapted to generalize to non-binary (or multifurcating) trees. By applying the introduced heuristics, one creates an *orchard* network. We explore a more theoretically motivated question in Chapter 8 on this class of networks. Namely, whether we can attach leaves to a *phylogenetic network* to transform it into an orchard network. We show that this is NP-hard and present a mixed-integer linear programming (MILP) formulation to solve this problem. Finally, concluding remarks and future directions are given in Chapter 9.

1.3. PART I: HIERARCHICAL OPTIMIZATION

The first part of the thesis deals with two problem classes: *bilevel* and *two-stage robust* optimization. Their formulations share a similar mathematical structure in which multiple problems appear in a *hierarchical* or *nested* order; a problem of a higher level requires the optimal solution of some decisions of a lower-level problem. This structure comes at a cost: problems from both optimization classes are generally NP-hard, even when their nested problems are polynomially solvable. The two classes face similar issues, and consequently, the high-level ideas of some of the methods proposed in this

thesis are shared. In Chapter 3 and Chapter 4 we use ML to avoid the nested structure of bilevel and two-stage robust optimization problems to some degree, respectively.

Although similar in structure, the literature on robust and bilevel optimization has mostly been developed in parallel throughout the years. In this thesis, the classes are also studied individually, as the methods must be tailored to class-specific characteristics. A clear misalignment is that bilevel optimization has two nested problems, whereas two-stage robust optimization has three, and a generalization to higher numbers of nested problems is not trivial. In an attempt to bridge the gap between the two research fields, Goerigk et al. [87] have shown similarities between (two-stage) robust and bilevel optimization and illustrate that sometimes the same algorithm can be used to solve problems of the two classes when certain restrictions hold.

Only recently there has been increasing interest in machine learning for hierarchical optimization classes with mixed-integer variables, of which the works presented in this thesis are some of the first. For the simpler class of non-nested optimization problems, i.e., regular combinatorial problems, there is a richer history of literature on using ML, with people focusing on solving specific (types of) problems [114] and accelerating exact algorithms like branch-and-bound and cutting planes for solving general mixed-integer linear programs. For extensive surveys of this research field, we refer the reader to [23, 175].

1.3.1. PROBLEM CLASSES

In the following, bilevel optimization (BiLO) and two-stage robust optimization (2RO) will be introduced. The formulations of general mixed-integer programming (MIP) and robust optimization (RO) are additionally provided for completeness.

MIXED INTEGER PROGRAMMING (MIP)

Mixed-integer programming is a class of mathematical optimization problems where some decision variables are restricted to integer values, while others can be continuous. Many real-life problems in which 'yes' or 'no' (i.e., binary) or discrete decisions need to be made, can be formulated as an MIP:

$$\begin{aligned} \min_{\mathbf{x}\in\mathcal{X}} \quad & F(\mathbf{x}) \\ \text{s.t.} \quad & G(\mathbf{x}) \geq \mathbf{0}, \end{aligned} \tag{MIP}$$

where $\mathbf{x}$ are the decisions in some domain $\mathcal{X} \subseteq \mathbb{R}^n$, that can be restricted such that (some) variables can only take integer values. The objective function is given as $F : \mathcal{X} \to \mathbb{R}$, and the constraints as $G : \mathcal{X} \to \mathbb{R}^m$, which can in the general setting all be non-linear. MIP has many applications in industries such as logistics, manufacturing, and energy. We only consider non-linearity to appear in the bilevel optimization problems we study in Chapter 3. The objective and constraint functions for the two-stage robust optimization problems in Chapter 4 and Chapter 5 are all linear. In this linear case, we deal with MILPs, which are already NP-hard [158]. With non-linearity, the problems are in general nonconvex, which implies there can be multiple locally optimal solutions; finding a globally optimal solution is also NP-hard, already for problems with only continuous variables. This makes mixed-integer non-linear programs (MINLP) harder than MILPs

[78]. For an overview of solvers for MINLPs, such as Benders decomposition and branch and bound, we refer the reader to [78].

BILEVEL OPTIMIZATION (BILO)

Bilevel optimization (BiLO) deals with hierarchical problems where one agent 'the *leader*' makes decisions $\mathbf{x} \in \mathcal{X}$ and optimizes their problem given the best response of another agent, 'the *follower*', $\mathbf{y} \in \mathcal{Y}$; the sets $\mathcal{X}$ and $\mathcal{Y}$ represent the domains of the variables (continuous, mixed-integer, or pure integer). This problem structure occurs in many different domains; for example, in transportation network design where a city builds roads to minimize the average travel time of its inhabitants. The city is the leader and the group of inhabitants is the follower. Another example arises in security, where someone decides on safety measures to undermine the activities of criminals. The person taking safety measures is the leader and the criminal is the follower. And there are many other applications [59]. Generally, a problem in which the decision maker is awaiting a response from someone else can be formulated as a multi-level optimization problem, where bilevel indicates that two levels are considered. The origin of bilevel optimization stems from works such as Stackelberg games [207, 208] and the first formulation from Bracken and McGill [42]. For a thorough introduction to bilevel optimization, we refer to [59]. The nested problem structure of BiLO is formulated as follows:

$$\begin{aligned} \min_{\mathbf{x}\in\mathcal{X},\mathbf{y}} \quad & F(\mathbf{x},\mathbf{y}) \\ \text{s.t.} \quad & G(\mathbf{x},\mathbf{y}) \geq \mathbf{0}, \\ & \mathbf{y} \in \arg\max_{\mathbf{y}'\in\mathcal{Y}} \{f(\mathbf{x},\mathbf{y}') : g(\mathbf{x},\mathbf{y}') \geq \mathbf{0}\}, \end{aligned} \tag{BiLO}$$

where the upper-level problem is the optimization problem of the leader, consisting of the objective function $F(\cdot)$ and constraints $G(\cdot)$. The lower-level problem is the optimization problem of the follower, with $f(\cdot)$ as its objective function and $g(\cdot)$ as its constraints. The constraint with the argmax term over the follower's problem illustrates that the leader considers an optimal solution $\mathbf{y}$, that is not necessarily unique. It also shows that the leader's decisions $\mathbf{x}$ parameterize the follower's problem; the follower responds to the leader. In this thesis, the so-called *optimistic* setting is considered: if the follower has multiple optima for a given leader's decision, the one that optimizes the leader's objective is implemented. Therefore, in the upper level we also optimize over $\mathbf{y}$ which comprises all the optimal solutions of the lower level. We consider the general mixed-integer non-linear case with $F, f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, $G : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{m_1}$, and $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{m_2}$ non-linear functions of the upper-level $\mathbf{x}$ and lower-level variables $\mathbf{y}$.

As an illustration, in Example 1.1 the *knapsack interdiction* problem is given. Interdiction problems are a special case of bilevel optimization where the leader disrupts the actions of the follower such that the objective functions of the two members are in direct opposition [60].

Example 1.1 (Bilevel Knapsack Interdiction Problem)**.** *The knapsack problem is a classic and well-studied problem in combinatorics and integer programming. As an example of BiLO, the knapsack interdiction problem as described by Tang et al. [195] is presented. The leader decides to interdict (a maximum of k) items of the knapsack solved in the follower's*

problem, n the number of items, p_i the profits, a_i the cost of item i, respectively, and the budget of the knapsack is b. The decisions of the leader are formulated as binary variables $\mathbf{x}$, where $x_i = 1$ if the follower (with decisions $\mathbf{y}$) cannot select the i-th item of the knapsack, i.e., $y_i = 0$. This is caused by the constraint $y_i + x_i \leq 1,\ i \in [n]$, in the follower's problem of the formulation:

$$\begin{aligned}
\min_{\mathbf{x}\in\{0,1\}^n,\mathbf{y}} \quad & \sum_{i=1}^{n} p_i y_i \\
s.t. \quad & \sum_{i=1}^{n} x_i \leq k, \\
& \mathbf{y} \in \operatorname*{arg\,max}_{\mathbf{y}'\in\{0,1\}^n} \quad \sum_{i=1}^{n} p_i y'_i \\
& \qquad s.t. \quad \sum_{i=1}^{n} a_i y'_i \leq b, \\
& \qquad\qquad y'_i + x_i \leq 1,\ i \in [n],
\end{aligned}$$

Solution approaches of BiLO. If the follower's problem has only continuous variables, the bilevel problem can be reformulated into a single-level problem by applying Karush-Kuhn-Tucker (KKT) conditions or duality techniques. When it contains integer variables, the problem becomes considerably more challenging. In this case, branch and cut [61, 76] is most appropriate for general linear problems. However, these do not scale well for larger instances. Kleinert et al. [118] survey more exact methods for bilevel optimization. Many problem-specific algorithms have been introduced. Bilevel optimization knows a rich literary history of meta-heuristics, as reviewed in [183]. Bilevel optimization also knows many applications in machine learning (e.g., hyperparameter optimization [106] and neural architecture search [138]), usually without constraints and with continuous variables. Gradient-based methods are often used in this setting.

ROBUST OPTIMIZATION (RO)

In robust optimization, the outer problem optimizes over decisions whereas the inner problem optimizes over uncertain parameters:

$$\begin{aligned}
\min_{\mathbf{x}\in\mathscr{X}} \quad & \max_{\boldsymbol{\xi}\in\Xi} c(\boldsymbol{\xi})^\top \mathbf{x} \\
\text{s.t.} \quad & T(\boldsymbol{\xi})\mathbf{x} \leq h(\boldsymbol{\xi}), \quad \forall \boldsymbol{\xi} \in \Xi,
\end{aligned} \tag{RO}$$

where $\mathbf{x} \in \mathscr{X} \subseteq \mathbb{R}^n$ are the decisions and the uncertain parameters $\boldsymbol{\xi}$ are assumed to be contained in a convex and bounded uncertainty set $\Xi \subseteq \mathbb{R}^q$. The coefficients of the objective function and constraints of the problem, $c(\boldsymbol{\xi}) \in \mathbb{R}^n$, $T(\boldsymbol{\xi}) \in \mathbb{R}^{r\times n}$, and $h(\boldsymbol{\xi}) \in \mathbb{R}^r$, may depend on the scenario $\boldsymbol{\xi}$. The intuition behind this problem is the following: the decision maker wants to find the best response to the worst-case scenario while the constraints are simultaneously satisfied for all scenarios. Hence, to solve the problem in a robust setting. Generally, the scenarios are assumed to be continuous and when this is the case, the problem can be transformed into a mathematically tractable formulation that uses the dual problem of the uncertainty set [21]. This is comparable to the techniques used for bilevel optimization when the follower's problem only has continuous

variables. In both cases, the problem becomes a single-level, or regular, MIP which is constructed by using KKT or duality techniques. Another approach is constraint generation [154] where iteratively violating scenarios are added to the problem until none pertain.

TWO-STAGE ROBUST OPTIMIZATION (2RO)

For two-stage robust optimization, another inner problem is added, where after the uncertain parameters 'are realized', another set of decisions are made:

$$\begin{aligned} \min_{\mathbf{x}\in\mathscr{X}} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\mathscr{Y}} \quad & c(\boldsymbol{\xi})^\top \mathbf{x} + d(\boldsymbol{\xi})^\top \mathbf{y} \\ \text{s.t.} \quad & T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y} \leq h(\boldsymbol{\xi}), \end{aligned} \tag{2RO}$$

where $\mathbf{x} \in \mathscr{X} \subseteq \mathbb{R}^n$ and $\mathbf{y} \in \mathscr{Y} \subseteq \mathbb{R}^m$ are referred to as the here-and-now (or first-stage) and wait-and-see (or second-stage) decisions, respectively, and the uncertain parameters $\boldsymbol{\xi} \in \Xi \subset \mathbb{R}^q$. The objective function and constraints of the problem, $c(\boldsymbol{\xi}) \in \mathbb{R}^n, d(\boldsymbol{\xi}) \in \mathbb{R}^m, T(\boldsymbol{\xi}) \in \mathbb{R}^{r\times n}, W(\boldsymbol{\xi}) \in \mathbb{R}^{r\times m}$, and $h(\boldsymbol{\xi}) \in \mathbb{R}^r$, may again depend on the scenario $\boldsymbol{\xi}$. This problem structure allows for making another round of decisions after the scenario is known, in a way, to adapt to the revealed uncertain parameters. This problem can be generalized to *multi-stage* robust optimization, where the min-max-min structure is repeated such that multiple stages of decisions and uncertainties are allowed. For instance, multi-day scheduling problems can be formulated this way, where daily inventory and delivery decisions depend on the uncertain daily demand.

Example 1.2 (Two-Stage Robust Facility Location Problem)**.** *As a classic example of 2RO, consider the two-stage robust facility location problem. A planner must open a subset of n facilities to serve m clients with uncertain demands while minimizing the cost of opening the facilities and serving the clients. A facility with capacity C_i can be opened at cost c_i. A customer has a demand $b_j(\boldsymbol{\xi})$ that depends on an uncertain scenario $\boldsymbol{\xi}$. Specifically, for a nominal demand $\bar{b}_j$ and some deviation Δ_j, the demand is $b_j(\boldsymbol{\xi}) = \bar{b}_j + \Delta_j \cdot \xi_j$ where we assume there is a budget B on the total percental amount of deviation, modeled by the budgeted uncertainty set $\Xi = \{\boldsymbol{\xi} \in [0,1]^m : \sum_{j=1}^m \xi_j \leq B\}$. Facility i can serve customer j at cost d_{ij}. The planner must decide which facilities to open before the uncertain demands are observed. Allocating facilities to customers is done after the uncertain (demand) scenario is realized. This gives the following formulation:*

$$\begin{aligned} \min_{\mathbf{x}\in\mathscr{X}} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\mathscr{Y}} \quad & \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \mathbf{y} \\ s.t. \quad & \sum_{j=1}^m y_{ij} \cdot (\bar{b}_j + \Delta_j \cdot \xi_j) \leq C_i \cdot x_i, && \forall i \in \{1,\dots,n\}, \\ & \sum_{i=1}^n y_{ij} = 1, && \forall j \in \{1,\dots,m\}. \end{aligned}$$

Here, $\mathscr{X} = \{0,1\}^n$ and entry $x_i = 1$ if facility i is opened in the first stage; $\mathscr{Y} = \{0,1\}^{n\times m}$ and $y_{ij} = 1$ if facility i is assigned to customer j in the second-stage. The constraints enforce that the facility capacities are not exceeded and ensure that each customer is served by

exactly one facility. The planner is concerned with finding the subset of facilities that minimizes the total cost over all scenarios while ensuring the selected facilities can serve the customers for any demand scenario, i.e., the first-stage decision $\mathbf{x}$ *gives a feasible second-stage decision* $\mathbf{y}$ *for all* $\boldsymbol{\xi} \in \Xi$.

Solution approaches of 2RO. Both single- and multi-stage robust mixed-integer problems are NP-hard even when their deterministic counterparts are polynomial-time solvable [46], where two-stage robust problems are much harder to solve than single-stage robust problems, especially when the second-stage decisions are integer. In fact, as Goerigk et al. [88] show, multi-stage robust optimization with budgeted uncertainty set with S stages is Σ^p_{2S-1}-complete, while other forms stay in NP. When dealing with integer first stage and continuous second stage, column-and-constraint generation (CCG) is one of the key approaches [197, 227]. While an extension of CCG has been proposed for mixed-integer recourse [232], it is often intractable and does not apply to pure integer second-stage problems. We use this CCG framework in Chapter 4 to both accelerate the method and also generalize it to mixed-integer recourse. A more thorough literature review on existing solvers for linear recourse will be given in this chapter as well.

Besides these exact solution methods, one heuristic approach is *K-adaptability* [28, 95, 189]. This is a relaxation of the problem in which the uncertainty set Ξ is (indirectly) partitioned into a pre-determined number of K subsets. A separate copy of the second-stage decisions $\mathbf{y}$ is then assigned to each subset. This means that the j-th copy of $\mathbf{y}$ gives a feasible and optimal worst-case solution to all scenarios of the j-th uncertainty subset. In Chapter 5, we introduce a learning-based approach to accelerate the branch-and-bound algorithm of Subramanyam et al. [189].

1.3.2. CONTRIBUTIONS OF PART I

The following are the main contributions of the first part of the thesis.

Machine-learning based single-level formulation for generic bilevel optimization problems: In Chapter 3, the method NEUR2BILO is introduced. This method leverages neural network predictions to transform the bilevel problem into a single-level problem. By eliminating the nested problem structure, the problem becomes much less complex. This becomes evident in the empirical study on four benchmark problems where NEUR2BILO performs approximately 100 to 1000 times faster than the state-of-the-art solvers for the harder instances while obtaining high-quality solutions for all. For training data generation, an efficient data generation approach is proposed based on only solving one of the nested problems. The procedure, performed offline, uses the solutions of the lower-level problem, an MIP. Solving this is considerably less complex than the bilevel problem, regardless of the complexity of the lower-level problem itself.

Learning-based formulation of the column-and-constraint-generation algorithm for two-stage robust optimization problems: In Chapter 4, the method NEUR2RO is introduced. This method, like NEUR2BILO, leverages neural network predictions to simplify the problem formulation such that fewer variables and constraints are required. However, NEUR2RO has a more intricate structure as it involves an iterative algorithm

(unlike the single-level formulation of NEUR2BILO) where the main and sub-problem of column-and-constraint generation (CCG) [227] are amended with neural network embeddings. This method has shown to be performing better than state-of-the-art for large instance sizes of two benchmark problems, with significant computational speed-ups while obtaining similar or better solutions. Training data generation takes a similar approach as in Chapter 3 where the solutions of the second-stage problem are used. Hence, also the solutions of only one nested problem are required. This problem is an MILP and therefore much easier to solve than the two-stage robust optimization problem.

An efficient modular neural network architecture for NEUR2RO: For the algorithm proposed in Chapter 4, a specific machine-learning model is proposed, as more generic ones would not suffice. Due to the iterative nature of CCG, repeatedly embedding neural networks would become problematic due to the extra variables and constraints required for NN embeddings. For NEUR2RO, a tailored architecture is proposed that uses a modular structure, such that only parts of the learned model cumulatively need to be embedded in the two problems.

Learning-based node selection strategy to accelerate the K-adaptability B&B algorithm for solving two-stage robust optimization problems: In Chapter 5, we propose a learning-based method to more effectively traverse the search tree of the K-adaptability branch-and-bound (B&B) algorithm [189]. The B&B algorithm does not scale well for higher instance sizes. Finding better solutions faster is dictated by the order in which one traverses through the search tree. The learning-based node-selection strategy we propose finds similar or better solutions up to 90% faster compared to a default random node-selection strategy.

A training data generation scheme inspired by Monte-Carlo Tree Search: A specialized training data generation scheme is proposed in Chapter 5 to represent the state space of the K-adaptability B&B search tree accurately. This method is also instance-size independent, which makes it possible to train on small instances and generalize to larger instances. This approach leverages the structure of Monte-Carlo Tree Search to estimate and assign labels to the data points, as assigning exact values is computationally intractable.

1.4. PART II: PHYLOGENETICS

The second part of this thesis continues with the combination of machine learning and optimization, but now for problems of a very different nature, arising in *phylogenetics*. This field studies evolutionary relationships among various organisms, using graph structures to represent the evolutionary events over time. Due to the complexity of inferring information from biological data onto graphs, this field has attracted many mathematicians and computer scientists. In what follows, a simplified biological background is given of the phylogenetic combinatorial problems studied in this thesis. First, phylogenetic *trees* are defined and subsequently phylogenetic *networks*, to which the main contributions pertain. Deriving an *optimal* network is known to be NP-hard for almost

all known problem formulations, and consequently *exact* methods do not scale well for larger instances. We explore the role of machine learning in creating good networks, which has not been done before. In recent years, machine learning has been used for reconstructing trees, primarily used for faster and accurate computations needed for tree inference [9, 10]. Like these approaches, we aim at methods that scale well and are widely applicable. However, we focus on constructing networks rather than trees. We do not apply ML directly to biological data, as is usually done for the generation of trees, but rather combine it with mathematical characterizations like *cherry-picking sequences*, which will be introduced later.

1.4.1. TREES

A *rooted phylogenetic tree* is a directed acyclic graph with a root (i.e., a node with in-degree 0 and out-degree more than 1), and labeled leaves (i.e. nodes of in-degree 1 and out-degree 0). These leaves represent taxa, which can be various groups of organisms, like different groups of plant species, as illustrated in Figure 1.1a. A phylogenetic tree shows a (hypothetical) evolutionary history by representing speciation events at every tree node (i.e., all other nodes than the root and leaves are of in-degree 1 and out-degree more than 1). Such an event symbolizes the evolution from one ancestral taxon into (at least two) transformed taxa caused by, for instance, geographical isolation of one group of the species or natural selection. A tree is *binary* when the root and tree nodes have an out-degree of exactly two, it is *non-binary* if the out-degree is more than two for at least one tree node or for the root. Even though it is in many cases unlikely that a speciation event creates more than two new species, non-binary trees are still beneficial as they can reflect uncertainty in the order of speciation events [122].

TREE RECONSTRUCTION METHODS

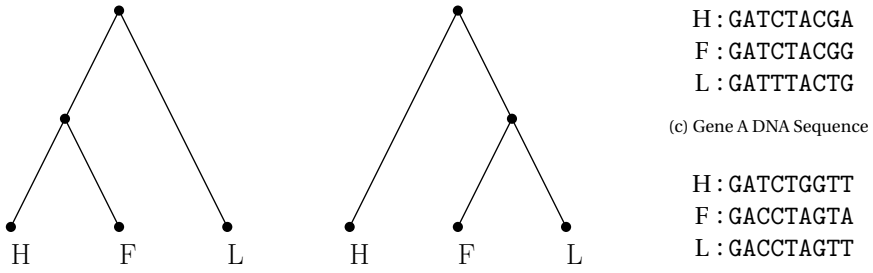
A phylogenetic tree is constructed by comparing the biological data (e.g., DNA sequences, protein sequences, or ecological traits) of various taxa. Commonly used phylogenetic tree reconstruction methods are of the following types. Distance-based methods (e.g., Neighbor-joining and unweighted pair group method with arithmetic mean, UPGMA) use pairwise distances of sequence alignments. These distances are for example used to heuristically obtain a tree by iteratively clustering the closest pair. More exhaustive search methods are also possible. Instead of summarizing distance metrics, character-based methods (e.g., maximum likelihood, maximum parsimony, and Bayesian inference) rather focus on each character of the sequence alignment. Maximum likelihood and Bayesian interference are statistical methods that search the parameter space (e.g., tree topologies and branch lengths) and select the best one(s) based on the likelihood of the parameter values. This likelihood is the probability that the evolutionary model given these parameters correctly models the underlying sequence alignment. For details on the above mentioned methods, among others, we refer the reader to Nei and Kumar [157] and Felsenstein [72]. Maximum likelihood methods and Bayesian inference are usually computationally intractable [55] and can only find optimal solutions for limited-sized datasets. To initialize the parameters for maximum likelihood methods, a tree can also be constructed using distance-based methods. Recently, machine-learning techniques have been introduced to accelerate the search for trees

with a high likelihood score [9, 10]. Other methods have also been introduced for tree inference, where machine learning and a more combinatorial approach is are used [234].

A PHYLOGENETIC FOREST

For some of the problems we solve in this thesis, the input is a *forest* or *set of phylogenetic trees* on the same set, or on similar sets, of taxa. Various trees can be reconstructed from the same set of organisms by using different methods, or, for example, by creating so-called *gene trees* based on different data. A gene is a segment of DNA that carries hereditary information, which includes encoding instructions used for building proteins or for supporting other genes. A gene tree is based on the DNA sequence of one specific gene present in all, or most, of the organisms studied. The evolutionary history of the species can be different from the gene trees if there is *incomplete lineage sorting* [58]. We however assume that this phenomenon does not occur in the data we study.

In the following example, two gene trees are constructed for three groups of plant species. The two trees in Figure 1.1 are not isomorphic; different evolutionary histories are shown. Whenever there is a conflicting set of trees on the same taxa, while assuming incomplete lineage sorting does not occur, it can suggest several things: incorrect data, inaccuracy of the method, or occurrences of complex evolutionary events that trees cannot express. Such events can be represented by *phylogenetic networks*, which will be discussed after the example.



(a) Gene A Tree (b) Gene B Tree (c) Gene A DNA Sequence (d) Gene B DNA Sequence

Figure 1.1: Gene trees of the (fictional) A and B genes for Hornworts (H), Ferns (F), and Lycophytes (L) groups of plant species. Example 1.3 demonstrates the reconstruction of the phylogenetic trees using the sequence alignment. These trees are inspired by the network created in Soucy et al. [187, Figure 5].

Example 1.3 (Gene Trees on Plant Species)**.** *We study two phylogenetic gene trees displaying the evolutionary relationships between three groups of plant species: Hornworts, Ferns, and Lycophytes. The first tree is based on the (fictional) A gene and the second tree on the (fictional) B gene. The resulting trees along with the (fictional) DNA sequences of each gene and taxon are given in Figure 1.1. We briefly show the high-level idea of how the trees can be derived via the distance-based method UPGMA. The (Hamming) distance between each pair of species is determined by simply determining the number of conflicts in the sequence alignment and then grouping the pair with the fewest conflicts (i.e., the closest) in the tree. Since only three taxa exist in this example, the tree can be constructed after one step. Otherwise a process would follow that repeatedly calculates a new distance*

1

matrix explaining the updated state as leaves are grouped at each step. For the A gene, we see that the DNA sequences of Hornworts and Ferns are the pair with the smallest distance. For the B gene, Ferns and Lycophytes are the closest pair. These pairs are grouped first in their respective gene trees.

1.4.2. NETWORKS

Non-treelike events representing complex evolutionary processes are called *reticulations*. Such events characterize *merging* of species, whereas the speciations in trees characterize *diverging* events. *Phylogenetic networks* can express both types in a directed acyclic graph, keeping the description of trees as described before with additional *reticulation nodes* (i.e., a node of in-degree more than 1, and out-degree of 1), symbolizing the merging of ancestors. Reticulations can, for example, represent the evolutionary events of *hybridization* (e.g., hybrid speciation and introgression), *lateral gene transfer* (LGT) (e.g., transformation, transduction, and conjugation [187], primarily observed in bacteria and archaea), and *recombination* [122]. In Figure 1.2, an example of a network on three plant taxa displaying a lateral gene transfer is given.

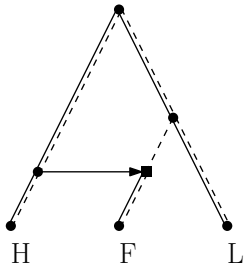


Figure 1.2: A phylogenetic network on three groups of plant species: Hornworts (H), Ferns (F), and Lycophytes (L). The circular nodes indicate the root, tree nodes, and leaf nodes. The square node is a reticulation node. The solid arcs show an embedding of Gene A Tree and the dashed arcs an embedding of Gene B Tree of Figure 1.1. This network is a subnetwork of the one reconstructed in Soucy et al. [187, Figure 5]. The directed horizontal (reticulation) arc corresponds to a lateral gene transfer from Hornworts to Ferns.

Many solution approaches for generating a network are based on *multilocus* data (i.e., sequence data of multiple genes, or loci). Some of the methods used to reconstruct trees are generalized to the reconstruction of networks. This includes maximum parsimony [156], maximum likelihood [224], and Bayesian inference [212], contained in `phylonet` [213] and `phylonetworks` [186]. These methods were already computationally expensive for reconstructing trees and their scalability has only worsened in this new setting. Such approaches are generally restricted to a small number of reticulations and taxa. The measures are still useful for comparing candidate networks by assigning performance scores to each, such as the likelihood or parsimony score.

Instead of directly inferring a phylogenetic network from multilocus data, many methods for reconstructing networks are based on combining sampled gene trees. While there exist more complex models that can identify hybridization events despite incomplete lineage sorting [148, 225], we will focus on the combinatorial variant that aims at constructing a network that *displays* all its gene trees. To illustrate, in Figure 1.2 a net-

work is given where the labeled arcs indicate how the two gene trees of Figure 1.1 are displayed: Gene A Tree via the solid arcs and Gene B Tree via the dashed ones.

NETWORK RECONSTRUCTION WITH GENE TREES AS INPUT

This thesis focuses on combinatorial optimization methods for inferring rooted phylogenetic networks from estimated gene trees. Following the parsimony principle, it is common to search for networks with as few reticulations as possible, as reticulation events are often considered relatively uncommon. The objective of our combinatorial problem is then to minimize the number of reticulations in the resulting network, while still displaying all the gene trees. This brings us to the informal description (see Section 6.2 for the formal definitions) of the associated computational problem: HYBRIDIZATION. To illustrate, the trees of Figure 1.1 can serve as the input, and the network of Figure 1.2 is a corresponding output of this problem.

HYBRIDIZATION
Input: A set of phylogenetic trees on similar, but not necessarily equal, sets of taxa.
Output: A phylogenetic network displaying the trees with minimum possible reticulation number.

This problem is already NP-hard for two binary trees with equal taxon sets [39]. However, the authors of this paper show that this problem is fixed-parameter tractable (FPT) and van Iersel and Linz [206] generalize this result by proving this holds for binary tree sets of any size. FPT problems can be solved by a computationally tractable algorithm with the restriction that some problem parameters are fixed. These results have given rise to the construction of numerous other algorithms, many of which are based on *agreement forests* [17, 37, 56, 215], all for two binary trees on the same taxon set. Exceptions of algorithms that share these assumptions are `hybroscale` [3], which can handle an arbitrary-sized tree set, and the algorithm of Huson and Linz [102], which applies to two non-binary trees on overlapping, but not equal, taxon sets. On a high level, an agreement forest is a set of subtrees on a subset of taxa, called components, which appear in each tree and are thus in agreement. Under some acyclicity assumption of the agreement forest, the reticulation number of the resulting phylogenetic network is equal to the number of components minus one [15]. To obtain the minimum reticulation number, the minimum sized forest is soughtf, which is called the *maximum acyclic agreement forest*.

Other combinatorial methods are, for example, Piovesan and Kelk [168] who propose an FPT algorithm for two non-binary trees on the same taxa, and CASS [205], which can be applied to generic inputs, but only for small reticulation numbers. The algorithm `PIRN` [220] can be applied to multiple non-binary trees and `PIRNs` [149] on multiple binary trees, both on the same sets of taxa. The latter two methods do not scale well for large taxon sets. There also exist many other methods that make restrictive assumptions on the network [103] and/or the input set [135, 204] and will due to their impracticality to general input trees not be discussed further.

As far as we know, there is no algorithm or heuristic that can handle tree sets of more than two multifurcating (i.e., non-binary) trees with overlapping, but not exactly equal, leaf sets. The proposed class of heuristics (in Chapter 6 and Chapter 7) solves exactly these types of instances, in which the mathematical characteristic of *cherry picking* plays a vital role.

Some years ago, the notion of cherry picking was introduced [101, 134]. This characterization has proven to be very effective for reconstructing phylogenetic networks on large instances. van Iersel et al. [201] propose an exact algorithm for reconstructing 'tree-child' networks using cherry picking. Their algorithm was applied to instances of up to 100 binary trees with equal taxon sets but can only handle reasonably small reticulation numbers. If the input trees are non-binary, a preprocessing heuristic makes them binary. However, the cherry-picking characterization is more general than assumed in their study; it can also directly be applied to multifurcating trees and to mostly overlapping, but not exactly equal, sets of taxa.

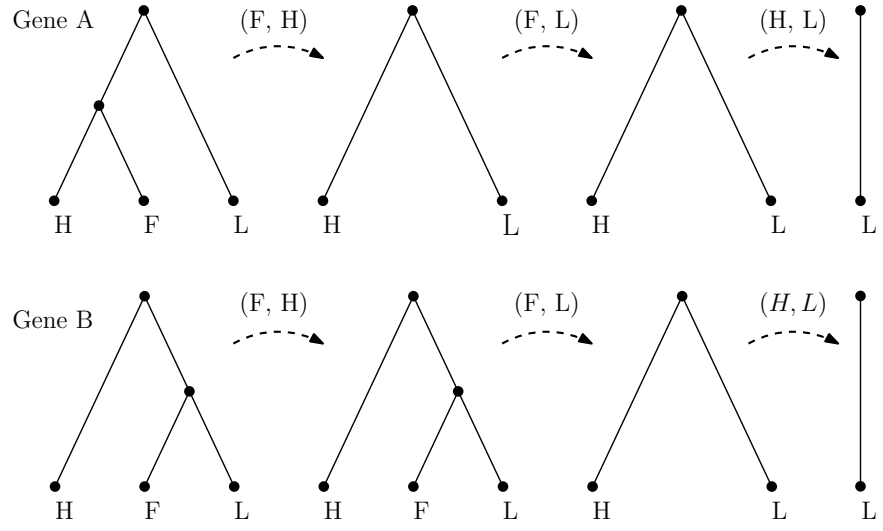
CHERRY PICKING

A cherry is an *ordered* leaf pair with a common parent. The name 'cherry picking' is inspired by the shape its incoming arcs depict. 'Picking' cherry (a, b) entails deleting leaf a and its incoming arc, and 'cleaning' everything up (i.e., suppressing the resulting in-degree 1 and out-degree 1 node). One can 'reduce' a tree by iteratively picking different cherries until only one leaf remains. Once all the trees are (simultaneously) reduced, a network can be constructed using the sequence of cherries used to reduce the trees. Formal definitions of these moves are given in Section 7.2.2. See the following example for a demonstration on plant gene trees.

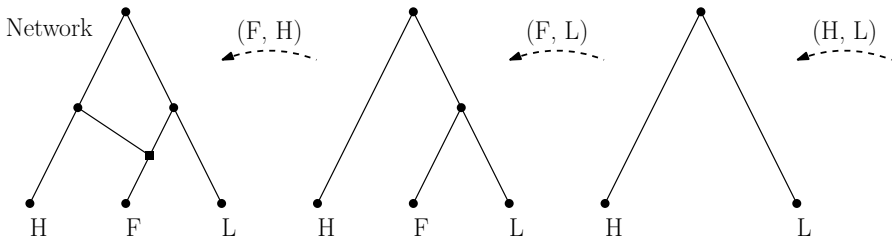
Example 1.4 (Cherry Picking Applied to Plant Trees)**.** *In this example, we show how the network in Figure 1.2 can be reconstructed from the gene trees in Figure 1.1by using cherry picking. The cherry-picking steps are illustrated in Figure 1.3a. where the cherries are simultaneously picked from the trees. If the cherry is not present in a tree, the tree remains the same. Suppose we pick cherry* (F, H) *first, thus* F *is deleted in Gene A Tree, and nothing changes for Gene B Tree. Then suppose cherry* (F, L) *is picked, present in Gene B Tree only. Finally, suppose cherry* (H, L) *is picked in both trees, leaving one leaf. The tree set is now fully reduced.*

Now that we have a cherry-picking sequence $S = (F, H), (F, L), (H, L)$*, we can reconstruct a network. This is done in Figure 1.3b. From right to left, we first initialize the network with cherry* (H, L)*. Then, we add* (F, L) *by adding* F *to* L*. Note that now all the leaves are in the network. However, one cherry,* (F, H)*, is left. This cherry is added by placing an arc from (the incoming arc of)* H *to (the incoming arc of)* F*, creating a node with in-degree 2 and out-degree 1: the reticulation node. Note that different choices may lead to a different network.*

In Example 1.4 we assumed to know which cherries to pick. In practice, this is not the case. The number of possible actions can be high; it corresponds to the number of unique cherries across all the trees, which can become very large as the tree set and taxon set(s) increase in size. This creates the necessity for an algorithm or heuristic to guide these decisions. In Chapter 6, we present a class of heuristics for reconstructing



(a) Cherry picking of gene trees



(b) Network reconstruction

Figure 1.3: A simulation of cherry picking for the example of three taxa: Hornworts (H), Ferns (F), and Lycophytes (L). In (a) the two gene trees are simultaneously reduced by cherry picking, resulting in the sequence $S = (F,H),(F,L),(H,L)$. In (b) this sequence is used to reconstruct the network (from right to left). Example 1.4 gives a more detailed description of the reduction.

phylogenetic networks for binary trees, where cherry picking plays the central role. Then in Chapter 7, this heuristic is tailored to handle non-binary trees with missing leaves (i.e., the taxon sets per tree are mostly overlapping but not equal). In the heuristics, iteratively, one of the available cherries is picked, until all trees are reduced. We aim to find the *shortest* cherry-picking sequence that reduces all trees as the length of the sequence directly affects the number of reticulations in the constructed network [202]. Thus, we must choose the *right* cherry in each iteration of the heuristic. One approach to deciding which cherry to pick is via a machine-learning prediction.

ORCHARD NETWORKS

As demonstrated earlier, applying cherry picking to 'merge' gene trees reconstructs a phylogenetic network. These networks belong to the *orchard* class, defined by their construction and reduction through a cherry-picking sequence [70, 109]. Apart from the computational implications, this class also has biologically motivated structures. Namely, an orchard network can be described as a tree with additional horizontal arcs [200]. The inverse is also true: each network with only lateral gene transfer (LGT) events is an orchard network, assuming there are no unsampled taxa. This brings us to the last chapter of Part II, Chapter 8, where we answer the following question: what is the minimum number of leaves that needs to be added to a network to make it orchard? This question could be relevant when one knows that all reticulations are caused by LGT events but the reconstructed network is not orchard. The studied question would then answer how many unsampled taxa prevent the network from only depicting horizontal events. We show this problem turns out to be NP-hard. The proof and an MILP formulation for solving this problem are introduced in Chapter 8. The results of the MILP model also give an hypothesis of the locations of the unsampled taxa in the network (i.e., to which arcs the auxiliary taxa can be added to make the network orchard).

Linz and Semple [134] propose adding leaves to trees to construct an optimal *arbitrary* network, and not an optimal 'tree-child' network. While our methods do not directly apply to this setting, they have inspired us with ideas for future work related to this concept. This discussion is reserved for Chapter 9.

1.4.3. CONTRIBUTIONS OF PART II

The following are the main contributions of the second part of the thesis.

An efficient and widely applicable class of cherry-picking heuristics for solving the HYBRIDIZATION problem on non-binary gene trees with missing leaves: In Chapter 6 we introduce a class of heuristics that combine cherry picking and machine learning to solve the HYBRIDIZATION problem for binary gene trees. We design a machine-learning model for predicting the correct cherries to pick in each iteration, based on features that describe the structure of the tree set. In Chapter 7, we apply this heuristic to instances of non-binary trees that may have missing leaves. The machine-learning element is changed to better handle this general class of inputs. We propose to add another predictor solely for the leaves before predicting the correct cherries, as the number of cherries in multifurcating trees with missing leaves grows quadratically in the number of taxa, compared to linearly for binary trees. We show that this method is practical as

demonstrated by the experiments on synthetic and real data of up to 100 trees and 100 leaves. Especially for large instances, the experiments show that using machine learning to search the solution space leads to good performances.

An efficient data generation scheme for training the ML models of the heuristic, based on creating labels by obtaining instances from solutions: For the heuristic proposed in Chapter 6, we propose a training data generation scheme that utilizes the relationship between phylogenetic networks and their displayed trees. From synthetically generated networks, which could be interpreted as the solution to the HYBRIDIZATION problem, a corresponding problem instance can be built by (randomly) selecting the network's displayed trees. In this way, "expert" data can be derived easily without having to construct an optimal network for a given instance.

The operation TREE EXPANSION when picking 'trivial' cherries which reduces the creation of unnecessary new cherries in intermediate steps: To improve the heuristic proposed in Chapter 6 with a simple deterministic step, this operation is proposed. When picking 'trivial' cherries (i.e., cherries occurring in every tree where both leaves are present), sometimes unnecessary complications in intermediate steps can arise, leading to higher reticulation numbers. We experimentally show that the obtained cherry-picking sequences are shorter when performing the TREE EXPENSION operation and prove that the resulting network is always feasible (i.e., the original trees are displayed in the network). We believe this operation can be utilized outside our specific framework and could therefore be seen as an independent contribution.

An NP-hardness proof of the orchard leaf distance problem: In Chapter 8, we define the proximity measure of the minimum number of leaves that need to be added to make a network orchard. We prove this measure is NP-hard to compute by giving a polynomial time reduction from DEGREE-3 VERTEX COVER.

An MILP formulation for the orchard leaf distance problem that labels reticulation arcs as horizontal or vertical: In Chapter 8, we reformulate the orchard leaf distance problem by using another definition of orchard networks: as trees with horizontal arcs. We can label all the reticulation arcs in the network to determine which are horizontal and which are not. To make the network orchard, a leaf is added to one of the incoming arcs for each reticulation with only vertical incoming arcs. The labeling is not unique, and to minimize the number of vertical reticulation arcs, we model this problem as a mixed integer linear program. Experimental results show that this method is very efficient for the real (with up to 32 reticulations) and simulated (up to 200 reticulations) networks we studied with most instances finishing within a fraction of a second.

I

Hierarchical Optimization

2

PRELIMINARIES

In this part, the three works on hierarchical optimization are presented. The first two chapters make use of value function approximations, where Chapter 3 introduces NEUR2BILO, a learning-based method for bilevel optimization problems, and Chapter 4 introduces NEUR2RO, a learning-based method for two-stage robust optimization problems. For both methods, a *trained* neural network, or any mixed integer linear program (MILP) representable machine-learning model, is embedded in the mixed integer program (MIP) as a substitution of the value function it aims to predict. This can entail (parts of) the objective value function and/or the constraints. In Section 2.1, we provide more details of this technique and discuss how the designed neural network architecture can generalize to larger instances while training on smaller ones.

2.1. VALUE FUNCTION APPROXIMATION

Integrating a machine-learning model into an MIP is a vital element of NEUR2BILO and NEUR2RO. Both methods use a neural network. The initial (and offline) step is obtaining training data and training a neural network. Then, this *trained* model is embedded into an MIP. This section explains what this entails and how it works for neural networks with rectified linear unit (ReLU) activation functions. Other predictors like classification or regression trees can be used instead, as long as the ML model can be expressed as MILP variables and constraints.

APPROXIMATION-BASED MODEL FORMULATION

Consider the following *general* problem formulation:

$$\min_{\mathbf{x}\in\mathcal{X},\mathbf{y}} \quad F(\mathbf{x}) + f(\mathbf{y}) \tag{2.1a}$$

$$\text{s.t.} \quad \mathbf{y} \in \mathcal{Y}(\mathbf{x}), \tag{2.1b}$$

where $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ and $\mathbf{y} \in \mathcal{Y}(\mathbf{x}) \subseteq \mathbb{R}^m$ are variables. The feasible regions $\mathcal{X}$ and $\mathcal{Y}$ can also be pure-integer or mixed-integer and $\mathcal{Y}(\mathbf{x})$ depends on variables $\mathbf{x}$. Moreover, $F(\cdot)$ and $f(\cdot)$ are real-valued functions.

With value function approximations, we substitute part of this problem with a trained neural network. To illustrate, we substitute the objective function that corresponds with $\mathbf{y}$ given its feasible region $\mathscr{Y}(\mathbf{x})$ of Problem (2.1) with a neural network:

$$\text{NN}(\mathbf{x}) \approx \min_{\mathbf{y}} \{f(\mathbf{y}) : \mathbf{y} \in \mathscr{Y}(\mathbf{x})\}.$$

After substitution of the neural network with its predicted value results, Problem (2.1) is transformed into:

$$\min_{\mathbf{x} \in \mathscr{X}} \quad F(\mathbf{x}) + \text{NN}(\mathbf{x}). \tag{2.2}$$

The neural network takes as input the variables $\mathbf{x}$, making the variables $\mathbf{y}$ redundant. Exploring the option of deleting a part of the variable set and constraints would be interesting for problems in which f or $\mathscr{Y}$ are unknown, or when $\mathscr{Y}(\mathbf{x})$ constitutes complex constraints [53, 90, 113, 121, 123, 153, 174]. In the case of NEUR2BILO where we deal with bilevel problems, $\mathscr{Y}(\mathbf{x})$ is the solution to the follower's problem that is parameterized by $\mathbf{x}$, see (BiLO).

Then, what does optimizing over a neural network mean? Note again that we deal with a *trained* neural network (i.e., all weights and biases are fixed), that takes $\mathbf{x}$ as input and predicts $p = \text{NN}(\mathbf{x})$. If we minimize over NN we aim to find the input $\mathbf{x}$ that minimizes the prediction p. This is achievable, in part, due to MILP representations of neural networks [54, 75, 177, 196]. One can alternatively use ML-based methods for "adversarial attacks", where gradient-based methods are used [127]. This works best if optimizing over the neural network is the sole goal. When other constraints or another objective function are present, MILP representations are very effective as you can easily combine the multiple optimization components into one model.

NEURAL NETWORK EMBEDDING

In the following, we use the formulation of Fischetti and Jo [75] with a feed-forward neural network of N hidden layers. For an illustration, see Figure 2.1 where the input (layer 0), the output (layer $N+1$), and two of the N hidden layers (layers 1 and 2) are shown. Every i-th node in layer l is assigned a continuous variable ϕ_i^l and for the hidden layers ($l \in \{1,\dots,N\}$) a continuous slack variable s_i^l and a binary variable z_i^l, which is 1 if the node is 'active' and 0 otherwise. We have $\boldsymbol{\phi}^0 = \mathbf{x}$ for the input layer and a regression task is assumed in this formulation. In neural networks, non-linearity can be introduced by applying a non-linear *activation* function to the hidden nodes. This enables the model to learn complex and non-linear relationships. Often, as is also the case in the works of this thesis, ReLU functions are used. The value of each hidden node is then given as:

$$\phi_i^l = \text{ReLU}(\mathbf{w}^\top \boldsymbol{\phi}^{l-1} + b_i^l) = \max(0, \mathbf{w}^\top \boldsymbol{\phi}^{l-1} + b_i^l),$$

where $\mathbf{w}$ is the vector of weights between all the $(l-1)$-layer nodes and the i-th node of layer l (sub- and super-scripts are omitted), and b_i^l the bias (i.e., an added constant for what cannot be represented solely from the input) of node i in layer l. By introducing the variables s_i^l and z_i^l, we can reformulate the above expression in linear terms:

$$\phi_i^l - s_i^l = \mathbf{w}^\top \boldsymbol{\phi}^{l-1} + b_i^l, \quad \text{and indicator constraints} \quad z_i^l = 1 \rightarrow s_i^l = 0,\ z_i^l = 0 \rightarrow \phi_i^l = 0,$$

with additional bounds on $\boldsymbol{\phi}$ and $\mathbf{s}$. These expressions force the binary variable to be 1 when a node is 'active', i.e., when $\mathbf{w}^\top \boldsymbol{\phi}^{l-1} + b_i^l > 0$, and $\phi_i^l = \mathbf{w}^\top \boldsymbol{\phi}^{l-1} + b_i^l$. Otherwise, it is set to 0 with $s_i^l = -\mathbf{w}^\top \boldsymbol{\phi}^{l-1} + b_i^l$. Indicator constraints are formulated as linear expressions using big-M formulations. This often causes issues with implementations, which can be remedied by good bounds on the continuous variables appearing in the indicator constraints.

For the i-th position in the output layer, we have $\phi_i^{N+1} = \text{OUTPUT}(\mathbf{w}^\top \phi^N + b_i^{N+1}) = p_i$. The function $\text{OUTPUT}(\cdot)$ is an activation function that maps the output layer's input to the desired output format; e.g., for regression usually the linear function, and for classification, the softmax function is used. In Fischetti and Jo [75], the activation function of the output layer is a ReLU function. By selecting the linear function instead, we simplify the model as no slack and binary variables are required for this layer. Other output functions such as sigmoid or softmax require non-linear constraints. (Piecewise-)linear approximations can also be leveraged to avoid non-linearity in the model.

NN representations are improved [5, 90, 210] and provided as open software in [24, 51, 199]; we used `gurobi-machinelearning` [93] in the implementations of NEUR2BILO and NEUR2RO.

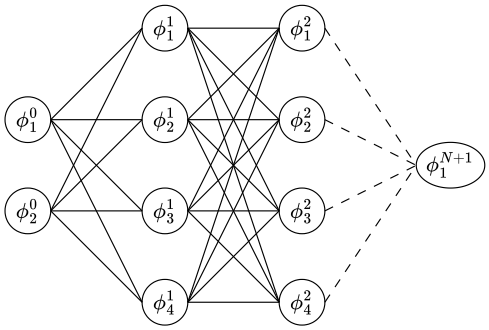


Figure 2.1: Feed-forward neural network with ReLU activations.

GENERALIZING OVER INSTANCES WITH DEEPSETS

Ideally when using machine learning in optimization, the same predictor can be used for different instances, and also of different sizes. For example, the neural network in the formulation of Problem (2.2), with input $\mathbf{x} \in \mathbb{R}^n$, can be applied to problems with different values of n. In our work, to meet this requirement we use *DeepSets* [226]: a neural network architecture design based on a *set* as input instead of a fixed dimensional vector. That is, instead of having a vector

$$[z_1, \dots z_n],$$

as input of the neural network, now we have a set

$$Z = \{z_1, \dots, z_n\}.$$

DeepSets proposes an architecture that guarantees the invariance under permutation in the order of items in the set and can handle different-sized sets. This means that when

we train an NN with this architecture, we can apply the model to instances of different-sized inputs than that of Z, and the order of the items does not influence the prediction.

The high-level idea of the architecture is as follows. Each item z of input set Z is transformed into some representation $\Phi(z)$, which is a fixed length vector. The sum of the representations of $\Phi(z)$ over all $z \in Z$ is processed using some non-linear function ρ. The prediction is then computed by:

$$p = \rho\big(\sum_{z \in Z} \Phi(z)\big), \tag{2.3}$$

which shows that Z can be of arbitrary size, as Φ is the same for all items of the set.

Alternatively, graph neural networks have been used in the field of optimization as MILP instances have an effective bipartite graph encoding [82] and combinatorial optimization often consists of graph problems already [114]. This class of neural networks can also be trained and applied to problem instances of different sizes. However, we opted for DeepSets as, at the time, NN representations for graph neural networks had not been studied extensively yet, and DeepSets turned out to be very flexible and effective for our purposes.

How is DeepSets applied in this thesis? In our work, we apply DeepSets in the following way (see Figure 2.2). For NEUR2RO and NEUR2BILO, something similar as the following set

$$\Big\{\big(x_1, \boldsymbol{\pi}^{(1)}\big), \ldots, \big(x_n, \boldsymbol{\pi}^{(n)}\big)\Big\},$$

is the input of the neural network, with x_i the variable and $\boldsymbol{\pi}^{(i)}$ a vector of instance parameters (e.g., objective and constraint coefficients) of the i-th decision. Note from (2.3) that each item of this set is input to the function Φ. We define Φ to be a multilayer perceptron (MLP), i.e., a fully connected feed-forward neural network. To include Φ in the architecture, one MLP is introduced into the NN per item of the input set that *shares* its parameters with the others (i.e., the parameters, weights and biases, of each MLP are the same). See the MLPs in the green boxes in Figure 2.2. The outputs of each $\Phi(\cdot)$ are then aggregated ($\bigoplus$). As in the DeepSets paper, we use the sum operation, but other aggregation operations can be used instead, depending on what works best for the prediction target. The same properties of order invariance and set size independence apply. This aggregated value is transformed via a non-linear function ρ, which is another MLP (that does not share its parameters with Φ). This is the red box in the figure. Since the parameters in the neural network do not change when the instance size varies, this problem can be trained and applied to problems of different sizes. To then utilize this architecture for value function approximations, the activations of Φ and ρ are set to be ReLU functions, and the model is embedded into an MIP using the MILP representations of Φ (with copies for each item) and ρ.

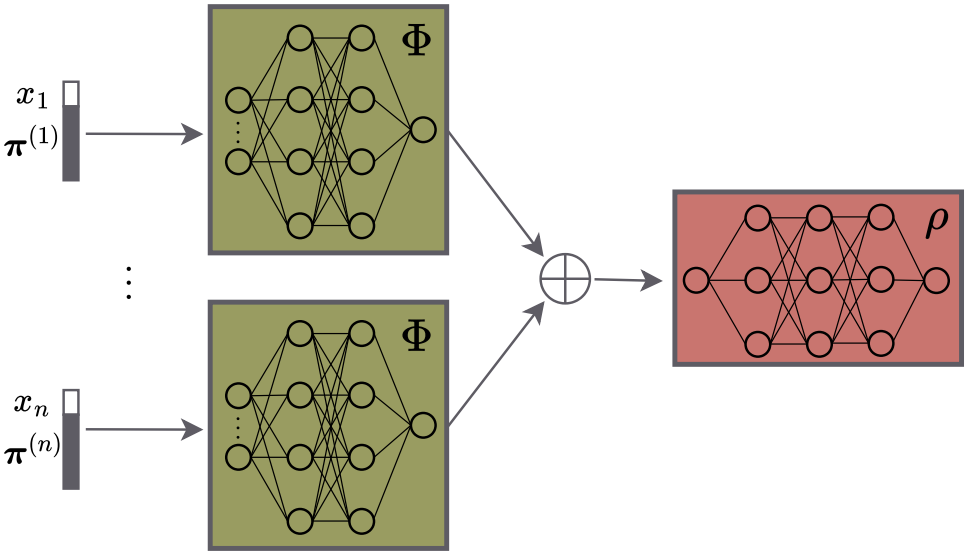


Figure 2.2: Example of DeepSets. The MLP Φ has two hidden layers and uses a parameter-sharing scheme for all inputs, the MLP ρ has three hidden layers, and $\bigoplus$ is the aggregation operation.

3

NEURAL BILEVEL OPTIMIZATION

Bilevel optimization deals with nested problems in which a leader takes the first decision to minimize their objective function while accounting for a follower's best-response reaction. Constrained bilevel problems with integer variables are particularly notorious for their hardness. While exact solvers have been proposed for mixed-integer linear bilevel optimization, they tend to scale poorly with problem size and are hard to generalize to the non-linear case. On the other hand, problem-specific algorithms (exact and heuristic) are limited in scope.

Proposed data-driven algorithm: *Under a data-driven setting in which similar instances of a bilevel problem are solved routinely, our proposed framework,* NEUR2BILO*, embeds a neural network approximation of the leader's or follower's value function, trained via supervised regression, into an easy-to-solve mixed-integer program. This method uses value function approximation, which was explained in Chapter 2.*

Data generation scheme: *For obtaining training data for the supervised regression model, for both approximation tasks, we first solve the follower's problem with a sampled leader's decision. This way, the model only requires solving many single-level problems to obtain data.*

Experimental results: NEUR2BILO *serves as a heuristic that produces high-quality solutions extremely fast. The experiments are performed for four applications with linear and non-linear objectives and pure and mixed-integer variables.*

This chapter is based on Dumouchelle et al. [65] published in **Advances in Neural Information Processing Systems 38 (NeurIPS, 2024)**. In collaboration with Justin Dumouchelle, Jannis Kurtz, and Elias B. Khalil.

3.1. INTRODUCTION

A motivating application. Consider the following *discrete network design problem* (DNDP) [171, 172]. A transportation planning authority seeks to minimize the average travel time on a road network represented by a directed graph of nodes N and links A_1 by investing in constructing a set of roads (i.e., links) from a set of options A_2, subject to a budget B. The planner knows the number of vehicles that travel between any origin-destination (O-D) pair of nodes. A good selection of links should take into account the drivers' reactions to this decision. One common assumption is that drivers will optimize their O-D paths such that a *user equilibrium* is reached. This is known as *Wardrop's second principle* in the traffic assignment literature, an equilibrium in which "no driver can unilaterally reduce their travel costs by shifting to another route" [146]. This is in contrast to the *system optimum*, an equilibrium in which a central planner dictates each driver's route, an unrealistic assumption that would not require bilevel modeling. A link cost function is used to model the travel time on an edge as a function of traffic. Let $c_{ij} \in \mathbb{R}_+$ be the capacity (vehicles per hour (vph)) of a link and $T_{ij} \in \mathbb{R}_+$ the free-flow travel time (i.e., travel time on the link without congestion). The US Bureau of Public Roads uses the following widely accepted formula to model the travel time $t(y_{ij})$ on a link used by y_{ij} vehicles per hour: $t(y_{ij}) = T_{ij}(1 + 0.15(y_{ij}/c_{ij})^4)$. As the traffic y_{ij} grows to exceed the capacity c_{ij}, a large quartic increase in travel time is incurred [146].

Bilevel optimization (BiLO) [14] models the DNDP and many problems in which an agent (the *leader*) makes decisions that minimize their cost function subject to another agent's (the *follower*'s) best response. In the DNDP, the *leader* is the transportation planner and the *follower* is the population of drivers, giving rise to the following optimization problem

$$
\begin{aligned}
\min_{\mathbf{x}\in\{0,1\}^{|A_2|},\mathbf{y}} \quad & \sum_{(i,j)\in A} y_{ij}\, t(y_{ij}) \\
\text{s.t.} \quad & \sum_{(i,j)\in A_2} g_{ij} x_{ij} \le B, \\
& \mathbf{y} \in \arg\min_{\mathbf{y}'\in\mathbb{R}_+^{|A|}} \quad \sum_{(i,j)\in A} \int_0^{y'_{ij}} t_{ij}(v)\, dv \\
& \qquad \text{s.t.} \quad \mathbf{y}' \text{ is a valid network flow,} \\
& \qquad\qquad x_{ij} = 0 \implies y'_{ij} = 0,
\end{aligned}
$$

where $A_2 \cap A_1 = \emptyset, A = A_1 \cup A_2$. The leader minimizes the total travel time across all links subject to a budget constraint and the followers' equilibrium which is expressed as a network flow on the graph augmented by the leader's selected edges that satisfies O-D demands; the integral in the follower's objective models the desired equilibrium and evaluates to $T_{ij} y'_{ij} + \frac{0.15 T_{ij}}{5 c_{ij}^4} \left(y'_{ij}\right)^5$.

Going beyond the DNDP, Dempe [59] lists more than 70 applications of BiLO ranging from pricing in electricity markets (leader is an electricity-supplying retailer that sets the price to maximize profit, followers are consumers who react accordingly to satisfy their demands [235]) to interdiction problems in security settings (leader inspects a budgeted subset of nodes on a road network, follower selects a path such that they evade inspection [211]).

Scope of this work. We are interested in *mixed-integer non-linear bilevel optimization* problems, simply referred to hereafter as *bilevel optimization* or BiLO, a very general class of bilevel problems where all constraints and objectives may involve non-linear terms and integer variables. At a high level, we have identified three limitations of existing computational methods for BiLO:

1. The state-of-the-art exact solvers of Fischetti et al. [76] and Tahernejad et al. [193] are limited to mixed-integer bilevel *linear* problems and do not scale well. When high-quality solutions to large-scale problems are sought after, such exact solvers may be prohibitively slow.
2. Specialized algorithms, heuristic or exact, do not generalize beyond the single problem they were designed for. For instance, the state-of-the-art exact Knapsack Interdiction solver [214] only works for a single knapsack constraint and fails with two or more, a significant limitation even if one is strictly interested in knapsack-type problems.
3. Existing methods, exact or heuristic, generic or specialized, are not designed for the "data-driven algorithm design" setting [12] in which similar instances are routinely solved and the goal is to construct generalizable high-efficiency algorithms that leverage historical data.

NEUR2BILO (for *Neural Bilevel Optimization*) is a learning-based framework for bilevel optimization that deals with these issues simultaneously. The following observations make NEUR2BILO possible:

1. **Data collection is "easy":** For a fixed decision of the leader's, the optimal value of the follower can be computed by an appropriate (single-level) solver (e.g., for mixed-integer programming (MIP) or convex programming), enabling the collection of samples of the form: (leader's decision, follower's value, leader's value).
2. **Offline learning in the data-driven setting:** While obtaining data online may be prohibitive, access to historical training instances affords us the ability to construct, offline, a large dataset of samples that can then serve as the basis for learning an approximate value function using supervised regression. The output of this training is a regressor mapping a pair consisting of an instance and a leader's decision to an estimated follower or leader value.
3. **MIP embeddings of neural networks:** If the regressor is MIP-representable, e.g., a feedforward ReLU neural network or a decision tree, it is possible to use an MIP solver to find the leader's decision that minimizes the regressor's output. This MIP, which includes any leader constraints, thus serves as an approximate single-level surrogate of the original bilevel problem instance.
4. **Follower constraints via the value function reformulation:** The final ingredient of the NEUR2BILO recipe is to include any of the follower's constraints, some of which may involve leader variables. This makes the surrogate problem a heuristic version of the well-known *value function reformulation* (VFR) in BiLO. The VFR transforms a bilevel problem into a single-level one, assuming that one can represent the follower's value (as a function of the leader's decision) compactly. This is typically im-

possible as the value function may require an exponential number of constraints, a bottleneck that is circumvented by our small (approximate) regression models.

5. **Theoretical guarantees:** For interdiction problems, a class of BiLO problems that attracts much attention, NEUR2BILO solutions have a constant, additive absolute optimality gap which mainly depends on the prediction accuracy of the regression model.

Through a series of experiments on (i) the bilevel knapsack interdiction problem, (ii) the "critical node problem" from network security, (iii) a donor-recipient healthcare problem, and (iv) the DNDP, we will show that NEUR2BILO is easy to train and produces, very quickly, heuristic solutions that are competitive with state-of-the-art methods.

3.2. BACKGROUND

Bilevel optimization (BiLO) deals with hierarchical problems where the *leader* (or *upper-level*) problem decides on $\mathbf{x} \in \mathcal{X}$ and parameterizes the *follower* (or *lower-level*) problem that decides on $\mathbf{y} \in \mathcal{Y}$; the sets $\mathcal{X}$ and $\mathcal{Y}$ represent the domains of the variables (continuous, mixed-integer, or pure integer). Both problems have their own objectives and constraints, resulting in the following model:

$$\min_{\mathbf{x}\in\mathcal{X},\mathbf{y}} \quad F(\mathbf{x},\mathbf{y}) \tag{3.1a}$$

$$\text{s.t.} \quad G(\mathbf{x},\mathbf{y}) \geq \mathbf{0}, \tag{3.1b}$$

$$\mathbf{y} \in \arg\max_{\mathbf{y}'\in\mathcal{Y}} \{f(\mathbf{x},\mathbf{y}') : g(\mathbf{x},\mathbf{y}') \geq \mathbf{0}\}, \tag{3.1c}$$

where we consider the general mixed-integer non-linear case with $F, f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, $G : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{m_1}$, and $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{m_2}$ non-linear functions of the upper-level $\mathbf{x}$ and lower-level variables $\mathbf{y}$.

The applicability of exact (i.e., global) approaches critically depends on the nature of the lower-level problem. A continuous lower-level problem admits a single-level reformulation that leverages the Karush-Kuhn-Tucker (KKT) conditions as constraints on $\mathbf{y}$. For linear programs in the lower level, strong duality conditions can be used in the same way. Solving a BiLO problem with integers in the lower level necessitates more sophisticated methods such as branch and cut [61, 76] along with some assumptions: DeNegre and Ralphs [61] do not allow for coupling constraints (i.e., $G(\mathbf{x},\mathbf{y}) = G(\mathbf{x})$) and both methods do not allow continuous upper-level variables to appear in the linking constraints ($g(\mathbf{x},\mathbf{y})$). Other approaches, such as Benders decomposition, are also applicable [79]. Gümüş and Floudas [91] propose single-level reformulations of mixed-integer non-linear BiLO problems using polyhedral theory, an approach that only works for small problems. Later, "branch-and-sandwich" methods were proposed [119, 165] where bounds on both levels' value functions are used to compute an optimal solution. Algorithms for non-linear BiLO generally do not scale well. Kleinert et al. [118] survey more exact methods.

Assumptions. In what follows, we make the following standard assumptions:

1. Either (i) the follower's problem has a feasible solution for each $\mathbf{x} \in \mathscr{X}$, or (ii) there are no coupling constraints in the leader's problem, i.e., $G(\mathbf{x},\mathbf{y}) = G(\mathbf{x})$;
2. The optimal follower value is always attained by a feasible solution [see 19, Section 7.2].

Value function reformulation. We consider the so-called *optimistic* setting: if the follower has multiple optima for a given decision of the leader's, the one that optimizes the leader's objective is implemented. We can then rewrite problem (3.1) using the *value function reformulation* (VFR):

$$\min_{\mathbf{x}\in\mathscr{X},\mathbf{y}\in\mathscr{Y}} \quad F(\mathbf{x},\mathbf{y}) \tag{3.2a}$$

$$\text{s.t.} \quad G(\mathbf{x},\mathbf{y}) \geq \mathbf{0}, \tag{3.2b}$$

$$g(\mathbf{x},\mathbf{y}) \geq \mathbf{0}, \tag{3.2c}$$

$$f(\mathbf{x},\mathbf{y}) \geq \Phi(\mathbf{x}), \tag{3.2d}$$

with the *optimal lower-level value function* defined as

$$\Phi(\mathbf{x}) = \max_{\mathbf{y}\in\mathscr{Y}} \{f(\mathbf{x},\mathbf{y}) : g(\mathbf{x},\mathbf{y}) \geq \mathbf{0}\}. \tag{3.3}$$

Lozano and Smith [142] used this formulation to construct an exact algorithm (without any public code) for solving mixed-integer non-linear BiLO problems with purely integer upper-level variables. Sinha et al. [181, 182, 184] propose a family of evolutionary heuristics for continuous non-linear BiLO problems that approximate the optimal value function by using quadratic and Kriging (i.e., a function interpolation method) approximations. Taking it one step further, Beykal et al. [35] extend the framework of the previous authors to handle mixed-integer variables in the lower level.

3.3. RELATED WORK

Learning for bilevel optimization. Besides the approaches of Sinha et al. [181, 182, 184] and Beykal et al. [35] discussed in Section 3.2, other learning-based methods have been introduced to solve BiLO problems. Bagloee et al. [11] present a heuristic for DNDP which uses a linear prediction of the leader's objective function. An iterative algorithm refines the prediction with new solutions, terminating after a pre-determined number of iterations. Chan et al. [52] propose to simultaneously optimize the parameters of a learning model for a subset of followers in a large-scale cycling network design problem. Here, only non-parametric or linear models are utilized as optimizing more sophisticated learning models is generally challenging with MILP-based optimization. Molan and Schmidt [150] make use of a neural network to predict the follower variables. The authors assume a setting with a black-box follower's problem, no coupling constraints, and continuous leader variables. Another learning-based heuristic is proposed by Kwon et al. [129] for a bilevel knapsack problem. This approach is knapsack-specific and requires a sophisticated, GPU-based, problem-specific graph neural network for which no code is

publicly available. Zhou et al. [233] propose a learning-based algorithm for binary bilevel problems which, similar to our approach, predicts the optimal value function and develops a single-level reformulation based on the trained model. They propose using a graph neural network and an input-supermodular neural network, both of which can only be trained on a single instance rather than learning across classes of instances as NEUR2BILO does. NEUR2BILO significantly outperforms this method as shown in Appendix 3.E. For continuous unconstrained bilevel optimization, a substantially different setting, many methods have been proposed recently due to interest in solving nested problems in machine learning (e.g., hyperparameter tuning and meta-learning) [139].

Data-driven optimization. The integration of a trained machine learning model into an MIP is a vital element of NEUR2BILO. This is possible due to MILP formulations of neural networks [54, 75, 177], and of other predictors like decision trees [29, 140]. These methods have become easily applicable due to open software implementations [24, 51, 145, 199] and the `gurobi-machinelearning` library [1]. One such application is constraint learning [71]. More similar to our setting are the approaches in [66, 67, 124] for predicting value functions of other nested problems such as two-stage stochastic and robust optimization. Our method caters to the specificities of BiLO, particularly in the lower-level approximation which performs well in highly-constrained BiLO settings such as the DNDP, has approximation guarantees based on the error of the predictive model, and computational results on problems with non-linear interactions between the variables in each stage of the optimization problem; these aspects distinguish NEUR2BILO from prior work.

3.4. METHODOLOGY

NEUR2BILO refers to two learning-based single-level reformulations for general BiLO problems. The reformulations rely on representing the thorny nested structure of a BiLO problem with a trained regression model that predicts either the upper-level or lower-level value functions. Section 3.A includes pseudocode for data collection, training, and model deployment.

3.4.1. NEUR2BILO

Upper-level approximation. The obvious bottleneck in solving BiLO problems is their nested structure. One rather straightforward way of circumventing this difficulty is to get rid of the lower level altogether in the formulation, but predict its optimal value. Namely, we predict the optimal upper-level objective value function as

$$\mathrm{NN}^u(\mathbf{x};\Theta) \approx F(\mathbf{x},\mathbf{y}^\star), \tag{3.4}$$

where Θ are the weights of a neural network, F the objective function of the leader (3.2b), and $\mathbf{y}^\star$ an optimal solution to the lower level problem (3.3). To train such a model, one can sample $\mathbf{x}$ from $\mathscr{X}$, solve (3.3) to obtain an optimal lower-level solution $\mathbf{y}^\star$, and sub-

[1] https://github.com/Gurobi/gurobi-machinelearning

sequently compute a label $F(\mathbf{x}, \mathbf{y}^\star)$. We can then model the single-level problem as

$$\min_{\mathbf{x}\in\mathcal{X}} \quad \mathrm{NN}^u(\mathbf{x};\Theta) \quad \text{s.t. } G(\mathbf{x}) \geq \mathbf{0}, \tag{3.5}$$

where we only optimize for $\mathbf{x}$ and thus dismiss the lower-level constraints and objective function. A trained feedforward neural network $\mathrm{NN}^u(\cdot;\Theta)$ with ReLU activations can be represented as a mixed-integer linear program (MILP) [75], where now the input (and output) of the network are decision variables. With this representation, Problem (3.5) becomes a single-level problem and can be solved using an off-the-shelf MIP solver. Note that linear and decision tree-based models also admit MILP representations [140].

This reformulation is similar to the approach by Bagloee et al. [11], wherein the upper-level value function is predicted using linear regression. Our method differs in that it is not iterative and does not require the use of "no-good cuts" (which avoid reappearing solutions $\mathbf{x}$). As such, our method is extremely efficient as will be shown experimentally.

The formulation of (3.5) only allows for problem classes that do not have coupling constraints, i.e., $G(\mathbf{x}, \mathbf{y}) = G(\mathbf{x})$. Moreover, the feasibility of a solution $\mathbf{x}$ in the original BiLO problem is not guaranteed, an issue that will be addressed later in this section (see **Bilevel feasibility**).

Lower-level approximation. This method makes use of the VFR (3.2). The VFR moves the nested complexity of a BiLO to constraint (3.2d), where the right-hand side is the optimal value of the lower-level problem, parameterized by $\mathbf{x}$. We introduce a learning-based VFR in which $\Phi(\mathbf{x})$ is approximated by a regression model with parameters Θ:

$$\mathrm{NN}^l(\mathbf{x};\Theta) \approx \Phi(\mathbf{x}). \tag{3.6}$$

Both NN^l and NN^u take in a leader's decision as input and require solving the follower (3.3) for data generation. By replacing $\Phi(\mathbf{x})$ with $\mathrm{NN}^l(\mathbf{x};\Theta)$ in (3.2d) and introducing a slack variable $s \in \mathbb{R}_+$, the surrogate VFR reads as:

$$\min_{\substack{\mathbf{x}\in\mathcal{X},\mathbf{y}\in\mathcal{Y}\\ s\geq 0}} \quad F(\mathbf{x},\mathbf{y}) + \lambda s \tag{3.7a}$$

$$\text{s.t.} \quad G(\mathbf{x},\mathbf{y}) \geq \mathbf{0}, \tag{3.7b}$$

$$g(\mathbf{x},\mathbf{y}) \geq \mathbf{0}, \tag{3.7c}$$

$$f(\mathbf{x},\mathbf{y}) \geq \mathrm{NN}^l(\mathbf{x};\Theta) - s. \tag{3.7d}$$

All follower and leader constraints of the original BiLO problem are part of Problem (3.7). However, without the slack variable s, the problem could become infeasible due to inaccurate predictions by the neural network. This happens when $\mathrm{NN}^l(\mathbf{x};\Theta)$ strictly overestimates the follower's optimal value for each $\mathbf{x}$. In this case, there does not exist a follower decision for which Constraint (3.7d) is satisfied. A value of $s > 0$ can be used to make Constraint (3.7d) satisfiable at a cost of λs in the objective, guaranteeing feasibility.

Bilevel feasibility. Given a solution $\mathbf{x}^\star$ or a solution pair $(\mathbf{x}^\star, \tilde{\mathbf{y}})$ returned by our upper- or lower-level approximations, respectively, we would like to produce a lower-level solution $\mathbf{y}^\star$ such that $(\mathbf{x}^\star, \mathbf{y}^\star)$ is bilevel-feasible, i.e., it satisfies the original BiLO in (3.1). The following procedure achieves this goal:

1. Compute the follower's optimal value under $\mathbf{x}^\star$, $\Phi(\mathbf{x}^\star)$, by solving (3.3).
2. Compute a bilevel-feasible follower solution $\mathbf{y}^\star$ by solving problem (3.2) with fixed $\mathbf{x}^\star$ and the right-hand side of (3.2d) set to $\Phi(\mathbf{x}^\star)$, a constant. Return $(\mathbf{x}^\star, \mathbf{y}^\star)$.

If only Assumption 1(i) is satisfied, then only the lower-level approximation is applicable and this procedure guarantees an optimistic bilevel-feasible solution for it. If only Assumption 1(ii) is satisfied, then this procedure can detect in Step 1 that an upper-level approximation's solution $\mathbf{x}^\star$ does not admit a follower solution, i.e., that it is infeasible, or calculates a feasible $\mathbf{y}^\star$ if one exists in Step 2. If both Assumptions 1(i) and 1(ii) are satisfied simultaneously, then this procedure guarantees an optimistic bilevel-feasible solution for either approximation.

Upper- v.s. lower-level approximation. Here, we note two important trade-offs between the upper- and lower-level approximations.

– **Generality**: Example 3.1 in Appendix 3.B shows that under Assumption 1(ii), it may happen that solving the upper-level approximation problem variant (3.5) returns an infeasible solution while the lower-level variant (3.7) does not.

– **Scalability**: The upper-level approximation has fewer variables and constraints than its lower-level counterpart as it does not represent the follower's problem directly. For problems in which the lower-level problem is large, e.g., necessitating constraints for each node and link to enforce a network flow in the follower solution as in the DNDP from the introduction, this property makes the upper-level approximation easier to solve, possibly at a sacrifice in final solution quality. This tradeoff will be assessed experimentally.

Limitations. Since NEUR2BILO is in essence a learning-based *heuristic*, it does not guarantee an optimal solution to the bilevel problem. However, it guarantees a feasible solution with the lower-level approximation and can only give an infeasible solution while using the upper-level approximation when only Assumption 1(ii) is satisfied. Moreover, as will be shown in Section 3.4.3, the performance of NEUR2BILO depends on the regression error, which is generally the case when integrating machine learning in optimization algorithms. Empirically, we note that the prediction error achieved on every problem is very low (see Appendix 3.J.3).

3.4.2. MODEL ARCHITECTURE

For ease of notation in previous sections, all regression models take as input the upper-level decision variables. However, in our experiments, we leverage instance information as well to train *a single model* that can be deployed on a family of instances. This is done by leveraging information such as coefficients in the objective and constraints for each

problem. For the model's architecture, the general principle deployed is to first explicitly represent or learn instance-based features. The second is to combine instance-based features with (leader) decision variable information to make predictions.

The overall architecture can be summarized as the following set of operations. Fix a particular instance of a BiLO problem and let n be the number of leader variables, $\mathbf{f}_i$ a vector of features for each leader variable x_i (independently of the variable's value), and $h(x_i)$ a feature map that describes the ith leader variable for a specific value of that variable. The functions Ψ^s, Ψ^d, and Ψ^v are neural networks with appropriate input-output dimensions. The vector Θ includes all learnable parameters of networks Ψ^s, Ψ^d, and Ψ^v. The functions SUM, CONCAT, and AGGREGATE sum up a set of vectors, concatenate two vectors into a single column vector, and aggregate a set of scalar values (e.g., by another neural network or simply summing them up), respectively. Our final objective value predictions are then given by the following sequence of steps:

1. Embedding the set of variable features $\{\mathbf{f}_i\}$ using a set-based architecture, e.g., the same network Ψ^d, summing up the resulting n variable embeddings, then passing the resulting vector to network Ψ^s, yielding a vector we refer to as the INSTANCEEMBEDDING:

$$\text{INSTANCEEMBEDDING} = \Psi^s(\text{SUM}(\{\Psi^d(\mathbf{f}_i)\}_{i=1}^n)).$$

   This is akin to the DeepSets approach of Zaheer et al. [226]. However, note that this step can alternatively be done via a feedforward or graph neural network depending on the problem structure.

2. Conditional on a specific assignment of values to the leader's decision vector $\mathbf{x}$, a per-variable embedding is computed by network Ψ^v to allow for interactions between the INSTANCEEMBEDDING and the specific assignment of variable i as represented by $h(x_i)$:

$$\text{VARIABLEEMBEDDING}(i) = \Psi^v(\text{CONCAT}(h(x_i), \text{INSTANCEEMBEDDING})).$$

3. The final value prediction for either of our approximations aggregates the variable embeddings possibly after passing them through a function g_i:

$$\text{NN}(\mathbf{x};\Theta) = \text{AGGREGATE}(\{g_i(\text{VARIABLEEMBEDDING}(i))\}_{i=1}^n).$$

   For example, if the follower's objective is a linear function and VARIABLEEMBEDDING(i) is a scalar, then it is useful to use the variable's known objective function coefficient d_i here, i.e.: $g_i(\text{VARIABLEEMBEDDING}(i)) = d_i \cdot \text{VARIABLEEMBEDDING}(i)$. The final step is to aggregate the per-variable $g_i(\cdot)$ outputs, e.g., by a summation for linear or separable objective functions.

NEUR2BILO is largely agnostic to the learning model utilized as long as it is MILP-representable. In our experiments, we primarily focus on neural networks, but for some problems also explore the use of gradient-boosted trees. More details on the specific architectures for each problem can be found in Appendix 3.J.1.

3.4.3. APPROXIMATION GUARANTEES

Lower-level approximation. Next, we present an approximation guarantee for the lower-level approximation with $\mathrm{NN}^l(\mathbf{x};\Theta)$. Appendix 3.C includes the complete proofs.

Since the prediction of the neural network is only an approximation of the true optimal value of the follower's problem $\Phi(\mathbf{x})$, NEUR2BILO may return sub-optimal solutions for the original problem (3.1). We derive approximation guarantees for a specific setup that appears in interdiction problems: the leader and the follower have the same objective function (i.e., $f(\mathbf{x},\mathbf{y}) = F(\mathbf{x},\mathbf{y})$ for all $\mathbf{x} \in \mathscr{X}, \mathbf{y} \in \mathscr{Y}$), and Assumption 1(i) holds. Consider a neural network that approximates the optimal value of the follower's problem up to an absolute error of $\alpha > 0$, i.e.,

$$|\mathrm{NN}^l(\mathbf{x};\Theta) - \Phi(\mathbf{x})| \leq \alpha \quad \text{for all } \mathbf{x} \in \mathscr{X}. \tag{3.8}$$

Furthermore, we define the parameter Δ as the maximum difference $f(\mathbf{x},\mathbf{y}) - f(\mathbf{x},\mathbf{y}') \geq 0$ over all $\mathbf{x} \in \mathscr{X}, \mathbf{y}, \mathbf{y}' \in \mathscr{Y}$ such that no $\tilde{\mathbf{y}} \in \mathscr{Y}$ exists which has function value $f(\mathbf{x},\mathbf{y}) > f(\mathbf{x},\tilde{\mathbf{y}}) > f(\mathbf{x},\mathbf{y}')$. We can bound the approximation guarantee of the lower-level NEUR2BILO as follows:

Theorem 3.1. *If the leader and the follower have the same objective function and* $\lambda > 1$, NEUR2BILO *returns a feasible solution* $(\mathbf{x}^\star,\mathbf{y}^\star)$ *for Problem* (3.1) *with objective value*

$$f(\mathbf{x}^\star,\mathbf{y}^\star) \leq \mathrm{opt} + 3\alpha + \frac{2}{\lambda}\Delta,$$

where opt *is the optimal value of* (3.1) *and* λ *the penalty term in* (3.7a) .

Upper-level approximation. As Example 3.1 shows, it may happen that the upper-level surrogate problem (3.5) returns an infeasible solution and hence no approximation guarantee can be derived in this case. However, in the case where all leader solutions are feasible and the neural network predicts for every $\mathbf{x} \in \mathscr{X}$ an upper-level objective value that deviates at most $\alpha > 0$ from the true one, then the returned solution trivially approximates the true optimal value with an absolute error of at most 2α. This follows since the worst that can happen is that the objective value of the optimal solution is overestimated by α while a solution with objective value $\mathrm{opt}+2\alpha$ is underestimated by α and hence has the same predicted value as the optimal solution. Problem (3.5) then may return the latter sub-optimal solution.

3.5. EXPERIMENTAL SETUP

Benchmark problems and their characteristics are summarized in Table 3.1; their MIP formulations are deferred to Appendix 3.D and brief descriptions follow:

- **Knapsack interdiction (KIP) [48]:** The leader interdicts a subset of at most k items and the follower solves a knapsack problem over the remaining (non-interdicted) items. The leader aims to minimize the follower's (maximization) objective.
- **Critical node problem (CNP) [50, 63]:** This problem regards the protection (by the leader) of resources in a network against malicious follower attacks. It has applications in the protection of computer networks against cyberattacks as demonstrated by Dragotto et al. [63].

Problem	Leader			Follower		
	x	Obj.	Cons.	**y**	Obj.	Cons.
KIP (↓↑)	Binary	Linear	Linear	Binary	Linear	Linear
CNP (↑↑)	Binary	Bilinear	Linear	Binary	Bilinear	Linear
DRP (↑↑)	Continuous	Linear	Linear	Mixed-Integer	Linear	Bilinear
DNDP (↓↑)	Binary	Non-Linear	Linear	Continuous	Non-Linear	Linear

Table 3.1: Problem class characteristics. All problems have a single budget constraint in the leader; for the follower, the DNDP has network flow constraints whereas other problems have a knapsack constraint. The arrows refer to minimization (↓) or maximization (↑) in leader and follower, respectively.

– **Donor-recipient problem (DRP) [84]:** This problem relates to the donations given by certain agencies to countries in need of, e.g., healthcare projects. The leader (the donor agency) decides on which proportion of the cost, per project, to subsidize, whereas the follower (a country) decides which projects it implements.
– **Discrete network design problem (DNDP) [171]:** This is the problem described in Section 3.1. We build on the work of Rey [171] who provided benchmark instances for the transportation network of Sioux Falls, South Dakota, and an implementation of the state-of-the-art method of Fontaine and Minner [79]. This network and corresponding instances are representative of the state of the DNDP in the literature.

Baselines. As mentioned previously, the branch-and-cut (B&C) algorithm by Fischetti et al. [76] is considered to be state-of-the-art for solving mixed-integer linear BiLO. The method is applicable if the continuous variables of the leader do not appear in the follower's constraints. Both KIP and CNP meet these assumptions. This algorithm will act as the baseline for these problems. For DRP, we compare against the results produced by an algorithm in the branch-and-cut paradigm (B&C+) from Ghatkar et al. [84]. For DNDP, the follower's problem only has continuous variables, so the baseline is a method based on KKT conditions (MKKT) [79]. Of the learning-based approaches for BiLO, we compare against Zhou et al. [233], given the generality of their approach and the availability of source code. NEUR2BILO decisively outperforms this method on KIP, finding solutions with 10-100× smaller mean relative error roughly 1000× faster; full results are deferred to Appendix 3.E.

Data collection & training. For each problem class, data is collected by sampling feasible leader decisions $\mathbf{x}$ and then solving $\Phi(\mathbf{x})$ to compute either the upper- or lower-level objectives as labels. We then train regression models to minimize the least-squares error on the training samples. Typically, data collection and training take less than one hour, a negligible cost given that for larger instances baseline methods require more time *per instance*. Additionally, the same trained model can be used on multiple unseen test instances. We report times for data collection and training in Appendix 3.J.2.

Evaluation & setup. For evaluation in KIP, CNP, and DRP, all solving was limited to 1 hour. For DNDP, we consider a more limited-time regime, wherein we compare NEUR2BILO at 5 seconds against the baseline at 5, 10, and 30 seconds. For all problems, we evaluate both the lower- and upper-level approximations with neural networks, namely NN^l and NN^u, respectively. For NN^l we set $\lambda = 1$ for all results presented in this chapter. Details of the computing setup are provided in Appendix 3.I. Our code and data are available at `https://github.com/khalil-research/Neur2BiLO`.

3.6. EXPERIMENTAL RESULTS

We summarize the results as measured by average solution times and mean relative errors (MREs). The relative error on a given instance is computed as $100 \cdot \frac{|obj_A - obj_{best}|}{|obj_{best}|}$, where obj_A is the value of the solution found by method A and obj_{best} is the best-known objective value for that instance. These results are reported in Table 3.2. More detailed results and box-plots of the distributions of relative errors are in Appendices 3.F and 3.G. Our experimental design answers the following questions:

Q1: Can NEUR2BILO find high-quality solutions quickly on classical interdiction problems? Table 3.2 compares NEUR2BILO to the B&C algorithm of Fischetti et al. [76]. NEUR2BILO terminates in 1-2% of the time required by B&C on the smaller ($n \leq 30$) well-studied KIP instances of Tang et al. [195]. However, when the instance size increases to $n = 100$, both NN^l and NN^u find much better solutions than NEUR2BILO in roughly 30 seconds, even when B&C runs for the full hour. Furthermore, Table 3.4 in Appendix 3.F shows that B&C requires 10 to 1,000× more time than NN^l or NN^u to find equally good solutions. In addition, the best solutions found by B&C at the termination times of NN^l or NN^u are generally worse, even for small instances.

Q2: Do these computational results extend to non-linear and more challenging BiLO problems? Interdiction problems such as the KIP are well-studied but are only a small subset of BiLO. We will shift attention to the more practical problems, starting with the CNP (Table 3.2). CNP includes terms that are bilinear (i.e., $z = xy$) in the upper- and lower-level variables, resulting in a much more challenging problem for general-purpose B&C. In this case, both NN^l and NN^u tend to outperform B&C as the problem size increases. In addition, the results on incumbents reported in Table 3.5 in Appendix 3.F are as good, if not even stronger than those of KIP.

Secondly, we discuss DRP (Table 3.6 in Appendix 3.F). For DRP, we evaluate on the most challenging instances from Ghahtarani et al. [83], all of which have gaps of ~ 50% at a 1-hour time limit with B&C+, a specialized B&C-based algorithm. Here NN^u performs remarkably well: it finds the best-known solutions on every single instance in roughly ~ 0.1 seconds at an average improvement in solution quality of 26% over B&C+.

Q3: How does NEUR2BILO perform on BiLO problems with complex constraints? Given that NEUR2BILO has strong performance on benchmarks with budget constraints, the next obvious question is whether it can be applied to BiLO problems that have complex constraints. To answer this, we will refer to the results in Table 3.2 for

Table 3.2: Mean relative error (MRE) and solving times for KIP, CNP, and DNDP. For KIP with $n \leq 30$, we directly evaluate on the 180 instances (10 per size) of Tang et al. [195]; each value is the average over 10 instances. For $n = 100$, our evaluation instances (100 per size) are generated using the same procedure of Tang et al. [195]. The no-learning baseline G-VFA is a VFR using the follower's greedy solution as lower-level value function approximation. For CNP, each row is averaged over 300 instances that are randomly sampled using the procedure described in Dragotto et al. [63]. For DNDP, each row is averaged over 10 instances from Rey [171]. The budget is a fraction of the total cost of all 30 possible candidate links; see Appendix 3.J.2 for more details.

Knapsack Interdiction Problem									
# Items	Interdiction	NN^l		NN^u		G-VFA		B&C	
(n)	Budget (k)	MRE	Time	MRE	Time	MRE	Time	MRE	Time
18	5	1.48	0.59	1.48	0.34	1.82	**0.14**	**0.00**	9.55
18	9	1.51	0.59	1.51	0.43	3.97	**0.22**	**0.00**	5.81
18	14	**0.00**	0.22	**0.00**	0.17	64.22	**0.03**	**0.00**	0.39
20	5	0.41	0.62	0.41	0.45	2.19	**0.25**	**0.00**	23.18
20	10	0.99	0.66	0.99	0.58	0.99	**0.36**	**0.00**	10.27
20	15	3.57	0.32	3.57	0.19	23.39	**0.02**	**0.00**	0.94
22	6	0.71	0.19	0.71	**0.18**	0.42	0.18	**0.00**	42.30
22	11	1.01	0.28	1.01	**0.28**	1.08	0.33	**0.00**	16.26
22	17	14.43	0.24	14.43	0.15	14.43	**0.13**	**0.00**	0.68
25	7	0.44	2.66	0.44	2.42	0.44	**0.64**	**0.00**	137.96
25	13	1.42	2.75	1.42	2.79	3.85	**1.24**	**0.00**	48.43
25	19	2.49	0.48	2.49	0.38	2.49	**0.13**	**0.00**	1.77
28	7	0.39	0.67	0.39	0.74	0.26	**0.62**	**0.00**	309.18
28	14	0.75	2.10	0.75	1.45	1.37	**1.29**	**0.00**	120.74
28	21	1.14	0.45	1.14	0.49	3.16	**0.31**	**0.00**	4.92
30	8	**0.00**	1.54	**0.00**	1.54	0.43	**0.97**	**0.00**	792.44
30	15	0.49	3.64	0.49	3.06	0.75	**1.35**	**0.00**	187.23
30	23	2.29	1.08	2.29	0.73	4.48	**0.25**	**0.00**	5.65
100	25	0.93	10.02	0.93	8.40	**0.00**	**4.19**	8.09	3,600.40
100	50	0.96	51.68	0.96	**49.28**	**0.04**	53.74	8.96	3,600.44
100	75	**0.08**	24.69	**0.08**	**23.78**	0.12	35.27	5.87	3,600.52
Avg. $n \leq 30$		1.86	1.06	1.86	0.91	7.21	**0.47**	**0.00**	95.43
Avg. $n = 100$		0.66	28.80	0.66	**27.15**	**0.05**	31.07	7.64	3,600.45

Critical Node Problem						
# Nodes	NN^l		NN^u		B&C	
($\|V\|$)	MRE	Time	MRE	Time	MRE	Time
10	3.20	0.04	2.75	**0.02**	**1.01**	4.24
25	2.60	0.23	1.77	**0.05**	**0.73**	3,244.20
50	1.42	0.38	0.98	**0.10**	**0.67**	3,600.30
100	1.12	0.48	**0.56**	**0.42**	1.79	3,600.65
300	2.01	1.12	**0.33**	**0.83**	2.32	3,600.54
500	1.33	1.69	**0.45**	**1.19**	2.47	3,600.80
Average	1.95	0.66	**1.14**	**0.43**	1.50	2,941.79

Discrete Network Design Problem								
		NN^l		NN^u		MKKT		
# Edges	Budget	MRE	Time	MRE	Time	MRE-5	MRE-10	MRE-30
10	0.25	1.21	2.95	0.36	**0.01**	5.78	0.51	**0.10**
10	0.5	0.73	3.35	1.22	**0.01**	6.47	2.17	**0.00**
10	0.75	0.47	2.80	1.32	**0.00**	5.80	**0.02**	0.06
20	0.25	6.05	5.02	2.64	**0.02**	7.78	5.12	**0.85**
20	0.5	1.01	4.91	4.36	**0.03**	6.00	2.52	**0.64**
20	0.75	0.85	4.47	0.91	**0.01**	7.87	0.22	**0.11**
Average		1.72	3.92	1.80	**0.01**	6.62	1.76	**0.29**

the DNDP. In this setting, we focus on a limited-time regime wherein we compare NEUR2BILO with a 5-second time limit to MKKT at time limits 5, 10, and 30 seconds. NN^u can achieve high-quality solutions much faster than any other method with only a minor sacrifice in solution quality, making it a great candidate for domains where interactive decision-making is needed (e.g., what-if analysis of various candidate roads, budgets, etc.).

NN^l, on the other hand, takes longer than NN^u but computes solutions that are more competitive with the baseline, the latter requiring 5× more time. We suspect that the better solution quality from NN^l is due to its explicit modeling of feasible lower-level decisions that "align" with the predictions, whereas NN^u may simply extrapolate poorly. In terms of computing time, one computational burden for NN^l is the requirement to model the non-linear upper- and lower-level objectives, which requires a piece-wise linear approximation based on Fontaine and Minner [79], a step that introduces additional variables and constraints. Appendix 3.F includes results for DNDP with gradient-boosted trees (GBT), demonstrating that other learning models are directly applicable and, in some cases, may even lead to better solution quality, faster optimization, and simpler implementation.

Q4: Can approximations derived from heuristics be useful? We now refer back to KIP and focus on the greedy value function approximation (G-VFA), a KIP-specific approximation that relies on the fact that greedy algorithms are typically good for 1-dimensional knapsack problems. Namely, the heuristic is based on ordering the items with their value-to-weight ratio [57] and is used as the knapsack solution in the follower problem, while still being parameterized by $\mathbf{x}$. This heuristic is embedded in a single-level problem as this heuristic is MILP-representable [see 8]; we note that we are not aware of uses in the literature of this approximation and it may be of independent interest. Generally, G-VFA performs quite well, and in some cases outperforms NN^l and NN^u, but there are clear cases where NN^l and NN^u outperform G-VFA demonstrating that learning is beneficial. In addition, heuristics like G-VFA can be utilized to compute features for NN^l and NN^u. For KIP, the inclusion of these features derived from G-VFA strongly improves the results (see Table 3.10 in Appendix 3.H.3). This demonstrates that there is value in leveraging any problem-specific MILP-representable heuristics as features for learning.

Q5: How does λ affect NN^l? Table 3.9 in Appendix 3.H.2 shows that a slack penalty of $\lambda = 0.1$ improves the performance of NN^l on some instances for DNDP, compared to the $\lambda = 1$ reported in Table 3.2, indicating that tuning over λ might be beneficial. As an alternative to adding slack, one can even dampen predictions of the value function to allow more flexibility using the empirical error observed during training; see Table 3.8 in Appendix 3.H.1.

3.7. CONCLUSION

In both its upper- and lower-level instantiations, NEUR2BILO finds high-quality solutions in a few milliseconds or seconds across four benchmarks that span applications in interdiction, network security, healthcare, and transportation planning. In fact, we are not aware of any bilevel optimization method which has been evaluated across such a diverse range of problems as existing methods make stricter assumptions that limit their applicability. NEUR2BILO models are generic, easy to train, and accommodating of problem-specific heuristics as features. One limitation of our experiments is that they lack a problem that involves coupling constraints in (3.1b). We could not identify benchmark problems with this property in the literature, but exploring this setting would be valuable. Of future interest are potential extensions to bilevel *stochastic* optimization [18], robust optimization with decision-dependent uncertainty [87] (a special case of BiLO), and multi-level problems beyond two levels, e.g. [131].

APPENDIX OF CHAPTER 3

3.A. NEUR2BILO PSEUDOCODE

Here, we outline pseudocode for NEUR2BILO. Algorithm 3.1 presents the pseudocode for data collection and training. Algorithm 3.2 presents the pseudocode for optimization. Following Algorithm 3.2, the objective is computed via the bilevel feasibility procedure detailed in Section 3.4.1. Note that data collection can be done once to collect labels for both the upper- and lower-level approximations. Additionally, a single trained model may be (and is in our experiments) evaluated across multiple test instances.

Algorithm 3.1: NEUR2BILO Data Collection and Training

```
   // Data Collection
1  𝒟 ← {}
2  for i = 1 to number of instances to sample do
3  |  𝒫 ← sampled instance. Note that 𝒫 is defined by F(·), G(·), f(·), g(·), 𝒴, and
   |    𝒳. For most BiLO problems, these functions are defined by the constraint
   |    and objective coefficients
4  |  for j = 1 to number of decisions per-instance do
5  |  |  x ← sampled upper-level decision
6  |  |  y* ← argmax_{y∈𝒴} {f(x, y) : g(x, y) ≥ 0}
7  |  |  Add (𝒫, x, F(x, y*), f(x, y*)) to 𝒟
8  |  end
9  end
10 return 𝒟
   // Training
11 if approximating upper-level then
12 |  Train regressor (NN^u) with features (𝒫, x) and label (F(·)) from dataset 𝒟
13 else if approximating lower-level then
14 |  Train regressor (NN^l) with the features (𝒫, x) and label (f(·)) from dataset 𝒟
15 end
16 return NN^u or NN^l
```

Algorithm 3.2: NEUR2BILO Optimization

Input : Evaluation instance $\mathscr{P}'$, trained model NN^l/NN^u. Note that the trained model (NN^l/NN^u) is used on multiple evaluation instances.

1 **if** *approximating upper-level* **then**
2 | $\mathbf{x}^\star \leftarrow$ upper-level solution from the upper-level approximation (Eq. (3.5))
3 **else if** *approximating lower-level* **then**
4 | $\mathbf{x}^\star \leftarrow$ upper-level solution from the lower-level approximation (Eq. (3.7))
5 **end**
6 **return** $\mathbf{x}^\star$

3.B. UPPER- V.S. LOWER-LEVEL APPROXIMATIONS

Example 3.1. *Consider the problem*

$$\begin{aligned} &\min_{x\in\{0,1\}} \; y \\ &s.t. \quad y \in \arg\max_{y\in\{0,1\}} \{y : 2x + y \le 1\}. \end{aligned}$$

Solution $x = 1$ makes the follower's problem infeasible. For solution $x = 0$, the optimal follower solution is $y = 1$ leading to the optimal value 1. Assume that the same trained neural network is used in both approaches; this is possible since leader and follower have the same objective functions. If it predicts $NN(0) = 2$ and $NN(1) = 0$, then the upper-level approximation problem (3.5) will return $x = 1$ which is infeasible whereas the lower-level approximation (3.7) correctly returns $x = 0$.

3.C. PROOFS FOR APPROXIMATION GUARANTEES

This section includes the full analysis of the derived approximation guarantee in Section 3.4.3 for the lower-level approximation with $\text{NN}^l(\mathbf{x};\Theta)$.

Recall that we look at a specific setup for which we derive approximation guarantees: the leader and the follower have the same objective function (i.e., $f(\mathbf{x},\mathbf{y}) = F(\mathbf{x},\mathbf{y})$ for all $\mathbf{x} \in \mathscr{X}, \mathbf{y} \in \mathscr{Y}$), we assume that Assumption 1(i) holds and that the neural network approximates the optimal value of the follower's problem up to an absolute error of $\alpha > 0$, i.e.,

$$|\text{NN}^l(\mathbf{x};\Theta) - \Phi(\mathbf{x})| \le \alpha \quad \text{for all } \mathbf{x} \in \mathscr{X}. \tag{3.9}$$

We furthermore define the parameter Δ as the maximum difference of functions values $f(\mathbf{x},\mathbf{y}) - f(\mathbf{x},\mathbf{y}') \ge 0$ over all $\mathbf{x} \in \mathscr{X}, \mathbf{y},\mathbf{y}' \in \mathscr{Y}$ such that no $\tilde{\mathbf{y}} \in \mathscr{Y}$ exists which has function value $f(\mathbf{x},\mathbf{y}) > f(\mathbf{x},\tilde{\mathbf{y}}) > f(\mathbf{x},\mathbf{y}')$. Note that Δ can be strictly larger than zero if the follower decisions are integer.

For a fixed $\mathbf{x} \in \mathscr{X}$, $\mathbf{y}^\star_{\text{NN}}(\mathbf{x})$ denotes an optimal solution of (3.7). Furthermore, for any given $\mathbf{y} \in \mathscr{Y}$ we denote by $s^\star(\mathbf{x},\mathbf{y})$ an optimal slack-value in Problem (3.7) if the upper- and lower-level variables are fixed to $\mathbf{x}$ and $\mathbf{y}$, respectively.

Observation 3.1. *For any* $\mathbf{x} \in \mathscr{X}$ *and* $\mathbf{y} \in \mathscr{Y}$ *we have*

$$s^\star(\mathbf{x},\mathbf{y}) = \max\{0, \mathrm{NN}^l(\mathbf{x};\Theta) - f(\mathbf{x},\mathbf{y})\}.$$

Lemma 3.1. *Assume the leader and the follower have the same objective function and* $\lambda > 1$*. Then, for any given* $\mathbf{x} \in \mathscr{X}$ *the following conditions hold for the optimal follower solution* $\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})$ *of Problem* (3.7)*:*

- *If* $\mathrm{NN}^l(\mathbf{x};\Theta) \geq \Phi(\mathbf{x})$*, then* $f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) = \Phi(\mathbf{x})$*, i.e.,* $(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$ *is feasible for the original bilevel problem.*
- *If* $\mathrm{NN}^l(\mathbf{x};\Theta) < \Phi(\mathbf{x})$*, then* $\mathrm{NN}^l(\mathbf{x};\Theta) - \frac{1}{\lambda}\Delta \leq f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) \leq \Phi(\mathbf{x})$.

Proof. Case 1: Let $\mathbf{x} \in \mathscr{X}$ for which it holds $\mathrm{NN}^l(\mathbf{x};\Theta) \geq \Phi(\mathbf{x})$ and assume the opposite of the statement is true, i.e., for the optimal reaction $\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})$ in (3.7) it holds that $\Phi(\mathbf{x}) > f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$. Since $\lambda > 0$ and due to Constraint (3.7d) the optimal slack value for solution $\mathbf{x}$ in Problem (3.7) is $s^\star(\mathbf{x},\mathbf{y}) = \mathrm{NN}^l(\mathbf{x};\Theta) - f(\mathbf{x},\mathbf{y})$. Assume $\mathbf{y}^\star(\mathbf{x})$ is the optimal follower reaction in (3.2) for $\mathbf{x}$, then it holds that:

$$\begin{aligned}
&f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) + \lambda s^\star(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))\\
&= f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) + \lambda\left(\mathrm{NN}^l(\mathbf{x};\Theta) - f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))\right)\\
&> f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) + \lambda\left(\mathrm{NN}^l(\mathbf{x};\Theta) - f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))\right) + (\lambda-1)\left(f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) - f(\mathbf{x},\mathbf{y}^\star(\mathbf{x}))\right)\\
&= f(\mathbf{x},\mathbf{y}^\star(\mathbf{x})) + \lambda\left(\mathrm{NN}^l(\mathbf{x};\Theta) - f(\mathbf{x},\mathbf{y}^\star(\mathbf{x}))\right)\\
&= f(\mathbf{x},\mathbf{y}^\star(\mathbf{x})) + \lambda s^\star(\mathbf{x},\mathbf{y}^\star(\mathbf{x}))
\end{aligned}$$

where the first inequality follows since $\lambda > 1$ and $f(\mathbf{x},\mathbf{y}^\star(\mathbf{x})) = \Phi(\mathbf{x}) > f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$ and the latter equality follows from $\mathrm{NN}^l(\mathbf{x};\Theta) \geq \Phi(\mathbf{x}) = f(\mathbf{x},\mathbf{y}^\star(\mathbf{x}))$. The latter result shows that the solution $(\mathbf{x},\mathbf{y}^\star(\mathbf{x}))$ has a strictly better objective value in the surrogate problem (3.7) than $(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$ which contradicts the optimality of $(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$.

Case 2: Let $\mathbf{x} \in \mathscr{X}$ be a leader's decision for which $\mathrm{NN}^l(\mathbf{x};\Theta) < \Phi(\mathbf{x})$ and assume the opposite of the statement, i.e., for the optimal reaction $\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})$ in (3.7) it holds that $\mathrm{NN}^l(\mathbf{x};\Theta) - \frac{1}{\lambda}\Delta > f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$. Hence the optimal slack value in (3.7) is

$$s^\star(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) = \mathrm{NN}^l(\mathbf{x};\Theta) - f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) > \frac{1}{\lambda}\Delta. \tag{3.10}$$

First, assume there exists another feasible solution $\bar{\mathbf{y}}(\mathbf{x})$ for Problem (3.7) with

$$f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) < f(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x})) < \mathrm{NN}^l(\mathbf{x};\Theta)$$

then solution $(\mathbf{x},\bar{\mathbf{y}}(x))$ has a strictly better objective value than $(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$ in (3.7) since increasing the value of f by δ decreases the value of the slack variable by δ which results in a better objective value since $\lambda > 1$, which contradicts the optimality of $(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$.

Second, assume there exists no other feasible solution $\bar{\mathbf{y}}(\mathbf{x})$ for Problem (3.7) with

$$f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) < f(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x})) < \mathrm{NN}^l(\mathbf{x};\Theta).$$

Then there must exists a feasible solution $\bar{\mathbf{y}}(\mathbf{x})$ with $f(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x})) \geq \mathrm{NN}^l(\mathbf{x};\Theta)$ and

$$f(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x})) - f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) \leq \Delta, \tag{3.11}$$

by definition of Δ. In this case, we have

$$\begin{aligned}
&f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) + \lambda s^\star(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) - f(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x})) - \lambda s^\star(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x}))\\
&= f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) + \lambda s^\star(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) - f(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x}))\\
&> f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) + \Delta - f(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x})) \geq -\Delta + \Delta = 0,
\end{aligned}$$

where the first equality follows since $s^\star(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x})) = 0$, the first inequality follows from (3.10) and the last inequality follows from (3.11). In summary, the latter results show that there exists a solution $(\mathbf{x},\bar{\mathbf{y}}(\mathbf{x}))$ for (3.7) which has strictly better objective value than $(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x}))$ which is a contradiction.

Note that the inequality $f(\mathbf{x},\mathbf{y}^\star_{\mathrm{NN}}(\mathbf{x})) \leq \Phi(\mathbf{x})$ follows directly from the definition of $\Phi(\mathbf{x})$. □

The latter lemma states, that if the neural network is overestimating the follower value for a solution $\mathbf{x} \in \mathscr{X}$, then the surrogate problem (3.7) still selects an optimal follower response. However, if the neural network underestimates the value, it may happen that the surrogate problem chooses a follower response for which the objective value is either smaller than the true value or differs by at most $\frac{1}{\lambda}\Delta$ from the estimation. Note that the latter term can be controlled by increasing the penalty λ.

By applying Lemma 3.1 we can bound the approximation guarantee of the lower-level NEUR2BILO.

▷ **Theorem 3.1.** *If the leader and the follower have the same objective function and $\lambda > 1$,* NEUR2BILO *returns a feasible solution $(\mathbf{x}^\star,\mathbf{y}^\star)$ for Problem (3.1) with objective value*

$$f(\mathbf{x}^\star,\mathbf{y}^\star) \leq \mathrm{opt} + 3\alpha + \frac{2}{\lambda}\Delta,$$

where opt *is the optimal value of (3.1) and λ the penalty term in (3.7a).*

Proof. Let $(\mathbf{x}^\star_{\mathrm{NN}},\mathbf{y}^\star_{\mathrm{NN}})$ be an optimal solution of the surrogate problem (3.7). By Lemma 3.1 and by definition (3.8) it follows that

$$\begin{aligned}
\Phi(\mathbf{x}^\star_{\mathrm{NN}}) &\geq f(\mathbf{x}^\star_{\mathrm{NN}},\mathbf{y}^\star_{\mathrm{NN}}) \geq \mathrm{NN}^l(\mathbf{x}^\star_{\mathrm{NN}};\Theta) - \frac{1}{\lambda}\Delta\\
&\geq \Phi(\mathbf{x}^\star_{\mathrm{NN}}) - \alpha - \frac{1}{\lambda}\Delta.
\end{aligned} \tag{3.12}$$

Following the three steps presented in Section 3.4.1 - **Bilevel feasibility**, NEUR2BILO returns a feasible solution $(\mathbf{x}^\star,\mathbf{y}^\star)$ for Problem (3.2) where $\mathbf{x}^\star = \mathbf{x}^\star_{\mathrm{NN}}$ and $f(\mathbf{x}^\star,\mathbf{y}^\star) = \Phi(\mathbf{x}^\star)$. Hence, the following holds:

$$f(\mathbf{x}^\star,\mathbf{y}^\star) = \Phi(\mathbf{x}^\star) \leq f(\mathbf{x}^\star,\mathbf{y}^\star_{\mathrm{NN}}) + \alpha + \frac{1}{\lambda}\Delta. \tag{3.13}$$

Assume $(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star})$ is an optimal bilevel solution of Problem (3.1) and $\mathbf{y}^{\star\star}_{\text{NN}}$ the optimal follower response in the surrogate problem (3.7). Then we have

$$f(\mathbf{x}^{\star}, \mathbf{y}^{\star}_{\text{NN}}) + s^{\star}(\mathbf{x}^{\star}, \mathbf{y}^{\star}_{\text{NN}}) \leq f(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}_{\text{NN}}) + s^{\star}(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}_{\text{NN}})$$

since $(\mathbf{x}^{\star}_{\text{NN}}, \mathbf{y}^{\star}_{\text{NN}})$ is an optimal solution of (3.7) with objective value given by (3.7a). From the latter inequality we obtain

$$\begin{aligned}
f(\mathbf{x}^{\star}, \mathbf{y}^{\star}_{\text{NN}}) &\leq f(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}_{\text{NN}}) + s^{\star}(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}_{\text{NN}}) - s^{\star}(\mathbf{x}^{\star}, \mathbf{y}^{\star}_{\text{NN}}) \\
&\leq f(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}) + s^{\star}(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}_{\text{NN}}) \\
&\leq f(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}) + \text{NN}^{l}(\mathbf{x}^{\star\star}; \Theta) - f(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}_{\text{NN}}) \\
&\leq f(\mathbf{x}^{\star\star}, \mathbf{y}^{\star\star}) + \Phi(\mathbf{x}^{\star\star}) + \alpha - (\Phi(\mathbf{x}^{\star\star}) - \alpha - \frac{1}{\lambda}\Delta) \\
&= \text{opt} + 2\alpha + \frac{1}{\lambda}\Delta
\end{aligned}$$

where the second inequality follows from $s^{\star}(\mathbf{x}^{\star}, \mathbf{y}^{\star}_{\text{NN}}) \geq 0$ and $\mathbf{y}^{\star\star}$ being an optimal follower solution for $\mathbf{x}^{\star\star}$. The third inequality follows from Observation 3.1 and the fourth inequality follows from (3.8) and from (3.12) applied to $\mathbf{x}^{\star\star}$.

Together with (3.13), this completes the proof. □

3.D. PROBLEM FORMULATIONS

3.D.1. KNAPSACK INTERDICTION

The bilevel knapsack problem with interdiction constraints as described in Tang et al. [195] is given by

$$\begin{aligned}
\min_{\mathbf{x}\in\{0,1\}^n, \mathbf{y}} \quad & \sum_{i=1}^{n} p_i y_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq k, \\
& \mathbf{y} \in \operatorname*{arg\,max}_{\mathbf{y}'\in\{0,1\}^n} \quad \sum_{i=1}^{n} p_i y'_i \\
& \qquad \text{s.t.} \quad \sum_{i=1}^{n} a_i y'_i \leq b, \\
& \qquad\qquad y'_i + x_i \leq 1, i \in [n],
\end{aligned}$$

where $\mathbf{x}$ are the leader's variables and $\mathbf{y}$ are that of the follower. The leader decides to interdict (a maximum of k) items of the knapsack solved in the follower's problem with n the number of items, p_i the profits, a_i the weight of item i, respectively, and the budget of the knapsack is denoted by b.

3.D.2. CRITICAL NODE PROBLEM

The critical node problem is described in Carvalho et al. [50] as follows

$$\begin{aligned}
\max_{\mathbf{x}\in\{0,1\}^n,\mathbf{y}} \quad & \sum_{i=1}^{n}\Big(p_i^d\big((1-x_i)(1-y_i)+\eta x_i y_i+\epsilon x_i(1-y_i)+\delta(1-x_i)y_i\big)\Big)\\
\text{s.t.} \quad & \sum_{i=1}^{n} d_i x_i \le D,\\
& \mathbf{y}\in \operatorname*{argmax}_{\mathbf{y}'\in\{0,1\}^n} \quad \sum_{i=1}^{n}\Big(p_i^a\big(-\gamma(1-x_i)(1-y_i')+(1-x_i)y_i'+(1-\eta)x_i y_i'\big)\Big)\\
& \qquad\qquad \text{s.t.} \quad \sum_{i=1}^{n} a_i y_i' \le A,
\end{aligned}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the leader's and follower's variables, respectively. Here, $\mathbf{x}$ denotes the decisions of the leader (defender) who selects which nodes to deploy resources to defend a set of nodes, while $\mathbf{y}$ are the decisions for the follower (attacker) for which nodes to attack. d_i and a_i are the costs for the x_i and y_i, respectively. D and A are the budgets for the defender and attacker, respectively. In this problem, the bilinearity arises in the objectives of both the leader and follower, which results in four outcomes for each possible combination of defending and attacking a node i. The first outcome arises when both the leader and follower do not select the node. In this case, the leader receives the full profit, p_i^d, and the follower pays an opportunity cost of $-\gamma p_i^a$ for not attacking an undefended node. Second is a successful attack, wherein the leader receives a reduced profit of δp_i^d and the follower receives the full profit p_i^a. Third is a mitigated attack, wherein the leader receives a profit of ηp_i^d for a degradation in operations, while the follower receives a profit of $(1-\eta)p_i^a$ for a mitigated attack. Fourth is a mitigation without an attack, wherein the leader receives a profit ϵp_i^d for a degradation in operations, while the follower receives a profit of 0 for a mitigated attack.

3.D.3. DONOR-RECIPIENT PROBLEM

The donor-recipient problem as described in Ghatkar et al. [84], and introduced in Morton et al. [152], is formulated as

$$\begin{aligned}
\max_{\mathbf{x}\in[0,1]^n,\mathbf{y},y_0} \quad & \sum_{i=1}^{n} w_i y_i\\
\text{s.t.} \quad & \sum_{i=1}^{n} c_i x_i \le B_d,\\
& (\mathbf{y},y_0)\in \operatorname*{argmax}_{\mathbf{y}'\in\{0,1\}^n,y_0'\in[0,1]} \quad \sum_{i=1}^{n} v_i y_i' + v_0 y_0'\\
& \qquad\qquad \text{s.t.} \quad \sum_{i=1}^{n}(c_i - c_i x_i)y_i' + c_0 y_0' \le B_r,
\end{aligned}$$

where the leader's decisions $\mathbf{x}$ represent those of the donor and the follower's decisions $(\mathbf{y},y_0)$ the ones of the recipient. The profit of project i is given as w_i for the leader and v_i

for the follower, the cost as c_i, and the budget of the leader, resp. follower, as B_d and B_r. Next to the projects, the recipient can allocate its budget to external projects, for which the profit is given as v_0 and the cost c_0.

3.D.4. DISCRETE NETWORK DESIGN PROBLEM

We use the standard formulation from Section 3.1 following the computational benchmarking study of Rey [171] and the code provided by the author [2].

3.E. LEARNING-BASED APPROACH OF ZHOU ET AL. [233]

This section compares our approach to a recent learning-based approach from Zhou et al. [233] based on code provided by the author[3]. We specifically compare the input-supermodular neural network (ISNN), i.e., the best-performing model from Zhou et al. [233]. Their approach requires sampling and training for each instance, which is reflected in the time, whereas the model for NN^l and NN^u can be trained once and evaluated across multiple instances, so the data collection and training time are excluded. We also restrict ISNN to run for one iteration given Zhou et al. [233] report very minimal improvements when increasing the number of iterations. Moreover, one iteration requires the least amount of time. Table 3.3 reports the MRE and time for each method for the knapsack instances from Tang et al. [195]. Generally, we can see a significant improvement over ISNN in both computing time and MRE.

Table 3.3: Comparison to ISNN from Zhou et al. [233] on the knapsack interdiction problem. n and k denote the number of items and the interdiction budget, respectively. We directly evaluate on the 180 instances (10 per size) of Tang et al. [195]; each value is the average over 10 instances. We compare the upper- and lower-level approximations, as well as the no-learning baseline (G-VFA) and the exact algorithm (B&C).

n	k	ISNN		NN^l		NN^u		G-VFA		B&C	
		MRE	Time	MRE	Time	MRE	Time	MRE	Time	MRE	Time
18	5	10.50	254.35	1.48	0.59	1.48	0.34	1.82	**0.14**	**0.00**	9.55
18	9	46.50	227.49	1.51	0.59	1.51	0.43	3.97	**0.22**	**0.00**	5.81
18	14	302.10	217.62	**0.00**	0.22	**0.00**	0.17	64.22	**0.03**	**0.00**	0.39
20	5	8.56	262.01	0.41	0.62	0.41	0.45	2.19	**0.25**	**0.00**	23.18
20	10	54.07	236.74	0.99	0.66	0.99	0.58	0.99	**0.36**	**0.00**	10.27
20	15	447.29	229.41	3.57	0.32	3.57	0.19	23.39	**0.02**	**0.00**	0.94
22	6	17.32	266.55	0.71	0.19	0.71	**0.18**	0.42	0.18	**0.00**	42.30
22	11	66.24	247.97	1.01	0.28	1.01	**0.28**	1.08	0.33	**0.00**	16.26
22	17	485.75	241.61	14.43	0.24	14.43	0.15	14.43	**0.13**	**0.00**	0.68
25	7	14.75	280.71	0.44	2.66	0.44	2.42	0.44	**0.64**	**0.00**	137.96
25	13	61.57	264.09	1.42	2.75	1.42	2.79	3.85	**1.24**	**0.00**	48.43
25	19	424.92	262.04	2.49	0.48	2.49	0.38	2.49	**0.13**	**0.00**	1.77
28	7	19.17	297.44	0.39	0.67	0.39	0.74	0.26	**0.62**	**0.00**	309.18
28	14	73.61	286.02	0.75	2.10	0.75	1.45	1.37	**1.29**	**0.00**	120.74
28	21	423.15	279.51	1.14	0.45	1.14	0.49	3.16	**0.31**	**0.00**	4.92
30	8	21.01	305.67	**0.00**	1.54	**0.00**	1.54	0.43	**0.97**	**0.00**	792.44
30	15	68.19	295.92	0.49	3.64	0.49	3.06	0.75	**1.35**	**0.00**	187.23
30	23	416.03	290.94	2.29	1.08	2.29	0.73	4.48	**0.25**	**0.00**	5.65
Average [195]		164.49	263.67	1.86	1.06	1.86	0.91	7.21	**0.47**	**0.00**	95.43

[2] https://github.com/davidrey123/DNDP/
[3] https://github.com/bozlamberth/LearnBilevel/

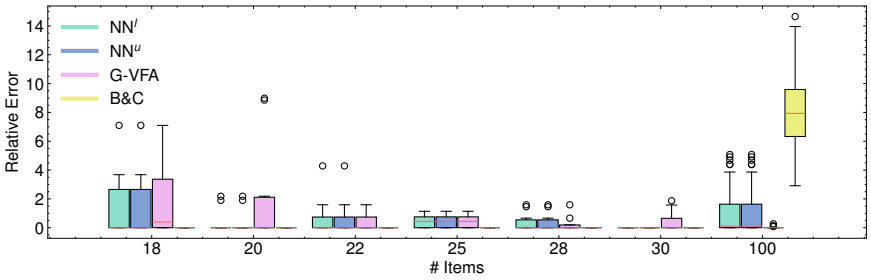


Figure 3.1: Box plot of relative errors for KIP with interdiction budget of $k = n/4$.

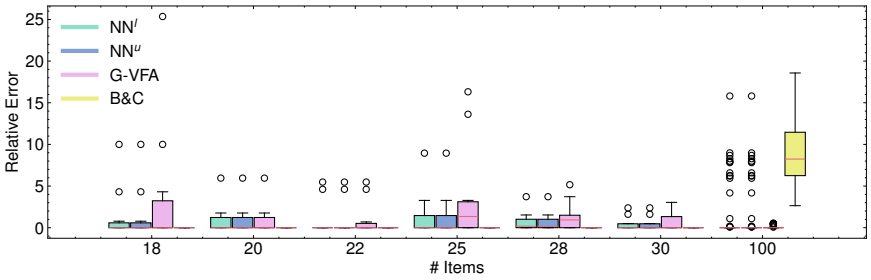


Figure 3.2: Box plot of relative errors for KIP with interdiction budget of $k = n/2$.

3.F. OBJECTIVE & INCUMBENT RESULTS

This section reports the more detailed information related to the objective values for each problem. Objective results for each problem are given in Tables 3.4-3.7. In addition, for KIP and CNP, as the solver from Fischetti et al. [77] provides easily accessible incumbent solutions, we include two additional metrics.

- The first metric “Solver Time Ratio" measures the time it takes the solver to obtain an equally good (or better) incumbent solution, divided by the solving time of the respective approximation. The number in brackets to the right indicates the number of instances for which the solver finds an equivalent solution.

- The second metric “Solver Relative Error at Time" measures the relative error of the best solution found by the solver compared to the respective approximation. The value in brackets to the right indicates the number of instances for which the solver finds an incumbent before the approximation is done solving.

3.G. DISTRIBUTIONAL RESULTS FOR RELATIVE ERROR

This section provides the box plots of the experimental results. For the KIP instances, the results are given in Figures 3.1 to 3.3 for different budget values. Figure 3.4 gives the results for CNG and Figures 3.5 and 3.6 for the DNDP experiments for different edge numbers.

Table 3.4: KIP objective and incumbent results split across two sub-tables for better readability. Each row is averaged over 10 instances, except for $n = 100$, which averages over 100 instances. NN^l and NN^u specify lower- and upper-level approximations. All times in seconds.

n	k	Objective				Mean Relative Error (%)			
		NN^l	NN^u	G-VFA	B&C	NN^l	NN^u	G-VFA	B&C
18	5	308.30	308.30	309.20	**303.50**	1.48	1.48	1.82	**0.00**
18	9	145.60	145.60	149.10	**143.40**	1.51	1.51	3.97	**0.00**
18	14	**31.00**	**31.00**	51.40	**31.00**	**0.00**	**0.00**	64.22	**0.00**
20	5	390.30	390.30	397.60	**388.50**	0.41	0.41	2.19	**0.00**
20	10	165.40	165.40	165.40	**163.70**	0.99	0.99	0.99	**0.00**
20	15	33.40	33.40	41.90	**31.40**	3.57	3.57	23.39	**0.00**
22	6	385.50	385.50	384.30	**382.70**	0.71	0.71	0.42	**0.00**
22	11	163.20	163.20	163.30	**161.00**	1.01	1.01	1.08	**0.00**
22	17	35.20	35.20	35.20	**29.20**	14.43	14.43	14.43	**0.00**
25	7	438.20	438.20	438.20	**436.20**	0.44	0.44	0.44	**0.00**
25	13	194.90	194.90	199.90	**191.50**	1.42	1.42	3.85	**0.00**
25	19	43.30	43.30	43.30	**41.80**	2.49	2.49	2.49	**0.00**
28	7	518.30	518.30	517.60	**516.10**	0.39	0.39	0.26	**0.00**
28	14	224.90	224.90	226.80	**223.40**	0.75	0.75	1.37	**0.00**
28	21	46.70	46.70	48.20	**46.20**	1.14	1.14	3.16	**0.00**
30	8	**536.30**	**536.30**	538.70	**536.30**	**0.00**	**0.00**	0.43	**0.00**
30	15	231.20	231.20	231.90	**230.00**	0.49	0.49	0.75	**0.00**
30	23	49.00	49.00	50.40	**47.50**	2.29	2.29	4.48	**0.00**
100	25	2,164.71	2,164.69	**2,145.07**	2,318.99	0.93	0.93	**0.00**	8.09
100	50	965.37	965.37	**956.76**	1,043.71	0.96	0.96	**0.04**	8.96
100	75	**245.01**	**245.01**	245.08	259.95	**0.08**	**0.08**	0.12	5.87

n	k	Solving Time				Solver Time Ratio			Solver Relative Error at Time		
		NN^l	NN^u	G-VFA	B&C	NN^l	NN^u	G-VFA	NN^l	NN^u	G-VFA
18	5	0.59	0.34	**0.14**	9.55	24.48 (10)	35.86 (10)	177.39 (10)	- (0)	- (0)	- (0)
18	9	0.59	0.43	**0.22**	5.81	12.31 (10)	18.52 (10)	73.49 (10)	- (0)	- (0)	- (0)
18	14	0.22	0.17	**0.03**	0.39	1.87 (10)	2.86 (10)	31.9 (10)	44.0 (2)	41.5 (2)	- (0)
20	5	0.62	0.45	**0.25**	23.18	50.68 (10)	65.56 (10)	420.04 (10)	- (0)	- (0)	- (0)
20	10	0.66	0.58	**0.36**	10.27	18.39 (10)	22.03 (10)	80.0 (10)	- (0)	- (0)	- (0)
20	15	0.32	0.19	**0.02**	0.94	2.82 (10)	4.75 (10)	54.29 (10)	- (0)	- (0)	- (0)
22	6	0.19	**0.18**	0.18	42.30	228.97 (10)	249.8 (10)	714.31 (10)	- (0)	- (0)	- (0)
22	11	0.28	**0.28**	0.33	16.26	69.05 (10)	74.99 (10)	129.04 (10)	- (0)	- (0)	- (0)
22	17	0.24	0.15	**0.13**	0.68	3.09 (10)	5.48 (10)	29.34 (10)	18.0 (1)	- (0)	- (0)
25	7	2.66	2.42	**0.64**	137.96	58.27 (10)	61.24 (10)	1102.38 (10)	28.69 (2)	28.69 (2)	28.69 (2)
25	13	2.75	2.79	**1.24**	48.43	21.14 (10)	25.13 (10)	67.86 (10)	- (0)	- (0)	- (0)
25	19	0.48	0.38	**0.13**	1.77	3.98 (10)	4.81 (10)	38.29 (10)	- (0)	- (0)	- (0)
28	7	0.67	0.74	**0.62**	309.18	671.57 (10)	518.35 (10)	1033.45 (10)	29.28 (8)	29.28 (8)	29.48 (8)
28	14	2.10	1.45	**1.29**	120.74	59.99 (10)	84.36 (10)	120.05 (10)	19.84 (2)	19.84 (2)	16.14 (2)
28	21	0.45	0.49	**0.31**	4.92	12.95 (10)	11.66 (10)	38.98 (10)	- (0)	- (0)	- (0)
30	8	1.54	1.54	**0.97**	792.44	497.06 (10)	455.29 (10)	1924.47 (10)	27.07 (10)	27.07 (10)	26.58 (10)
30	15	3.64	3.06	**1.35**	187.23	56.14 (10)	66.07 (10)	254.88 (10)	86.11 (6)	86.11 (6)	85.19 (6)
30	23	1.08	0.73	**0.25**	5.65	7.5 (10)	8.79 (10)	48.27 (10)	- (0)	- (0)	- (0)
100	25	10.02	8.40	**4.19**	3,600.40	- (0)	- (0)	- (0)	34.36 (100)	34.99 (100)	37.93 (100)
100	50	51.68	**49.28**	53.74	3,600.44	23.56 (5)	26.24 (5)	- (0)	59.36 (100)	60.81 (100)	60.48 (100)
100	75	24.69	**23.78**	35.27	3,600.52	133.35 (4)	152.72 (4)	138.07 (5)	177.01 (100)	196.86 (100)	193.94 (100)

Table 3.5: CNP objective and incumbent results. Each row averaged over 300 instances. All times in seconds.

$\|V\|$	Objective			Mean Relative Error (%)			Times			Solver Time Ratio		Solver Relative Error at Time	
	NN^l	NN^u	B&C	NN^l	NN^u	B&C	NN^l	NN^u	B&C	NN^l	NN^u	NN^l	NN^u
10	224.47	225.10	**228.63**	3.20	2.75	**1.01**	1.69	**1.19**	3,600.80	136.66 (288)	191.34 (289)	269.0 (1)	- (0)
25	562.72	566.23	**572.51**	2.60	1.77	**0.73**	1.69	**1.19**	3,600.80	736.84 (275)	3934.02 (271)	2.22 (248)	2.57 (124)
50	1,139.27	1,143.95	**1,148.17**	1.42	0.98	**0.67**	1.69	**1.19**	3,600.80	718.74 (225)	3840.41 (183)	1.94 (295)	3.17 (190)
100	2,285.15	**2,297.47**	2,272.30	1.12	**0.56**	1.79	1.69	**1.19**	3,600.80	645.37 (131)	926.6 (90)	2.4 (283)	2.96 (278)
300	6,781.91	**6,882.42**	6,755.07	2.01	**0.33**	2.32	1.69	**1.19**	3,600.80	41.65 (166)	167.38 (47)	1.49 (245)	2.65 (243)
500	11,348.60	**11,439.25**	11,208.43	1.33	**0.45**	2.47	1.69	**1.19**	3,600.80	106.9 (83)	99.48 (15)	1.51 (206)	2.45 (205)

Table 3.6: DRP objective results. Each row corresponds to a single instance from dataset 15, i.e., the most challenging instances from Ghatkar et al. [84]. All times in seconds.

Instance #	Objective			Relative Error (%)			Times		
	NN^l	NN^u	B&C+	NN^l	NN^u	B&C+	NN^l	NN^u	B&C+
1	34,356.00	**59,524.00**	47,206.00	42.28	**0.00**	20.69	**0.09**	1.44	3,600.09
2	33,713.00	**54,764.00**	39,526.00	38.44	**0.00**	27.82	**0.12**	1.52	3,600.08
3	36,717.00	**66,967.00**	46,792.00	45.17	**0.00**	30.13	**0.14**	2.85	3,600.07
4	36,414.00	**54,908.00**	44,486.00	33.68	**0.00**	18.98	**0.07**	1.68	3,637.23
5	33,090.00	**59,627.00**	43,355.00	44.51	**0.00**	27.29	**0.10**	1.96	3,600.07
6	36,691.00	**56,603.00**	39,006.00	35.18	**0.00**	31.09	**0.08**	2.93	3,600.10
7	31,354.00	**55,569.00**	43,443.00	43.58	**0.00**	21.82	**0.09**	1.58	3,600.14
8	35,710.00	**54,414.00**	39,839.00	34.37	**0.00**	26.79	**0.09**	0.87	3,600.10
9	38,961.00	**61,869.00**	45,288.00	37.03	**0.00**	26.80	**0.16**	4.55	3,600.16
10	36,965.00	**60,488.00**	43,194.00	38.89	**0.00**	28.59	**0.12**	3.57	3,600.10
Averaged	35,397.10	**58,473.30**	43,213.50	39.31	**0.00**	26.00	**0.11**	2.30	3,603.82

Table 3.7: DNDP objective results. Each is averaged across 10 instances. All times in seconds.

# of edges	budget	Objective						
		NN^l	NN^u	GBT^l	GBT^u	MKKT-5	MKKT-10	MKKT-30
10	0.25	6,201.25	6,145.27	6,214.37	6,147.02	6,484.98	6,155.69	**6,129.65**
10	0.5	5,532.92	5,557.28	5,531.27	5,640.77	5,849.03	5,618.41	**5,492.23**
10	0.75	5,202.82	5,246.29	5,211.07	5,225.02	5,477.72	**5,179.39**	5,181.30
20	0.25	5,478.52	5,272.98	5,272.07	5,210.07	5,535.14	5,423.02	**5,180.67**
20	0.5	4,347.58	4,490.04	4,356.47	4,390.52	4,563.35	4,416.83	**4,330.21**
20	0.75	4,084.19	4,085.09	4,061.68	4,135.70	4,363.00	4,057.72	**4,053.02**

# of edges	budget	Relative Error (%)							Times			
		NN^l	NN^u	GBT^l	GBT^u	MKKT-5	MKKT-10	MKKT-30	NN^l	NN^u	GBT^l	GBT^u
10	0.25	1.21	0.36	1.43	0.38	5.78	0.51	**0.10**	2.95	**0.01**	3.19	0.09
10	0.5	0.73	1.22	0.72	2.74	6.47	2.17	**0.00**	3.35	**0.01**	3.66	0.07
10	0.75	0.47	1.32	0.63	0.91	5.80	**0.02**	0.06	2.80	**0.00**	3.02	0.06
20	0.25	6.05	2.64	2.38	1.41	7.78	5.12	**0.85**	5.02	**0.02**	5.02	0.23
20	0.5	1.01	4.36	1.22	2.02	6.00	2.52	**0.64**	4.91	**0.03**	5.02	0.21
20	0.75	0.85	0.91	0.32	2.14	7.87	0.22	**0.11**	4.47	**0.01**	4.69	0.13

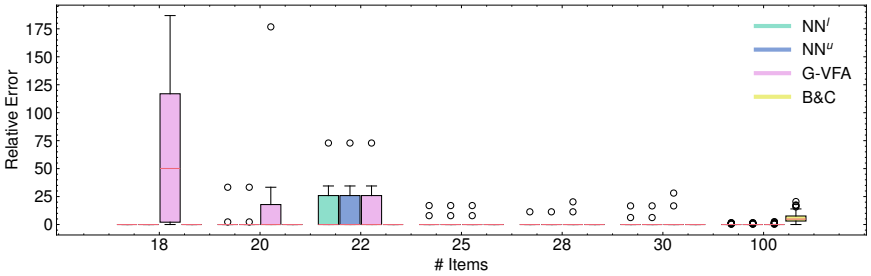


Figure 3.3: Box plot of relative errors for KIP with interdiction budget of $k = 3n/4$.

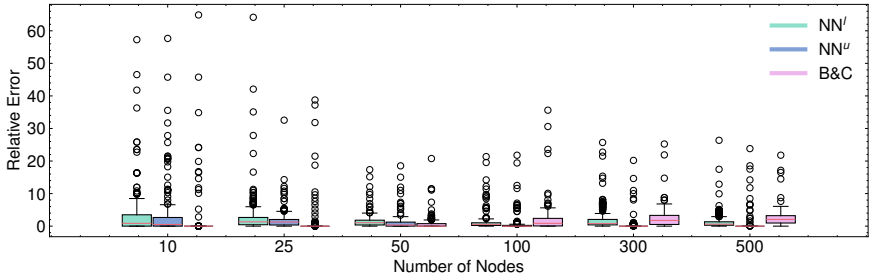


Figure 3.4: Box plot of relative errors for CNP. B&C does not find any upper-level solutions for 2 of the 300 instances of size $|V| = 500$, so these are excluded from the plot.

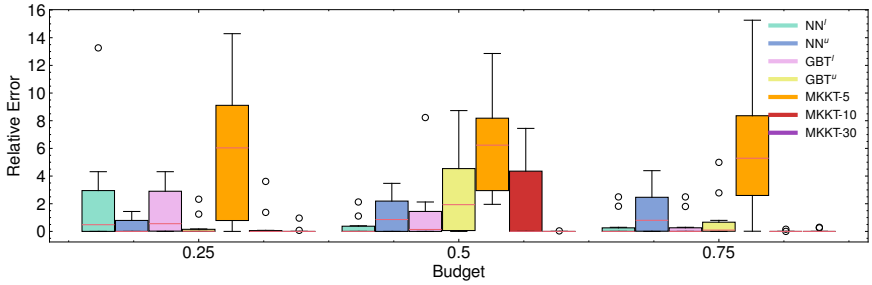


Figure 3.5: Box plot of relative errors for DNDP with 10 edges. MKKT-{5,10,30} corresponds to MKKT run with each respective time limit.

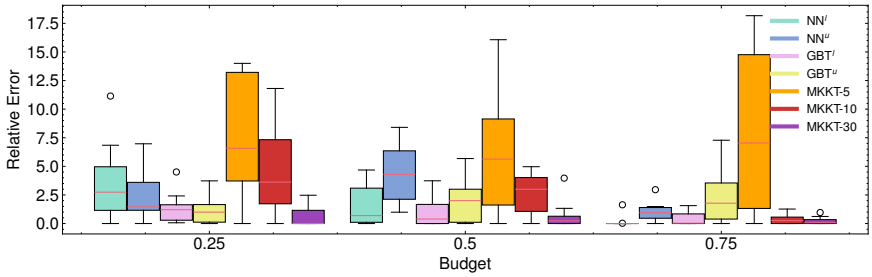


Figure 3.6: Box plot of relative errors for DNDP with 20 edges. MKKT-{5,10,30} corresponds to MKKT run with each respective time limit.

Table 3.8: KIP results comparing NN^l, NN^d, and NN^n. Each row is an average over 10 instances, except for $n = 100$, which is an average over 100 instances. All times in seconds.

n	k	Objective			Mean Relative Error (%)			Times		
		NN^l	NN^d	NN^n	NN^l	NN^d	NN^n	NN^l	NN^d	NN^n
18	5	**308.30**	308.40	318.40	**0.00**	0.03	3.28	**0.59**	0.83	1.06
18	9	**145.60**	**145.60**	152.90	**0.00**	**0.00**	6.70	**0.59**	1.21	0.81
18	14	**31.00**	37.50	40.00	**0.00**	16.91	48.23	**0.22**	0.32	0.35
20	5	**390.30**	**390.30**	413.90	**0.00**	**0.00**	6.48	**0.62**	0.79	1.38
20	10	**165.40**	**165.40**	175.90	**0.00**	**0.00**	6.60	**0.66**	1.47	1.76
20	15	33.40	**32.50**	55.70	**3.33**	14.29	100.91	**0.32**	0.38	0.96
22	6	**385.50**	386.80	403.00	**0.00**	0.27	4.56	**0.19**	0.37	0.80
22	11	163.20	**162.10**	179.20	0.55	**0.07**	11.83	**0.28**	0.85	1.23
22	17	**35.20**	**35.20**	49.00	5.15	**4.63**	69.91	0.24	**0.19**	0.41
25	7	**438.20**	**438.20**	446.50	**0.00**	**0.00**	1.98	2.66	**2.40**	3.85
25	13	**194.90**	195.50	206.50	**0.00**	0.26	6.67	**2.75**	3.25	4.83
25	19	**43.30**	**43.30**	64.40	**1.69**	**1.69**	92.49	**0.48**	0.74	1.84
28	7	**518.30**	**518.30**	532.20	**0.00**	**0.00**	2.80	**0.67**	0.83	2.37
28	14	**224.90**	**224.90**	234.70	**0.00**	**0.00**	4.60	**2.10**	2.69	3.72
28	21	**46.70**	49.90	60.70	**0.00**	7.48	37.45	**0.45**	0.83	1.67
30	8	**536.30**	537.10	537.70	**0.00**	0.18	0.25	**1.54**	1.86	3.07
30	15	**231.20**	**231.20**	232.80	**0.16**	**0.16**	0.82	**3.64**	4.18	5.03
30	23	**49.00**	50.70	51.90	**0.00**	2.79	4.48	**1.08**	1.50	1.76
100	25	2,164.71	**2,164.13**	2,168.52	0.04	**0.01**	0.22	**10.02**	12.28	19.81
100	50	965.37	**965.26**	974.28	0.03	**0.02**	1.01	**51.68**	61.09	72.86
100	75	**245.01**	245.10	262.66	**0.04**	0.08	8.18	**24.69**	30.48	81.30

3.H. ABLATION

3.H.1. LOWER-LEVEL VALUE FUNCTION CONSTRAINTS

In this section, we present an ablation study comparing alternative types of value function approximation (VFA) for the lower-level approximation on the KIP. Namely, we compare the approach used in the main body of the chapter, NN^l, which utilizes a slack variable to ensure feasibility. In addition, we include NN^n which does not use a slack at all, and NN^d, which uses the largest error in the validation set to scale the prediction down. Table 3.8 reports objectives, relative errors, and solving times of each method. In general, the solution quality of NN^l slightly exceeds that of NN^d, while NN^n does significantly worse. The latter results is unsurprising given that any underestimation will cause a loss of feasibility for potentially high quality upper-level decisions. NN^l is additionally generally the fastest to optimize as well.

3.H.2. THE EFFECT OF λ

In this section, we present results with $\lambda = 0.1$ for DNDP. Table 3.9 presents relative error and solving times for this setting. Notably, this choice of λ tends to provide higher quality solutions than $\lambda = 1$, as reported in the main body of the chapter in Table 3.2. Tuning this hyperparameter further can thus improve the already strong numerical results reported for DNDP, and possibly other problems.

3.H.3. GREEDY FEATURES FOR KNAPSACK

This section examines the impact of using greedy features on the KIP problem. We specifically compare a model trained purely on the coefficients to a model trained on the coefficients with additional features derived from KIP-specific greedy heuristics. From Table 3.10, there is a clear advantage with the greedy features in terms of solution quality at the cost of increased solving time.

Table 3.9: DNDP results for $\lambda = 0.1$. Each is averaged across 10 instances. NN^l and GBT^l are the learning-based formulations with slack for the lower-level approximation. NN^u and GBT^u are the learning-based formulations for the upper-level approximation.

# of edges	budget	NN^l		NN^u		GBT^l		GBT^u		MKKT		
		MRE	Time	MRE	Time	MRE	Time	MRE	Time	MRE-5	MRE-10	MRE-30
10	0.25	2.44	2.63	0.36	**0.01**	1.43	3.26	0.38	0.09	5.78	0.51	**0.10**
10	0.5	0.39	2.90	1.22	**0.01**	1.33	3.54	2.74	0.07	6.47	2.17	**0.00**
10	0.75	0.48	2.23	1.32	**0.00**	0.47	2.19	0.91	0.06	5.80	**0.02**	0.06
20	0.25	3.62	5.02	2.46	**0.02**	1.36	5.02	1.23	0.23	7.59	4.96	**0.67**
20	0.5	1.56	4.91	4.41	**0.03**	0.98	5.02	2.06	0.21	6.05	2.57	**0.69**
20	0.75	**0.17**	3.44	1.03	**0.01**	0.43	4.75	2.27	0.13	8.00	0.35	0.23
Average		1.44	3.52	1.80	**0.01**	1.00	3.96	1.60	0.13	6.61	1.76	**0.29**

Table 3.10: KIP results comparing NN^l with and without greedy-based features NN^d. Each row averaged over 10 instances, except for $n = 100$, which is an average over 100 instances. All times in seconds.

n	k	Objective		Mean Relative Error (%)		Times	
		NN^l greedy	NN^l no greedy	NN^l greedy	NN^l no greedy	NN^l greedy	NN^l no greedy
18	5	**308.30**	314.90	**0.85**	2.93	0.59	**0.06**
18	9	**145.60**	150.50	**1.17**	4.28	0.59	**0.07**
18	14	**31.00**	41.60	**0.00**	55.04	0.22	**0.05**
20	5	**390.30**	404.40	**0.00**	3.71	0.62	**0.06**
20	10	**165.40**	172.00	**0.55**	4.06	0.66	**0.05**
20	15	**33.40**	36.50	**0.00**	7.31	0.32	**0.06**
22	6	**385.50**	390.60	**0.59**	1.88	0.19	**0.07**
22	11	**163.20**	170.80	**0.00**	4.31	0.28	**0.07**
22	17	**35.20**	39.10	**7.91**	31.93	0.24	**0.06**
25	7	**438.20**	446.30	**0.11**	1.67	2.66	**0.08**
25	13	**194.90**	197.20	**0.89**	2.58	2.75	**0.07**
25	19	**43.30**	49.10	**0.00**	13.53	0.48	**0.07**
28	7	**518.30**	537.70	**0.15**	3.63	0.67	**0.07**
28	14	**224.90**	225.90	**0.21**	0.61	2.10	**0.08**
28	21	**46.70**	52.10	**0.00**	11.85	0.45	**0.08**
30	8	**536.30**	556.50	**0.00**	3.61	1.54	**0.08**
30	15	**231.20**	233.70	**0.21**	1.20	3.64	**0.09**
30	23	**49.00**	51.30	**0.00**	4.97	1.08	**0.08**
100	25	**2,164.71**	2,473.08	**0.00**	14.22	10.02	**0.54**
100	50	**965.37**	1,062.92	**0.04**	10.23	51.68	**0.52**
100	75	**245.01**	313.44	**0.00**	27.62	24.69	**0.53**

3.I. COMPUTING SETUP

The experiments for the benchmarks were run on a computing cluster with an Intel Xeon CPU E5-2683 and Nvidia Tesla P100 GPU with 64GB of RAM (for training). Pytorch 2.0.1 [164] was used for all neural network models and scikit-learn 1.4.0 was used for gradient-boosted trees in the DNDP [166]. Gurobi 11.0.1 [92] was used as the MILP solver and `gurobi-machinelearning` 1.4.0 was used to embed the learning models into MILPs.

3.J. MACHINE LEARNING DETAILS

3.J.1. MODELS, FEATURES, & HYPERPARAMETERS

For all problems, we derive features that correspond to each upper-level decision variable, as well as general instance features.

KIP, CNP, DRP

For KIP, CNP, DRP, we have n decisions in both the upper- and lower-level of the problems. For the learning model, we utilize a set-based architecture [226], wherein we first represent the objective and constraint coefficients for each upper-level and lower-level decision, independent of the decision ($\mathbf{f}_i$). Each of these are passed through a feed-forward network with shared parameters (Ψ_d) to compute an m-dimension embedding. The embeddings are then summed and passed through another feed-forward network (Ψ_s) to compute the instance's k-dimensional embedding. This instance embedding is then concatenated with features related to the upper- and lower-level that are dependent on the decision ($h(x_i)$). The concatenated vector is passed through a feed-forward network with shared parameters (Ψ_v) to predict n scalar values (i.e., one for each decision). The final prediction is equal to the dot product of the n predictions with the objective function coefficients of the upper- or lower-level problem, depending on the type of value function approximation. This final step exploits the separable nature of the objective functions in question as they can all be expressed as $\sum_{i=1}^{n} c_i z_i$, where c_i is a *known* coefficient and z_i is a decision variable or a function of a set of decision variables with index i. The objectives for KIP, CNP, and DRP all satisfy this property. We leverage this knowledge of the coefficients of separable objective functions as an inductive bias in the design of the learning architecture to facilitate convergence to accurate models. The decision-dependent and decision-independent features are summarized in Table 3.11. One minor remark for KIP is that since it is an interdiction problem, we multiply the concatenated vector, i.e., the input to Ψ_v, by $(1 - x_i)$ as a mask given that the follower cannot select the same items as the leader. For all instances, we do not perform systematic hyperparameter tuning. The sub-networks Ψ_d, Ψ_s, Ψ_v are feed-forward networks with one hidden layer of dimension 128. The decision-independent feature embedding dimension (m) is 64, and the instance embedding dimension (k) is 32. We use a batch size of 32, a learning rate of 0.01, and Adam [117] as an optimizer.

DNDP

We train neural network models (one hidden layer, 16 neurons, a learning rate of 0.01 with the Adam optimizer) and gradient-boosted trees (default scikit-learn hyperparameters, except for `n_estimators` = 50). The inputs to these models are 30-dimensional binary vectors representing the subset of links selected by the leader.

Table 3.11: Features for KIP, CNP, and DRP. Most features are derived directly from the objective and constraint coefficients, so refer to Appendix 3.D for the definitions. For KIP, additional features are computed using simple greedy heuristics. We compute x_i^{dg}, y_i^{dg}, obj^{dg}, which correspond to a purely greedy strategy, i.e., the upper-level interdicts the k items with the largest profit to cost ratio (p_i/a_i) and the lower-level decisions are the largest remaining highest profit to cost ratio items. For $h(x_i)$ in KIP, we also include lower-level decisions based on G-VFA (y_i^g).

Problem	Type	Features
KIP	$\mathbf{f}_i$	$\frac{p_i/a_i}{\max_i\{p_i/a_i\}}$, p_i, a_i, k/n, x_i^{dg}, y_i^{dg}, obj^{dg}/n
	$h(x_i)$	$\mathbf{f}_i$, x_i, y_i^g
CNP	$\mathbf{f}_i$	$\frac{p_i^d/d_i}{\max_i\{p_i^d/d_i\}}$, $\frac{p_i^a/a_i}{\max_i\{p_i^a/a_i\}}$, d_i, a_i, p_i^a, p_i^d, γ, η, ϵ, δ, A, D
	$h(x_i)$	$\mathbf{f}_i$, x_i, $-\gamma(1-x_i)$, $(1-x_i)$, $(1-\eta)x_i$
DRP	$\mathbf{f}_i$	$\frac{w_i/c_i}{\max_i\{w_i/c_i\}}$, $\frac{v_i/c_i}{\max_i\{c_i/v_i\}}$, w_i, v_i, c_i, B_d, B_r
	$h(x_i)$	$\mathbf{f}_i$, x_i

3.J.2. DATA COLLECTION & TRAINING TIMES

For KIP, CNP, DRP, we sample 1,000 instances according to the procedures specified in Tang et al. [195], Dragotto et al. [63], and Ghatkar et al. [84], respectively. For each instance, we sample 100 upper-level decisions, i.e., 100,000 samples in total. Additionally, for KIP, CNP, DRP, the lower-level problems are solved with 30 CPUs in parallel. For training, we train for 1,000 epochs. However, if the validation mean absolute error does not improve in 200 iterations, we terminate early. Data collection and training times are reported in Table 3.12.

For DNDP, we use the Sioux Falls transportation network provided by [171] along with the author's 60 test instances. All instances use the same base network with different sets of candidate links to add and different budgets. There are 30 candidate links in total, and each test instance involves a subset of 10 or 20 of these links. To construct a training set, we sample 1000 leader decisions by first uniformly sampling an integer between 1 and 20, then uniformly sampling that many candidate links out of the set of 30 options; samples with total cost exceeding 50% of the total cost of all 30 edges are rejected as they are likely to exceed realistic budgets.

3.J.3. PREDICTION ERROR

For KIP, CNP, and DRP, we provide the Mean Absolute Error (MAE), as well as the Mean Absolute Label (MAL) as a reference to access the prediction quality for the validation data in Table 3.13. The table shows that models achieve a MAE of at most $\sim 1e^{-6}$ with a MAL ranging from 0.006 to 200 for all KIP, CNP, and DRP instances. For DNDP, both neural network and gradient-boosted tree models achieve Mean Absolute Percentage Error (MAPE) $\sim 5\%$.

Table 3.12: Data collection and training times for all problems. Note that as KIP is an interdiction problem, the same trained model can be used for the upper- and lower-level approximation, so we simply leave the upper-level as - for this problem. All times in seconds.

Problem	Data Collection	Training Time	
		Lower	Upper
KIP ($n = 18$)	142.08	2576.43	-
KIP ($n = 20$)	172.65	4714.88	-
KIP ($n = 22$)	141.61	2346.20	-
KIP ($n = 25$)	170.30	4007.75	-
KIP ($n = 28$)	142.34	2684.80	-
KIP ($n = 30$)	168.91	1835.27	-
KIP ($n = 100$)	164.16	3467.26	-
CNP ($\|V\| = 10$)	1,397.58	1839.60	4670.87
CNP ($\|V\| = 25$)	1,522.32	2072.60	4841.31
CNP ($\|V\| = 50$)	1,823.16	2103.50	2963.64
CNP ($\|V\| = 100$)	1,872.07	1944.08	2931.43
CNP ($\|V\| = 300$)	3,662.89	3800.02	3598.04
CNP ($\|V\| = 500$)	4,742.06	2263.68	6214.35
DRP	1939.24	1768.82	1784.15
DNDP	1033.15	1.96	3.19

Table 3.13: Prediction errors for all problems. Note that as KIP is an interdiction problem, the same trained model can be used for the upper- and lower-level approximation, so we simply leave the lower-level as - for this problem.

Problem	Upper-Level Approximation		Lower-Level Approximation	
	MAE	MAL	MAE	MAL
KIP ($n = 18$)	$5.05e^{-09}$	2.0992	-	-
KIP ($n = 20$)	$5.98e^{-09}$	2.5369	-	-
KIP ($n = 22$)	$4.00e^{-06}$	2.6933	-	-
KIP ($n = 25$)	$3.46e^{-10}$	3.0036	-	-
KIP ($n = 28$)	$1.48e^{-08}$	3.7327	-	-
KIP ($n = 30$)	$2.85e^{-08}$	3.7445	-	-
KIP ($n = 100$)	$1.21e^{-08}$	13.104	-	-
CNP ($\|V\| = 10$)	$1.35e^{-08}$	5.4606	$6.74e^{-06}$	1.5272
CNP ($\|V\| = 25$)	$1.22e^{-06}$	12.3452	$1.09e^{-08}$	3.7224
CNP ($\|V\| = 50$)	$1.23e^{-07}$	23.9687	$4.77e^{-09}$	7.7536
CNP ($\|V\| = 100$)	$4.33e^{-08}$	46.5468	$3.97e^{-06}$	14.7491
CNP ($\|V\| = 300$)	$1.83e^{-08}$	135.8972	$2.54e^{-07}$	44.577
CNP ($\|V\| = 500$)	$8.54e^{-08}$	222.3805	$9.51e^{-08}$	76.212
DRP	$2.51e^{-07}$	0.0062	$7.82e^{-08}$	0.0703
DNDP	0.0292	0.4297	0.02504	0.4990

4

NEURAL TWO-STAGE ROBUST INTEGER OPTIMIZATION

Robust optimization provides a mathematical framework for modeling and solving decision-making problems under worst-case uncertainty. This work addresses two-stage robust optimization (2RO) problems (also called adjustable robust optimization), wherein first-stage and second-stage decisions are made before and after uncertainty is realized, respectively. This results in a nested min-max-min optimization problem which is extremely challenging computationally, especially when the decisions are discrete.

Proposed data-driven algorithm: *We propose* NEUR2RO*, an efficient machine-learning-driven instantiation of column-and-constraint generation (CCG), a classical iterative algorithm for 2RO. Specifically, we learn to estimate the value function of the second-stage problem via a novel neural-network architecture that is easy to optimize over by design. We use the value function approximations, explained in Chapter 2 to embed our neural network into the problem formulations of CCG.*

Data generation scheme: *For obtaining data, we solve many instances of the most inner problem (the second-stage problem), by first sampling decisions of the first stage and uncertain scenarios, which are the inputs of this problem. This way, we only have to solve single-level problems to gather training data.*

Experimental results: NEUR2RO *produces high-quality solutions quickly, outperforming state-of-the-art methods on two-stage knapsack and capital budgeting problems. Compared to existing methods, which often run for hours,* NEUR2RO *finds similar quality solutions in a few seconds or minutes.*

This chapter is based on Dumouchelle et al. [66] published in **The Twelfth International Conference on Learning Representations (ICLR, 2024)** and the journal extension Dumouchelle et al. [64] available as a preprint. In collaboration with Justin Dumouchelle, Jannis Kurtz, and Elias B. Khalil. The code is available at `https://github.com/khalil-research/Neur2RO`.

4.1. INTRODUCTION

A wide range of real-world optimization problems in logistics, finance, and healthcare, among others, can be modeled by discrete optimization models [167]. While such mixed-integer (linear) programs (MILPs) can still be challenging to solve, the problem size that can be tackled with modern solvers has increased significantly thanks to algorithmic developments [2, 219]. In recent years, the incorporation of machine learning (ML) models into established algorithmic frameworks has received increasing attention [23, 229].

While most of ML for discrete optimization has focused on deterministic problems, in many cases, decision-makers face uncertainty in the problem parameters, e.g., due to forecasting or measurement errors in quantities of interest such as customer demand in inventory management. Besides the stochastic optimization approach, for which learning-based heuristics have been proposed recently [67], another popular approach to incorporate uncertainty into optimization models is *robust optimization*, where the goal is to find solutions that are optimal considering the worst realization of the uncertain parameters in a pre-defined uncertainty set [21]. This more conservative approach has been extended to two-stage robust optimization (2RO) where some of the decisions can be made on the fly after the uncertain parameters are realized [22]; see Yanıkoğlu et al. [222] for a survey.

Example (Capital budgeting). As a classical example of a two-stage robust problem, consider the capital budgeting problem as defined in Subramanyam et al. [189] where a company decides to invest in a subset of n projects. Each project i has an uncertain cost $c_i(\boldsymbol{\xi})$ and an uncertain profit $r_i(\boldsymbol{\xi})$ that both depend on the nominal cost and profit, respectively, and some risk factor $\boldsymbol{\xi}$ that dictates the difference from the nominal values to the actual ones. This risk factor, which we call an uncertain scenario, is contained in a given uncertainty set Ξ. The company can invest in a project either before or after observing the risk factor $\boldsymbol{\xi}$, up to a budget B. In the latter case, the company generates only a fraction η of the profit, which reflects a penalty of postponement. The objective of the capital budgeting problem is to maximize the total revenue subject to the budget constraint. This problem can be formulated as:

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} \max_{\mathbf{y}\in\mathcal{Y}} \quad \mathbf{r}(\boldsymbol{\xi})^\top(\mathbf{x}+\eta\mathbf{y}) \tag{4.1a}$$

$$\text{s.t.} \quad \mathbf{x}+\mathbf{y} \leq \mathbf{1} \tag{4.1b}$$

$$\mathbf{c}(\boldsymbol{\xi})^\top(\mathbf{x}+\mathbf{y}) \leq B. \tag{4.1c}$$

Here, $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$ and x_i and y_i are the binary variables that indicate whether the company invests in the i-th project in the first- or second-stage, respectively. Constraint (4.1b) ensures that the company can invest in each project only once and constraint (4.1c) ensures that the total cost does not exceed the budget.

2RO with integer decisions is much harder to solve than deterministic MILPs, especially when the uncertain parameters appear in the constraints and the second-stage decisions are discrete. Even evaluating the objective value of a solution in this case is algorithmically challenging [232]. In Subramanyam et al. [189], none of the generated

capital budgeting instances could be solved even approximately in a two-hour time limit for $n = 25$, terminating with an optimality gap of around 6%. In contrast to deterministic optimization problems, there is only limited literature on using ML methods to improve robust optimization [85, 111].

CONTRIBUTIONS

We propose Neural Two-stage Robust Optimization (NEUR2RO), an ML framework that can quickly compute high-quality solutions for 2RO. Our contributions are as follows:

- **ML in a novel optimization setting:** 2RO (also known as *adjustable RO*) has been receiving increased interest from the operations research community [222] and our work is one of the first to leverage ML in this setting.
- **ML at the service of a classical optimization algorithm:** to deal with the highly constrained nature of real-world optimization problems and rather than attempting to predict solutions directly, we "neuralize" a well-established 2RO algorithm, a strategy that combines the best of both worlds: correctness of an established algorithm with the predictive capabilities of an accurate neural network.
- **A compact, generalizable neural architecture** that is MILP-representable and estimates the thorny component of a 2RO problem, namely the value of the second-stage problem. The network is invariant to problem size and parameters, allowing, for example, the use of the same architecture for capital budgeting instances with a different number of projects and budget parameters.
- **Competitive experimental results** on capital budgeting and a two-stage robust knapsack problem, both benchmarks in the 2RO literature. NEUR2RO finds solutions that are of similar quality to or better than the state of the art. Large instances benefit the most from our method, with 100× reduction and 10 to 100× reductions in running time for knapsack and capital budgeting, respectively.

4.2. BACKGROUND

4.2.1. TWO-STAGE ROBUST OPTIMIZATION

2RO problems involve two types of decisions. The first set of decisions, $\mathbf{x}$, are referred to as *here-and-now* decisions and are made before the uncertainty is realized. The second set of decisions, $\mathbf{y}$, are referred to as the *wait-and-see* decisions and can be made on the fly after the uncertainty is realized. The uncertain parameters $\boldsymbol{\xi}$ are assumed to be contained in a convex and bounded uncertainty set $\Xi \subset \mathbb{R}^q$. The 2RO problem aims at finding a first-stage solution $\mathbf{x}$ which minimizes the worst-case objective value over all scenarios $\boldsymbol{\xi} \in \Xi$, where for each scenario the best possible second-stage decision $\mathbf{y}(\boldsymbol{\xi})$ is implemented. Mathematically, a 2RO problem is given by

$$\min_{\mathbf{x}\in\mathscr{X}} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\mathscr{Y}} \quad \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x} + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y} \tag{4.2a}$$

$$\text{s.t.} \quad T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi}), \tag{4.2b}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^m$ are feasible sets for the first and second stage decisions, respectively. In this work, we consider the challenging case of integer sets $\mathcal{X}$ and $\mathcal{Y}$. All parameters of the problem, namely $\mathbf{c}(\boldsymbol{\xi}) \in \mathbb{R}^n, \mathbf{d}(\boldsymbol{\xi}) \in \mathbb{R}^m, W(\boldsymbol{\xi}) \in \mathbb{R}^{r \times m}, T(\boldsymbol{\xi}) \in \mathbb{R}^{r \times n}$, and $\mathbf{h}(\boldsymbol{\xi}) \in \mathbb{R}^r$ depend on the scenario $\boldsymbol{\xi}$. We make the following assumption which is satisfied for the capital budgeting problem (and implicitly knapsack, which does not involve constraint uncertainty).

Assumption. For every $\mathbf{x} \in \mathcal{X}$, we have a method that calculates a scenario $\boldsymbol{\xi} \in \Xi$ for which the second-stage constraints $T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi})$ over $\mathbf{y} \in \mathcal{Y}$ are infeasible or verifies that no such scenario exists.

In the extended version of this work [64], this assumption is dismissed. Additional formulations are required to handle the extra complexity of the problem.

Both single- and multi-stage robust mixed integer problems are NP-hard even for deterministic problems that can be solved in polynomial time [46]. Compared to single-stage problems, which are often computationally tractable as they can be solved using reformulations [21] or constraint generation [154], two-stage problems are much harder to solve. When dealing with integer first-stage and continuous recourse, CCG is one of the key approaches [197, 227]. However, many problems, such as the ones we study here, deal with (mixed-)integer second-stage decisions. While an extension of CCG has been proposed that is able to handle mixed-integer recourse [232], this method is not well-established and often intractable and the results do not apply for pure integer second-stage problems.

In the case that the uncertainty only appears in the objective function, the 2RO can be solved by oracle-based branch-and-bound methods [112], branch-and-price [6], or iterative cut generation using Fenchel cuts [62]. For special problem structures and binary uncertainty sets, a Lagrangian relaxation can be used to transform 2RO problems with constraint uncertainty into 2RO with objective uncertainty which can then be solved by the aforementioned methods [130, 188].

4.2.2. COLUMN-AND-CONSTRAINT GENERATION

The main idea of CCG is to iterate between a *main problem* (MP) and an *adversarial problem* (AP). The MP is a relaxation of the original problem that only considers a finite subset of the uncertainty set $\Xi' \subset \Xi$. The latter problem can be modeled as an MILP by introducing copies of the second-stage decision variables $\mathbf{y}$ for each of the scenarios in Ξ'. After calculating an optimal solution of the MP, the AP finds new scenarios in the uncertainty set that cut off the current solution in the MP. When no such scenario can be found, the optimality of the current MP solution is guaranteed. For mathematical formulations of the two problems and a more detailed description of the CCG procedure, see Figure 4.1 and Appendix 4.B.1.

CCG often fails to calculate an optimal solution in reasonable time since both the MP and the AP are very hard to solve. In each iteration, the size of the MP increases leading to it being difficult to solve to optimality even with commercial MILP solvers such as Gurobi [92]. Furthermore, solving the AP is extremely challenging for integer second-stage variables. In Zhao and Zeng [232], the authors present a column-and-constraint

algorithm that solves the AP if the second stage is a mixed-integer problem; this leads to a CCG for the AP inside the main CCG, a most intractable combination. Additionally, the method of Zhao and Zeng [232] does not apply to purely integer second-stage decisions such as the problems we consider here.

4.3. RELATED WORK

Robust optimization. Besides the exact solution methods mentioned in Section 4.2.1, several heuristic methods have been developed to derive near-optimal solutions for mixed-integer 2RO problems. Methods that solve 2RO heuristically are K-adaptability [28, 95, 189], decision rules [31, 32], and iteratively splitting the uncertainty set [170]. Machine-learning techniques have been developed to speed up solution algorithms for the K-adaptability problem in Julien et al. [111]. In Goerigk and Kurtz [85] a decision tree classifier is trained to predict good start scenarios for the CCG. While being heuristic solvers, all of the above methods are still computationally highly demanding. In this work, the K-adaptability branch-and-bound algorithm by Subramanyam et al. [189] is used as a baseline since it is one of the only methods that can calculate high-quality solutions for reasonable problem sizes. For an elaborate overview of the latter algorithm, see Appendix 4.B.2 or Chapter 5 in which K-adaptability is extensively discussed.

Besides improving algorithmic performance, ML methods have been used to construct uncertainty sets based on historical data. In Goerigk and Kurtz [86] one-class neural networks are used to construct highly complex and non-convex uncertainty sets. Results from statistical learning theory are used to derive guarantees for ML-designed uncertainty sets in Tulabandhula and Rudin [198]. Other approaches use principal component analysis and kernel smoothing [160], support vector clustering [178, 179, 180], statistical hypothesis testing [33], or Dirichlet process mixture models [47, 159]. In Wang et al. [209] uncertainty sets providing a certain probabilistic guarantee are derived by solving a CVaR-constrained bilevel problem by an augmented Lagrangian method. While interesting and related, we here assume the uncertainty set is known.

MILP representations of neural networks. One key aspect of NEUR2RO is representing neural networks as constraints and variables in MILPs, which was first explored in [54, 75, 177, 196]. These representations have motivated active research to improve the MILP solving efficiency of optimizing over-trained models [5, 90, 210], as well as several software contributions [24, 51], in addition to Gurobi, a commercial MILP solver, providing an open-source library. The use of embedding trained predictive models to derive approximate MILPs has been explored for non-linear constraints or intractable constraints [90, 113, 121, 153, 174], stochastic programming [67, 124], and bilevel optimization [65]. As NEUR2RO is based on an approximation for intractable 2RO problems with embedded neural networks, the latter area of research is the most closely related. However, the min-max-min optimization in 2RO renders previous learning-based MILP approximations unsuitable.

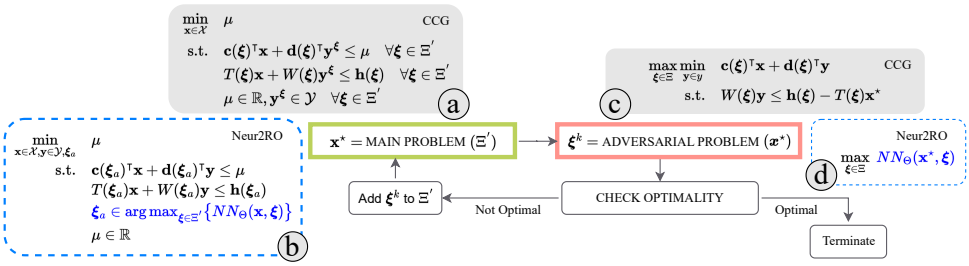


Figure 4.1: Column-and-constraint generation: in each iteration, a *main problem* (box (a)) is solved to find a good first-stage solution $\mathbf{x}^\star$ for the set of scenarios that have been identified thus far (initially, none). Then, an *adversarial problem* (box (c)) is solved to obtain a new scenario for which the solution $\mathbf{x}^\star$ is not feasible anymore in MP. If no such scenario exists, then $\mathbf{x}^\star$ is *optimal* and CCG terminates. Otherwise, the adversarial scenario is added to the set of worst-case scenarios and we iterate to MP. For each of the MP and AP, we show two versions: classical (CCG, boxes (a) and (c)) and learning-augmented (NEUR2RO, dashed boxes (b) and (d)).

4.4. METHODOLOGY

At a high level, our approach aims to train a neural network that predicts the optimal second-stage objective value function and then integrates this model within a CCG framework to obtain first-stage decisions. We rely on a training dataset of historical instances that can be used or generated, as is typically assumed in ML-for-optimization work.

4.4.1. LEARNING MODEL

As mentioned before, CCG is computationally very expensive. Both the MP and AP contribute to its intractability (see Figure 4.1 boxes (a) and (c) for descriptions). In the MP, for each added scenario, a new second-stage decision $\mathbf{y}$ is introduced. When a large number of scenarios are required to obtain a robust solution, the number of variables grows rapidly. Moreover, the AP is especially hard when the second-stage decisions are integer, which is the case we consider. In our learning-augmented approach, we replace the intractable elements of the CCG with MILP representations of a trained NN which is computationally much easier to handle (Figure 4.1).

We train a neural network that can accurately predict the optimal value of the second-stage problem for a given input of a first-stage decision, an uncertainty realization, and the problem's specification, $\mathscr{P}$. The problem specification refers to the coefficients and size of the optimization problem, e.g., the nominal values of the profit and costs in the capital budgeting problem. More formally we train a neural network $\mathrm{NN}_\Theta(\cdot)$, to approximate the optimal value of the integer problem

$$\mathrm{NN}_\Theta(\mathbf{x}, \boldsymbol{\xi}, \mathscr{P}) \approx \min_{y \in \mathscr{Y}} \{\mathbf{c}_\mathscr{P}(\boldsymbol{\xi})^\top \mathbf{x} + \mathbf{d}_\mathscr{P}(\boldsymbol{\xi})^\top \mathbf{y} : W_\mathscr{P}(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}_\mathscr{P}(\boldsymbol{\xi}) - T_\mathscr{P}(\boldsymbol{\xi})\mathbf{x}\}, \tag{4.3}$$

where Θ are the weights of the neural network. For ease of notation, we hereafter omit $\mathscr{P}$ in the formulation. For more details on the architecture of $\mathrm{NN}_\Theta(\cdot)$, see Section 4.4.3. Alternatively, as $\mathbf{c}_\mathscr{P}(\boldsymbol{\xi})^\top \mathbf{x}$ is a scalar product of the input vectors, we instead could only predict the second-stage objective, i.e., $\mathbf{d}_\mathscr{P}(\boldsymbol{\xi})^\top \mathbf{y}$, subject to the same constraints. How-

ever, as demonstrated in Appendix 4.H.3, predicting the sum of first- and second-stage objectives achieves higher-quality solutions.

4.4.2. ML-BASED COLUMN-AND-CONSTRAINT GENERATION

Having defined the learning task, we now describe the ML-based approximate CCG algorithm. For an overview of this method, see Figure 4.1.

Main problem. Given a finite subset of scenarios $\Xi' \subset \Xi$, we reformulate the MP using an argmax operator which selects a scenario that achieves the worst objective function value when replacing the second-stage objective value by the neural network formulation.

$$\min_{\mathbf{x}\in\mathscr{X},\mathbf{y}\in\mathscr{Y},\boldsymbol{\xi}_a\in\Xi} \quad \mathbf{c}(\boldsymbol{\xi}_a)^\top\mathbf{x} + \mathbf{d}(\boldsymbol{\xi}_a)^\top\mathbf{y} \tag{4.4a}$$

$$\text{s.t.} \quad W(\boldsymbol{\xi}_a)\mathbf{y} + T(\boldsymbol{\xi}_a)\mathbf{x} \leq \mathbf{h}(\boldsymbol{\xi}_a), \tag{4.4b}$$

$$\boldsymbol{\xi}_a \in \operatorname{argmax}_{\boldsymbol{\xi}\in\Xi'}\{\mathrm{NN}_\Theta(\mathbf{x},\boldsymbol{\xi})\}. \tag{4.4c}$$

Modeling the argmax can be done by adding additional linear constraints and binary variables, which we explicitly show in Appendix 4.C. The MP results in an MILP formulation.

This formulation is indeed not the only option; for example, one could instead consider another formulation, called max, which provides a more intuitive formulation and does not require modeling second-stage variables; details are provided in Appendix 4.H.1. However, (4.4) has one key property that motivates its efficacy. From the machine learning perspective, rather than requiring a neural network to be an accurate estimator of the objective for each scenario, we only require that the neural network be able to identify the maximal scenario. Prediction inaccuracy is then compensated for in (4.4a) by exactly modeling the second-stage cost. As a result, when solving the MP, the true optimal first-stage decision for the selected scenario will be the minimizer, rather than a potentially suboptimal first-stage decision based on any inaccuracy of the learning model. Appendix 4.H.1 presents an ablation comparing both the solution quality of the argmax and max formulations on the knapsack problem, and establishes that the argmax formulation indeed computes higher quality solutions across every instance.

Adversarial problem. In the AP of NEUR2RO, we replace the inner optimization problem over $\mathbf{y}$ by its prediction $\mathrm{NN}_\Theta(\mathbf{x}^\star,\boldsymbol{\xi})$, where $\mathbf{x}^\star$ is given (see Figure 4.1 box (d)). When we deal with constraint uncertainty, we first check if there exists a scenario $\boldsymbol{\xi}\in\Xi$, such that no feasible $\mathbf{y}\in\mathscr{Y}$ exists for the constraints

$$W(\boldsymbol{\xi})\mathbf{y} + T(\boldsymbol{\xi})\mathbf{x}^\star \leq \mathbf{h}(\boldsymbol{\xi}),$$

which we can do by the assumption from Section 4.2.1. If such a scenario exists, we add it to Ξ' and continue with solving the MP again. Note that in this case $\mathbf{x}^\star$ is not feasible for MP in the next iteration. If no such scenario could be found, we calculate an optimal solution $\boldsymbol{\xi}^\star$ of the AP in box (d) of Figure 4.1 which can be done by using the MILP representation for NN. Note that if Ξ is a polyhedron or an ellipsoid, then this

problem results in a mixed-integer linear or quadratic problem, respectively, which can be solved by state-of-the-art solvers such as Gurobi. We compare the optimal value of the latter problem with the objective values of all scenarios that were considered in the MP before. If the following holds

$$\max_{\boldsymbol{\xi}\in\Xi} \mathrm{NN}_\Theta(\mathbf{x}^\star, \boldsymbol{\xi}) \geq \max_{\boldsymbol{\xi}\in\Xi'} \mathrm{NN}_\Theta(\mathbf{x}^\star, \boldsymbol{\xi}) + \varepsilon \tag{4.5}$$

for a pre-defined accuracy parameter $\varepsilon > 0$, then we add $\boldsymbol{\xi}^\star$ to Ξ' and continue with the MP. Otherwise, we stop the algorithm. Finally, note that we can calculate both types of scenarios in each iteration and add them both to Ξ' before we iterate to the MP. As the adversarial problem requires finding the worst-case uncertainty over a neural network's input, heuristic approaches may significantly improve solving time with minimal degradation in solution quality. Appendix 4.H.2 presents an ablation demonstrating significantly lower solution time at a minimal cost of solution quality for sampling- and relaxation-based heuristics.

Convergence. Since our algorithm does not apply the standard CCG steps, the convergence guarantee from the classical algorithm does not hold. However, we prove in Appendix 4.F that it holds if only finitely many first-stage solutions exist, which is the case if all first-stage variables are integer and $\mathcal{X}$ is bounded; this indeed holds for the knapsack and capital budgeting problems.

Theorem 4.1. *If $\mathcal{X}$ is finite, the ML-based CCG terminates after a finite number of iterations.*

4.4.3. ARCHITECTURE

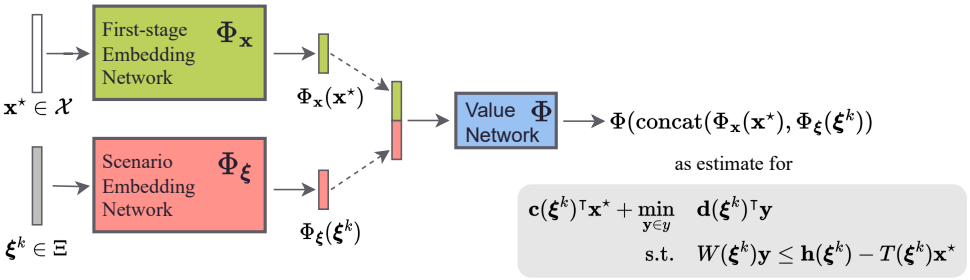


Figure 4.2: neural-network architecture for ML-based CCG. The current first-stage solution, $\mathbf{x}^\star$, is embedded once using the network $\Phi_{\mathbf{x}}(\cdot)$. A scenario $\boldsymbol{\xi}^k$ is embedded using the network $\Phi_{\boldsymbol{\xi}}(\cdot)$. To estimate the value of the second-stage optimization problem corresponding to a particular pair $(\mathbf{x}^\star, \boldsymbol{\xi}^k)$, the two embedding vectors are concatenated into one (dashed arrows) and then passed into the final *Value Network*.

For the ML-based CCG, one requirement is the optimization, in each iteration, over several trained neural networks in the MP (one for each scenario in (4.4c)) and a single trained neural network in the AP. Generally, increasing the size of the networks will lead to more challenging and potentially intractable optimization problems. For that reason,

developing an architecture that can be efficiently optimized over is a crucial aspect of an efficient ML-based CCG algorithm.

To achieve efficient optimization, we embed the first-stage decisions, $\mathbf{x}$, and a scenario, $\boldsymbol{\xi}$, into low-dimensional embeddings using networks $\Phi_{\mathbf{x}}$ and $\Phi_{\boldsymbol{\xi}}$, respectively. These embeddings are concatenated and passed through a final small neural network (the *Value Network*) Φ that predicts the objective of the optimal second-stage response; see Figure 4.2 for a pictorial representation.

Main problem optimization. When representing the trained models in the MILP, we only have to represent the embedding network $\Phi_{\mathbf{x}}$ and the small value network Φ, which can be done by classical MILP representations of ReLU NNs [75]. Since the scenario parameters are not variables here, the scenario embeddings $\Phi_{\boldsymbol{\xi}}(\boldsymbol{\xi}^k)$ can be precomputed via a forward pass for each scenario, i.e., no MILP representation is needed for $\Phi_{\boldsymbol{\xi}}$. If Φ is a small neural network, then representing a large number of copies of the network (one per scenario) remains amenable to efficient optimization.

Adversarial problem optimization. For the AP, we only require representing $\Phi_{\boldsymbol{\xi}}$ and Φ as the embedding of $\mathbf{x}$ can be precomputed with a forward pass.

Generalizing across instances. For simplicity of notation and presentation, the previous sections have omitted the generalization across instances, which is a key aspect of the generality of our methodology. To generalize across instances, invariance to the number, ordering, constraint coefficients, and objective coefficients of decision variables is required. To handle this, NEUR2RO leverages set-based neural networks [226] for $\Phi_{\mathbf{x}}$ and $\Phi_{\boldsymbol{\xi}}$. Specifically, embeddings are computed for each single first-stage and scenario variable (x_i and ξ_i) using their values, constraints, and objective coefficients, via a network with shared parameters. These embeddings are then aggregated and passed through an additional feed-forward neural network to derive the first-stage and scenario embeddings. For a detailed diagram of this architecture, see Appendix 4.D.

4.5. EXPERIMENTAL SETUP

Computational setup. All experiments were run on a computing cluster with an Intel Xeon CPU E5-2683 and Nvidia Tesla P100 GPU with 64GB of RAM (for training). Pytorch 1.12.1 [164] was used for all learning models. Gurobi 10.0.2 [92] was used as the MILP solver and gurobi-machinelearning 1.3.0 was used to embed the neural networks into MILPs. For evaluation, all solving was limited to 3 hours. For NEUR2RO, we terminate a solve of the MP or AP early if no improvement in solution is observed in 180 seconds. Our code and data are available at `https://github.com/khalil-research/Neur2RO`.

2RO problems. We benchmark NEUR2RO on two 2RO problems from the literature, namely a two-stage knapsack problem and the capital budgeting problem. In both cases, our instances are as large or larger than considered in the literature. The two-stage knapsack problem is in the first stage a classical knapsack problem. The second stage has decisions for responding to an uncertain profit degradation. The capital budgeting prob-

lem is described in the introduction. For a detailed description of these problems, see Appendix 4.A. Below we briefly detail each problem.

– **Knapsack.** For the knapsack problem, we use the same instances as in Arslan and Detienne [6], which have been inspired by Ben-Tal et al. [21]. They have categorized their instances into four groups: uncorrelated (UN), weakly correlated (WC), almost strongly correlated (ASC), and strongly correlated (SC), which affects the correlation of the nominal profits of items with their cost and, in turn, the difficulty of the problem. More correlated instances are much harder to solve. We consider instances of sizes $n \in \{20, 30, 40, 50, 60, 70, 80\}$.

– **Capital budgeting.** These problem instances are generated similar to Subramanyam et al. [189].While uncertain parameters appear in the constraints (see (4.1c)), we can easily verify the assumption given in Section 4.2.1 as follows: for every $\mathbf{x}$ we check if $\max_{\boldsymbol{\xi} \in \Xi} \mathbf{c}(\boldsymbol{\xi}) \leq B$, where the maximum can be easily calculated since it is a linear problem over Ξ. If the latter inequality is true, the second-stage problem is feasible since we can choose $\mathbf{y} = \mathbf{0}$. On the other hand, if the inequality is violated, then no feasible second-stage solution exists since $\mathbf{y} \geq \mathbf{0}$, and hence the maximizing scenario will be added to MP. We consider instances of sizes $n \in \{10, 20, 30, 40, 50\}$.

Baselines. For knapsack, we compare to the branch-and-price (BP) algorithm from Arslan and Detienne [6], the state-of-the-art for 2RO problems with objective uncertainty. We use the instances and the objective values and solution times reported in their paper[1]. For capital budgeting, we use the K-adaptability approach of Subramanyam et al. [189] (more details in Appendix 4.B.2) with $k = 2, 5, 10$ as a baseline; CCG is not tractable for this problem due to its integer recourse.

Evaluation. After NEUR2RO finds the first-stage decision $\mathbf{x}^\star$, we obtain the corresponding objective value by solving 4.2 for a fixed $\mathbf{x}$. For knapsack, this can be efficiently solved by constraint generation of $\mathbf{y}$. For capital budgeting on the other hand, due to constraint uncertainty, we cannot use the constraint generation approach. Instead we sample scenarios from Ξ, and solve 4.2 with fixed $\mathbf{x}$ and $\boldsymbol{\xi}$. See Appendix 4.E for a more detailed explanation of these methods.

As previously mentioned, we use K-adaptability as the baseline for the capital budgeting problem. This method solves 4.2 only approximately with the approximation quality getting better with larger K at an increase in solution times. We take the first-stage solution found by K-adaptability and compare it with the one of NEUR2RO using the scenario sampling approach just described.

For the evaluation of the objective values, two metrics are considered. We use relative error (RE), i.e., the gap to the best-known solution, to compare solution quality. Specifically, if $obj^\star$ is the value of the best solution found by NEUR2RO or a baseline for a particular instance, then for algorithm A with objective obj_A, the RE is given by $100 \cdot \frac{|obj^\star - obj_A|}{|obj^\star|}$. To compare efficiency, we compare the average solution time.

[1] Available at https://github.com/borisdetienne/RobustDecomposition.

Data collection & training. For data collection, we sample sets of instances, first-stage decisions, and scenarios to obtain features. The features are provided in Appendix 4.I. Labels are then computed by solving the corresponding innermost optimization problem, i.e., a tractable deterministic MILP as both $\mathbf{x}$ and $\boldsymbol{\xi}$ are fixed. Additionally, this process is highly parallelizable since each optimization problem is independent. For knapsack and capital budgeting, we randomly sample 500 instances, 10 first-stage decisions per instance, and 50 scenarios per first-stage decision, resulting in 250,000 data points. The dataset is split into 200,000 and 50,000 samples for training and validation, respectively.

We train one size-independent model for each problem for 500 epochs. The data collection times, training times, and total times (in seconds) are 2,162, 3,789, and 5,951 for knapsack and 3,212, 2,195, and 5,407 for capital budgeting. We note that both times are relatively insignificant given that we provide approximately twice the time (3 hours) to solve a single instance during evaluation. Furthermore, the model for NEUR2RO generalizes across instance parameters and sizes. Appendix 4.I provides full detail on model hyperparameters and training.

4.6. EXPERIMENTAL RESULTS

For knapsack, we test our method and the baseline on 18 instances per correlation type and instance size (504 instances). For capital budgeting, we test on 50 instances per instance size (250 instances). We note that training and validation data are generated using the procedures specified in the corresponding papers, and different instances are used for testing. For optimization of NEUR2RO, this section presents results for solving the MIP and MILP formulations for the MP and AP, with the argmax formulation outlined in Section 4.4 for the MP. Tables 4.1-4.2 report the median RE and solving times. In addition, for more detailed distributional information, boxplots and more detailed metrics, are provided in Appendix 4.G.

Table 4.1: Median RE and solving times for knapsack instances. For each row, the median RE and average solving time are computed over 18 instances. All times in seconds. The smallest (best) values in each row/metric are in bold.

Correlation Type	# items	Median RE		Times	
		NEUR2RO	BP	NEUR2RO	BP
Uncorrelated	20	1.417	**0.000**	4	**0**
	30	1.188	**0.000**	6	**1**
	40	1.614	**0.000**	9	**3**
	50	1.814	**0.000**	**9**	12
	60	1.146	**0.000**	**14**	18
	70	1.408	**0.000**	**16**	46
	80	0.968	**0.000**	**11**	388
Weakly Correlated	20	1.582	**0.000**	**5**	29
	30	2.236	**0.000**	**11**	454
	40	1.595	**0.000**	**20**	6179
	50	1.757	**0.000**	**19**	8465
	60	0.695	**0.000**	**77**	9242
	70	0.165	**0.000**	**15**	10800
	80	**0.000**	0.341	**21**	10800
Almost Strongly Correlated	20	1.439	**0.000**	**5**	9
	30	0.782	**0.000**	**6**	2708
	40	0.497	**0.000**	**10**	4744
	50	0.019	**0.000**	**7**	8852
	60	**0.000**	0.016	**14**	10261
	70	**0.017**	0.031	**13**	10800
	80	**0.000**	0.265	**12**	10800
Strongly Correlated	20	1.604	**0.000**	**5**	9
	30	0.610	**0.000**	**7**	2473
	40	0.443	**0.000**	**11**	5665
	50	0.073	**0.000**	**9**	8240
	60	0.042	**0.010**	**11**	10800
	70	**0.020**	0.027	**16**	10800
	80	**0.000**	0.179	**13**	10800

Table 4.2: Combined results for capital budgeting instances. For each row, the median RE and average solving time are computed over 50 instances. All times in seconds. The smallest (best) values in each row/metric are in bold.

# items	Median RE				Times			
	NEUR2RO	$K=2$	$K=5$	$K=10$	NEUR2RO	$K=2$	$K=5$	$K=10$
10	1.105	1.140	**0.000**	**0.000**	59	**20**	9561	10800
20	**0.000**	0.196	0.112	0.064	**324**	8702	10800	10800
30	0.109	**0.020**	0.073	0.032	**602**	10801	10800	10800
40	**0.009**	0.074	0.011	0.019	**739**	10806	10801	10801
50	**0.001**	0.033	0.039	0.020	**1032**	10807	10804	10801

Knapsack. Table 4.1 demonstrates a clear improvement in scalability, with the solving time of NEUR2RO ranging between 4 and 77 seconds, while the solving time for BP scales directly with difficulty induced by the size and correlation type. For the more difficult instances, i.e., instances with a large number of items and (almost) strong correlation, NEUR2RO generally finds better quality solutions over 100 times faster than BP, which is a very strong result considering BP is the state-of-the-art for problems with objective uncertainty. Figures 4.6-4.7 of Appendix 4.G further demonstrate that the distribution of RE achieved by NEUR2RO, not just the median, is far more favorable than BP's on the most challenging instances. For easier instances, NEUR2RO is less competitive in terms of solution quality as BP converges to optimal solutions within the time limit. However, even for these instances, NEUR2RO achieves a median RE of 2.235% in the worst-case, often still 1-2 orders of magnitude faster than BP, with the exception of a few very easy instances.

Capital budgeting. NEUR2RO achieves the lowest median RE for 20, 40, and 50-item instances, i.e., the two largest and most challenging instance sets. The distribution of RE for 40 and 50-item instances provided in Figure 4.8 of Appendix 4.G is indeed consistent with the median result, as it illustrates that NEUR2RO finds quality solutions on the majority of the instances. In terms of solving time, NEUR2RO generally converges much faster than K-adaptability, resulting in a very favorable trade-off: we can find better or equally good solutions 10 to 100 times faster. Note that the relative errors are quite small in an absolute sense. For example, for 30-item instances, NEUR2RO has a median RE of 0.109 compared to the best baseline's 0.020; solutions that are within 0.109% of the best achievable may be acceptable in practice. Note that we have also measured the median RE for K-adaptability assuming a shorter time limit, namely the same amount of time as NEUR2RO on each instance. Taking the incumbent solution found by K-adaptability at that time point typically yields worse solutions than those reported in Table 4.2, see Appendix 4.H.4 for details. Compared to the knapsack, the solving time is generally much larger as the instance size increases. We speculate that this may relate to the uncertainty in the objective of the first-stage decision or the budget constraints that are not present in the knapsack problem.

In summary, for both benchmark problems, NEUR2RO achieves high-quality solutions. For relatively easy or small instances, state-of-the-art methods sometimes find slightly better solutions, often at a much higher computational cost. However, as the instances become more difficult, NEUR2RO demonstrates a clear improvement in overall solution quality and computing time.

4.7. CONCLUSION

With the uncertainty in real-world noisy data, the economy, the climate, and other avenues, there is an increasing need for efficient robust decision-making. We have shown how NEUR2RO uses MILP-representable feedforward neural networks to estimate the thorny component of a family of two-stage robust optimization instances, namely the value of the second-stage problem. The neural-network architecture delicately combines low-dimensional embeddings of a first-stage decision and a scenario to produce the second-stage value estimate. Using an off-the-shelf MILP solver, we then use the neural network in a classical iterative algorithm for 2RO. NEUR2RO to find competitive solutions compared to state-of-the-art methods on two challenging benchmark problems, knapsack and capital budgeting, at a substantial reduction in solution time. In an extension of this work [64], we do not impose any conditions on the constraints, and introduce additional formulations to deal with this.

APPENDIX OF CHAPTER 4

4.A. 2RO PROBLEMS

4.A.1. ROBUST TWO-STAGE KNAPSACK

We consider the two-stage knapsack problem as defined in Arslan and Detienne [6] with a set of n items. Each item i has a weight c_i and an uncertain profit $p_i(\boldsymbol{\xi}) = \bar{p}_i - \xi_i \hat{p}_i$, where $\bar{p}_i$ is the expected profit, $\hat{p}_i$ its maximum deviation and ξ_i the *uncertain* profit degradation factor, where the degradation happens after the first stage. In this problem we have a budgeted uncertainty set $\Xi = \{\boldsymbol{\xi} \in [0,1]^n : \sum_{i=1}^n \xi_i \leq \Gamma\}$. The first stage decision is to choose a subset of items to produce. Then in the second stage, there are three different responses to the profit degradation: (i) accept the degraded profit, (ii) repair the item by using an additional t_i units from the budget to recover the original profit $\bar{p}_i$, or (iii) outsource the item for a cost of f_i units, such that the item's profit results in $\bar{p}_i - f_i$. This gives the following problem formulation:

$$\begin{aligned}
\min_{\mathbf{x}\in\{0,1\}^n} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\{0,1\}^n, \mathbf{r}\in\{0,1\}^n} \quad & \sum_{i=1}^n (f_i - \bar{p}_i) x_i + (\hat{p}_i \xi_i - f_i) y_i - \hat{p}_i \xi_i r_i \\
\text{s.t.} \quad & \sum_{i=1}^n c_i y_i + t_i r_i \leq C \\
& r_i \leq y_i \leq x_i \qquad \forall i \in \{1,\dots,n\},
\end{aligned}$$

where x_i is the first-stage decision to produce item i. For the second-stage decisions, we have y_i and r_i: (i) $y_i = 1$ if item i is produced *without* repairing and $y_i = 0$ if the item is outsourced, and (ii) r_i is the decision for repairing item i.

4.A.2. CAPITAL BUDGETING

Consider the capital budgeting problem in Subramanyam et al. [189], where a company aims to invest in a subset of n projects. For each project, i, the uncertain cost, and profit are respectively defined as

$$c_i(\boldsymbol{\xi}) = \left(1 + \boldsymbol{\Phi}_i^\intercal \boldsymbol{\xi}/2\right)\bar{c}_i \quad \text{and} \quad r_i(\boldsymbol{\xi}) = \left(1 + \boldsymbol{\Psi}_i^\intercal \boldsymbol{\xi}/2\right)\bar{r}_i, \quad \forall i \in \{1,\dots,n\},$$

where $\bar{c}_i$ and $\bar{r}_i$ are the nominal cost and nominal profit of project i. $\boldsymbol{\Phi}_i^\intercal$ and $\boldsymbol{\Psi}_i^\intercal$ are the i-th row vectors of the sensitivity matrices $\boldsymbol{\Phi}, \boldsymbol{\Psi} \in \mathbb{R}^{n\times 4}$, with $\boldsymbol{\xi} \in \Xi = [-1,1]^4$. We use the problem formulation described in 4.1.

4.B. 2RO ALGORITHMS

In this section, we describe the column-and-constraint generation algorithm in more detail and the K-adaptability problem, briefly describing one of its solution methods.

4.B.1. COLUMN-AND-CONSTRAINT GENERATION

The CCG iterates between the *main problem* and the *adversarial problem* (AP).

The MP is given as

$$\min_{\mathbf{x}\in\mathscr{X}} \max_{\boldsymbol{\xi}\in\Xi'} \min_{y\in\mathscr{Y}} \quad \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x} + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y} \tag{4.6a}$$

$$\text{s.t.} \quad T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi}), \tag{4.6b}$$

where $\Xi' \subset \Xi$ is a finite subset of scenarios. Clearly, the MP provides a lower bound on the optimal value of (4.2). To solve the MP, for each scenario in Ξ' a copy of the second-stage variables is generated. Using a level-set transformation, the problem can be formulated as

$$\min_{\mathbf{x}\in\mathscr{X}} \quad \mu \tag{4.7a}$$

$$\text{s.t.} \quad \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x} + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y}^{\boldsymbol{\xi}} \leq \mu \quad \forall \boldsymbol{\xi} \in \Xi' \tag{4.7b}$$

$$T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y}^{\boldsymbol{\xi}} \leq \mathbf{h}(\boldsymbol{\xi}) \quad \forall \boldsymbol{\xi} \in \Xi' \tag{4.7c}$$

$$\mu \in \mathbb{R}, \mathbf{y}^{\boldsymbol{\xi}} \in \mathscr{Y} \quad \forall \boldsymbol{\xi} \in \Xi', \tag{4.7d}$$

which is a linear integer problem that state-of-the-art solvers, such as Gurobi, can solve. In each iteration of the CCG an optimal solution $(x^\star, \mu^\star)$ of (4.7) is calculated. Afterwards, the AP is solved, which is defined as

$$\max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\mathscr{Y}} \quad \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y} \tag{4.8a}$$

$$\text{s.t.} \quad W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi}) - T(\boldsymbol{\xi})\mathbf{x}^\star. \tag{4.8b}$$

Since the optimal value of the AP is the objective value of the current solution $\mathbf{x}^\star$, it provides an upper bound on the optimal value of (4.2). We define the optimal value to be equal to infinity if there exists a scenario $\boldsymbol{\xi} \in \Xi$ for which no feasible second-stage solution $\mathbf{y}$ exists. If the optimal value of the AP is larger than $\mu^\star$ then we add the optimal scenario $\xi^\star$ to Ξ' and start again from solving MP. Otherwise, we stop the algorithm since the upper bound is smaller or equal to the lower bound, and hence $\mathbf{x}^\star$ is an optimal solution. The whole procedure is presented in Algorithm 4.1.

Algorithm 4.1: Column-and-Constraint Generation

Input : A problem instance; Ξ, the uncertainty set.
Output : Optimal first-stage decisions

Set $ub = \infty$, $lb = -\infty$
$\Xi' = \{\xi_0\}$ for any $\boldsymbol{\xi}_0 \in \Xi$
while $ub - lb > 0$ **do**
  Calculate an optimal solution $\mathbf{x}^\star, \mu^\star$ of the main problem (4.7) and set $lb = \mu^\star$.
  Calculate an optimal solution $\boldsymbol{\xi}^\star$ (with optimal value $opt^\star$) of the adversarial problem (4.8) where $\mathbf{x} = \mathbf{x}^\star$.
  Set $\Xi' = \Xi' \cup \{\boldsymbol{\xi}^\star\}$ and $ub = \min\{ub, opt^\star\}$.
end
return $\mathbf{x}^\star$

CCG often fails to calculate an optimal solution in a reasonable time since both the MP and the AP are very hard to solve in the case of integer second-stage variables. In each iteration, the size of MP increases since we have to add new constraints and a copy of all integer second-stage decisions $\mathbf{y}$. This often leads to the situation that after even a small number of iterations, the MP cannot be solved to optimality anymore by classical integer optimization solvers as Gurobi.

Furthermore, solving the AP is extremely challenging for integer second-stage variables. Indeed, the problem can be formulated as a bilevel problem where the follower problem contains integer variables. In Zhao and Zeng [232] the authors present a column-and-constraint algorithm that solves the AP if the second-stage is a mixed-integer problem. One drawback is that this method is not applicable if the second-stage does not contain continuous variables, as is the case for many problems, e.g., the capital budgeting problem. Furthermore, the method involves solving a very large mixed-integer bilinear problem, which is computationally enormously challenging. The whole procedure must be executed in each iteration of the main CCG algorithm.

4.B.2. K-ADAPTABILITY

The K-adaptability approach was introduced in Bertsimas and Caramanis [28] and later studied for objective uncertainty and constraint uncertainty in [83, 95, 111, 128, 189]. The main idea of the approach is to calculate a set of K second-stage solutions already in the first-stage. Instead of choosing the best feasible second-stage solution for each scenario $\boldsymbol{\xi}$, we choose the best of the K calculated second-stage solutions. Since we restrict the number of second-stage reactions, this approach leads to feasible solutions of (4.2), which are not necessarily optimal. While for larger K the approximation guarantee gets provably better, the problem gets harder to solve at the same time. Furthermore, it was shown in Subramanyam et al. [189] that it may happen that K has to be chosen exponentially large to guarantee optimality for (4.2). The K-adaptability problem can be formulated as

$$\min_{\mathbf{x}\in\mathcal{X},\mathbf{y}^1,\dots,\mathbf{y}^k\in\mathcal{Y}} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\{\mathbf{y}^1,\dots,\mathbf{y}^k\}} \quad \mathbf{c}(\boldsymbol{\xi})^\top\mathbf{x}+\mathbf{d}(\boldsymbol{\xi})^\top\mathbf{y} \tag{4.9a}$$

$$\text{s.t.} \quad W(\boldsymbol{\xi})\mathbf{y}+T(\boldsymbol{\xi})\mathbf{x}\leq\mathbf{h}(\boldsymbol{\xi}). \tag{4.9b}$$

The K-adaptability problem is very challenging to solve, especially in the constraint uncertainty case. The best-known method for this case was introduced in Subramanyam et al. [189]. The authors perform a branch-and-bound algorithm over partitions of the uncertainty set. They consider K-partitions of finite scenarios sets, which are iteratively generated, and assign each of the second-stage solutions to one of the partitions. This approach was later improved by applying machine-learning methods to improve the branching decisions [111]. As an alternative approach in [170], an iterative uncertainty set splitting method is presented, which converges to the exact optimal value of the two-stage robust problem.

In case of objective uncertainty, the K-adaptablity problem is easier (but still hard) to solve [7, 83] and can be approximated if K is not too small; see Kurtz [128].

4.C. DETAILED FORMULATION

This section presents the detailed argmax formulation for (4.4). We assume that at this iteration in the MP, we have scenarios $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_k$ and that M and L are upper and lower bounds on the prediction of the network. The complete formulation is then given by

$$\min_{\mathbf{x}\in\mathscr{X},\mathbf{y}\in\mathscr{Y},\boldsymbol{\xi}_a\in\Xi,\mathbf{p},u,\mathbf{z}\in\{0,1\}^k} \quad \mathbf{c}(\boldsymbol{\xi}_a)^\intercal\mathbf{x}+\mathbf{d}(\boldsymbol{\xi}_a)^\intercal\mathbf{y} \tag{4.10a}$$

$$\text{s.t.} \quad W(\boldsymbol{\xi}_a)\mathbf{y}+T(\boldsymbol{\xi}_a)\mathbf{x}\leq\mathbf{h}(\boldsymbol{\xi}_a), \tag{4.10b}$$

$$p_i=\mathrm{NN}_\Theta(\mathbf{x},\boldsymbol{\xi}_i) \qquad \forall i\in\{1,\ldots,k\} \tag{4.10c}$$

$$u\geq p_i \qquad \forall i\in\{1,\ldots,k\} \tag{4.10d}$$

$$u\leq p_i+(M-L)(1-z_i) \qquad \forall i\in\{1,\ldots,k\} \tag{4.10e}$$

$$\sum_{i=1}^{k} z_i=1 \qquad \forall i\in\{1,\ldots,k\} \tag{4.10f}$$

$$\boldsymbol{\xi}_a=\sum_{i=1}^{k} z_i\cdot\boldsymbol{\xi}_i \tag{4.10g}$$

To model the argmax, we introduce k binary variables $\mathbf{z}$ and $k+1$ continuous variables $\mathbf{p}$ and u, which are used to model big-M that ensure $\mathbf{z}$ is 1 at the index of the maximizer and 0 everywhere else. $\boldsymbol{\xi}_a$ is then given by a linear combination of the scenarios multiplied with $\mathbf{z}$.

4.D. EXTENDED NN ARCHITECTURE

We show the extended neural-network architecture used in the experiments in Figure 4.3.

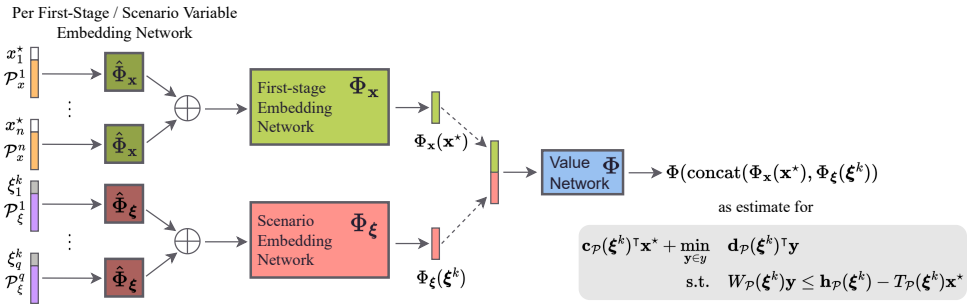


Figure 4.3: The extended neural-network architecture for ML-based CCG. Compared to the NN architecture shown in the main text (Figure 4.2), this model uses the set-based method to be able to generalize across instance sizes. Let $\mathbf{x}^\star\in\mathbb{R}^n$ and $\boldsymbol{\xi}\in\mathbb{R}^q$. Then, $\hat{\Phi}_\mathbf{x}$ and $\hat{\Phi}_{\boldsymbol{\xi}}$ are the embedding networks for $x_i, i\in[n]$ and $\xi_j, j\in[q]$, respectively. The features are comprised of the single variable and single-variable specific problem specifications $\mathscr{P}^i_\mathbf{x}, i\in[n]$ and $\mathscr{P}^j_{\boldsymbol{\xi}}, j\in[q]$ for first-stage decisions and scenarios, respectively. The outputs of the $\hat{\Phi}$ networks are aggregated for $\mathbf{x}^*$ and $\boldsymbol{\xi}$ separately. These embeddings are the input of the original NN given in the main part.

4.E. 2RO WITH FIXED FIRST-STAGE DECISIONS

When we compare the calculated solutions of NEUR2RO and the baseline in our experiments, we need to calculate the objective value of a solution $\mathbf{x}^\star \in \mathcal{X}$ exactly or approximately. The former involves solving the AP (4.8) for a given solution. Solving this problem is intractable when we have uncertain parameters in the constraints. We first expand on how the adversarial would be solved in a tractable way if the uncertain parameters only appear in the objective function. Subsequently, we describe an approach to approximately solve the AP, which is based on sampling scenarios from Ξ.

4.E.1. OBJECTIVE UNCERTAINTY

For the special case of objective uncertainty, the AP can be solved much more efficiently. In this case, the adversarial problem is given as

$$\max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\mathcal{Y}} \quad \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y} \tag{4.11a}$$

$$\text{s.t.} \quad W\mathbf{y} \leq \mathbf{h} - T\mathbf{x}^\star, \tag{4.11b}$$

which can be reformulated as

$$\max_{\boldsymbol{\xi}\in\Xi} \quad \alpha \tag{4.12a}$$

$$\text{s.t.} \quad \alpha \leq \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y} \quad \forall \mathbf{y} \in \bar{\mathcal{Y}}, \tag{4.12b}$$

where $\bar{\mathcal{Y}} = \left\{\mathbf{y} \in \mathcal{Y} : \quad W\mathbf{y} \leq \mathbf{h} - T\mathbf{x}^\star\right\}$. While the set $\bar{\mathcal{Y}}$ can contain an exponential number of solutions, the latter problem can be solved by iteratively generating the constraints for $y \in \bar{\mathcal{Y}}$.

4.E.2. CONSTRAINT UNCERTAINTY

We collect all scenarios $\xi \in \Xi$ which were generated during training and during the solution procedures of the baseline algorithm and our algorithm (including the scenarios calculated by the AP) in the set $\Xi^{samples}$. Then for the two returned solutions $\mathbf{x}^*$ and $\mathbf{x}^{baseline}$ we compare

$$\max_{\boldsymbol{\xi}\in\Xi^{samples}} \min_{\mathbf{y}\in\mathcal{Y}} \quad \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y} \tag{4.13a}$$

$$\text{s.t.} \quad W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi}) - T(\boldsymbol{\xi})\mathbf{x}^\star, \tag{4.13b}$$

where we replace $\mathbf{x}$ by the corresponding solution $\mathbf{x}^*$ or $\mathbf{x}^{baseline}$. The latter problem can be solved by calculating the optimal value of the second-stage problem for each scenario independently and choosing the worst-case overall optimal values.

4.F. CONVERGENCE

▷ **Theorem 4.1.** *If $\mathcal{X}$ is finite, the ML-based CCG terminates after a finite number of iterations.*

Proof. The main idea is to show that the condition (4.5) cannot hold in infinitely many iterations. Since we stop the algorithm if 4.5 is not true anymore, then finite termination of the algorithm follows.

Assume the algorithm does not terminate in a finite number of iterations. Let l_t and r_t be the values of the left-hand side and right-hand side of inequality 4.5 in iteration t of the algorithm, i.e.,

$$l_t := \max_{\boldsymbol{\xi} \in \Xi} \quad \mathrm{NN}_\Theta(\mathbf{x}^t, \boldsymbol{\xi})$$

and

$$r_t := \max_{\boldsymbol{\xi} \in \Xi^t} \quad \mathrm{NN}_\Theta(\mathbf{x}^t, \boldsymbol{\xi}).$$

where $\mathbf{x}^t$ is the optimal solution of MP in the t-th iteration and Ξ^t the finite set of scenarios used in the MP in iteration t. Let $\mathbf{x} \in \mathscr{X}$ be a feasible first-stage solution and let $l_t(\mathbf{x})$ and $r_t(\mathbf{x})$ be the sub-sequences which contain the values of l_t and r_t only for the iterations where x is an optimal solution of the MP. Then either this sequence is finite or, if it is infinite, the sequence $\{r_t(\mathbf{x})\}_t$ is monotonous and bounded where monotony follows since $\Xi^t \subset \Xi^{t+1}$ and since the same x is used. The sequence is bounded since Ξ is a bounded set and NN_Θ a piecewise-linear function (as is known for feedforward ReLU networks [151]) and the maximum of a piecewise linear function over a bounded set is bounded. Hence, $\{r_t(x)\}_t$ converges to a finite value $r^\star(\mathbf{x})$. Furthermore, it holds $l_t(\mathbf{x}) \leq r_{t+1}(\mathbf{x})$ since the optimal scenario of the left-hand-side is added to Ξ^t which is a subset of the set later used to evaluate $r_{t+1}(\mathbf{x})$. It follows that

$$r_t(\mathbf{x}) \leq l_t(\mathbf{x}) - \varepsilon \leq r_{t+1}(\mathbf{x}) - \varepsilon$$

for all t which contradicts the convergence of $r_t(\mathbf{x})$. Hence the sequence $r_t(\mathbf{x})$ must be finite. Since only finitely many first-stage solutions $\mathbf{x}$ exist, and the latter result holds for all of them, the number of iterations of the algorithm must be finite. □

4.G. DISTRIBUTIONAL RESULTS FOR RELATIVE PERFORMANCE

In this section, we provide distributional information for the RE for knapsack in Tables 4.3-4.4 and Figures 4.4-4.8.

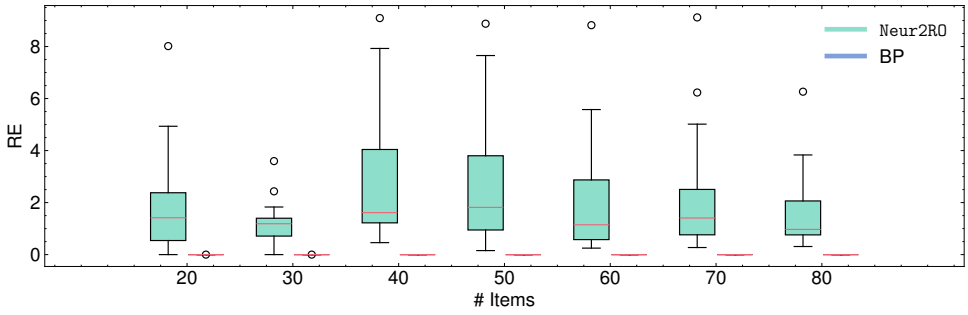


Figure 4.4: Boxplot of RE for baseline and NEUR2RO on UN knapsack instances.

Table 4.3: Table of distributional information for knapsack. For each row, all RE statistics are computed over 18 instances.

Correlation Type	# items	Mean RE		Median RE		RE 1st Quartile		RE 3rd Quartile	
		NEUR2RO	BP	NEUR2RO	BP	NEUR2RO	BP	NEUR2RO	BP
Uncorrelated	20	2.005	**0.000**	1.417	**0.000**	0.541	**0.000**	2.379	**0.000**
	30	1.189	**0.000**	1.188	**0.000**	0.712	**0.000**	1.399	**0.000**
	40	2.895	**0.000**	1.614	**0.000**	1.221	**0.000**	4.042	**0.000**
	50	3.032	**0.000**	1.814	**0.000**	0.946	**0.000**	3.801	**0.000**
	60	2.099	**0.000**	1.146	**0.000**	0.577	**0.000**	2.872	**0.000**
	70	2.214	**0.000**	1.408	**0.000**	0.761	**0.000**	2.506	**0.000**
	80	1.591	**0.000**	0.968	**0.000**	0.758	**0.000**	2.063	**0.000**
Weakly Correlated	20	2.569	**0.000**	1.582	**0.000**	1.229	**0.000**	4.010	**0.000**
	30	2.664	**0.000**	2.236	**0.000**	0.616	**0.000**	4.293	**0.000**
	40	2.320	**0.000**	1.595	**0.000**	1.164	**0.000**	2.292	**0.000**
	50	2.183	**0.145**	1.757	**0.000**	0.793	**0.000**	2.674	**0.000**
	60	2.165	**0.390**	0.695	**0.000**	**0.000**	**0.000**	3.445	**0.458**
	70	0.884	**0.338**	0.165	**0.000**	**0.000**	**0.000**	0.623	**0.175**
	80	**0.392**	0.691	**0.000**	0.341	**0.000**	**0.000**	**0.165**	0.831
Almost Strongly Correlated	20	2.355	**0.000**	1.439	**0.000**	0.000	**0.000**	2.757	**0.000**
	30	1.166	**0.113**	0.782	**0.000**	0.075	**0.000**	1.911	**0.000**
	40	0.825	**0.335**	0.497	**0.000**	0.019	**0.000**	1.606	**0.000**
	50	**0.314**	0.884	0.019	**0.000**	**0.000**	**0.000**	**0.229**	1.251
	60	**0.197**	0.523	**0.000**	0.016	**0.000**	**0.000**	**0.268**	1.129
	70	**0.551**	0.615	**0.017**	0.031	**0.000**	**0.000**	**1.058**	1.227
	80	**0.388**	0.694	**0.000**	0.265	**0.000**	**0.000**	**0.554**	0.770
Strongly Correlated	20	2.387	**0.000**	1.604	**0.000**	0.905	**0.000**	3.018	**0.000**
	30	1.068	**0.121**	0.610	**0.000**	0.054	**0.000**	1.939	**0.000**
	40	0.658	**0.191**	0.443	**0.000**	0.002	**0.000**	0.888	**0.000**
	50	**0.411**	0.648	0.073	**0.000**	**0.000**	**0.000**	**0.780**	0.963
	60	**0.322**	0.367	0.042	**0.010**	**0.000**	**0.000**	**0.173**	0.693
	70	**0.389**	0.738	**0.020**	0.027	**0.000**	**0.000**	**0.535**	0.793
	80	**0.318**	0.668	**0.000**	0.179	**0.000**	**0.000**	**0.245**	0.906

Table 4.4: Table of distributional information for capital budgeting. For each row, all RE statistics are computed over 50 instances.

# items	Mean RE				Median RE				RE 1st Quartile				RE 3rd Quartile			
	NEUR2RO	$K=2$	$K=5$	$K=10$	NEUR2RO	$K=2$	$K=5$	$K=10$	NEUR2RO	$K=2$	$K=5$	$K=10$	NEUR2RO	$K=2$	$K=5$	$K=10$
10	2.558	2.849	**1.029**	1.165	1.105	1.140	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	3.534	4.349	**0.547**	1.557
20	0.423	0.304	**0.232**	0.266	**0.000**	0.196	0.112	0.064	**0.000**	0.094	0.013	0.000	0.410	0.453	**0.320**	0.362
30	0.408	0.149	0.131	**0.084**	0.109	**0.020**	0.073	0.032	0.002	**0.000**	0.003	0.000	0.337	0.182	0.212	**0.110**
40	0.234	0.114	0.098	**0.073**	**0.009**	0.074	0.011	0.019	**0.000**	0.001	**0.000**	0.002	**0.121**	0.180	0.137	0.137
50	0.090	0.107	0.090	**0.056**	**0.001**	0.033	0.039	0.020	**0.000**	0.000	**0.000**	0.002	**0.050**	0.193	0.139	0.084

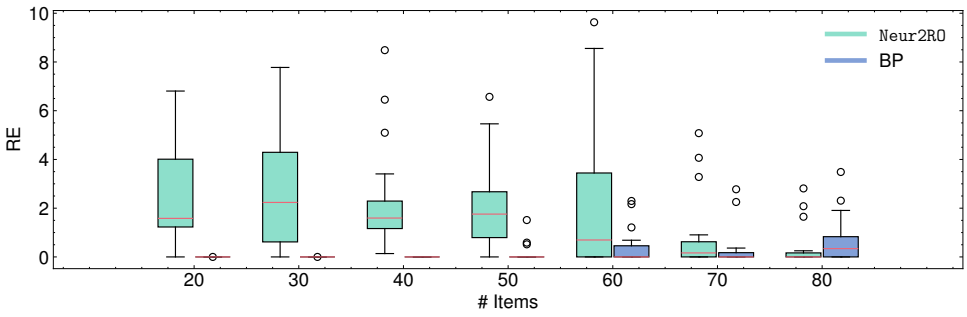


Figure 4.5: Boxplot of RE for baseline and NEUR2RO on WC knapsack instances.

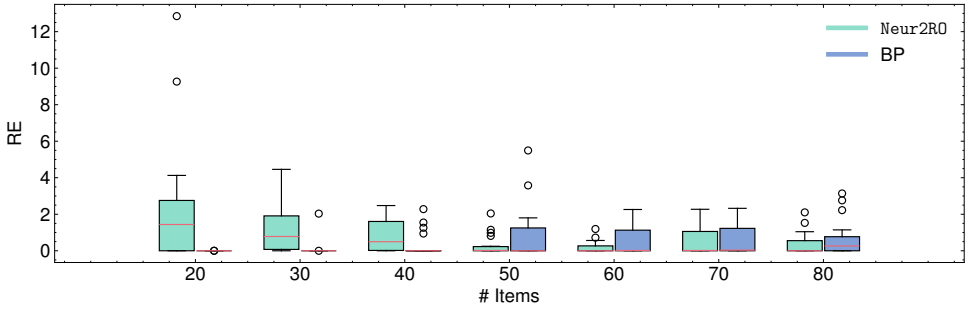


Figure 4.6: Boxplot of RE for baseline and NEUR2RO on ASC knapsack instances.

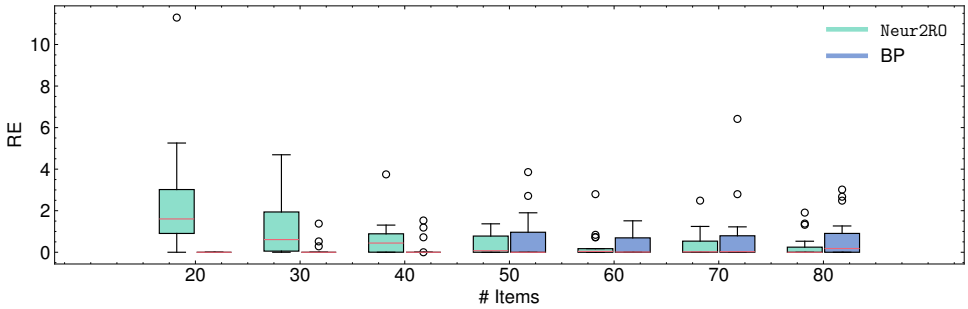


Figure 4.7: Boxplot of RE for baseline and NEUR2RO on SC knapsack instances.

4.H. ABLATION

This section presents an ablation across two aspects of NEUR2RO, namely, the formulation of the MP and the method to obtain worst-case scenarios. Both results are presented on the knapsack instances.

4.H.1. MAIN PROBLEM FORMULATION

As an alternative to the formulation using argmax over a set of scenarios. One more straightforward formulation is to consider instead the max over all of the scenarios, which is given by

$$\min_{\mathbf{x}\in\mathscr{X},\alpha} \quad \alpha \tag{4.14a}$$

$$\text{s.t.} \quad \alpha \geq \text{NN}_\Theta(\mathbf{x}, \boldsymbol{\xi}_i) \qquad \forall k \in \{1,\dots,K\}. \tag{4.14b}$$

Table 4.5 reports the MRE of the argmax and max formulations and the solving time. Table 4.5 demonstrates an improvement in solution quality, with argmax obtaining a lower MRE in every case and a lower computing time in most cases.

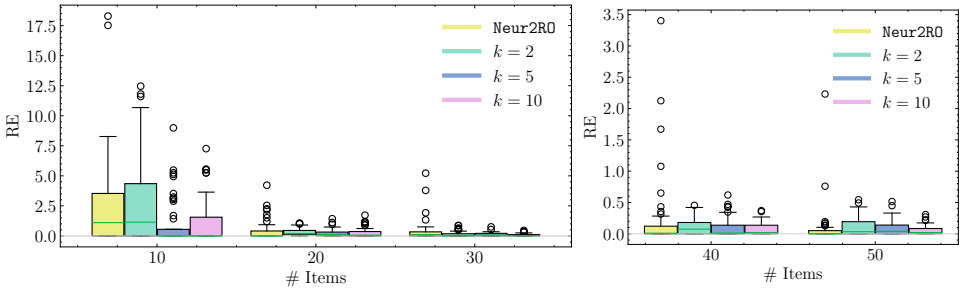


Figure 4.8: Box plot of RE for K-adaptability baselines (where lower-case k is used in legend) and NEUR2RO on capital budgeting instances.

4.H.2. WORST-CASE SCENARIO ACQUISITION

This section compares the adversarial approach for determining scenarios to a sampling and a linear programming (LP) relaxation-based approach.

SAMPLING-BASED SCENARIO ACQUISITION

For sampling, as a baseline, we sample 100,000 scenarios, and then to approximate the AP, we take the maximizer over a forward pass. Table 4.6 demonstrates a clear trade-off between solution quality and efficiency. Generally, sampling improves average solving time across all instances but leads to worse solution quality as the instance size increases.

LP RELAXATION-BASED SCENARIO ACQUISITION

For 2RO, the uncertainty set is often polyhedral, which scenarios can be heuristically obtained via a LP relaxation. For the LP relaxation, we compare the performance of the standard MILP-based scenario acquisition (standard), i.e., solving the AP to optimality, to the relaxation (LP relaxation). For both problems, we report the RE to the baselines. Tables 4.7 and 4.8 present the knapsack and capital budgeting results, respectively. In general, we can observe that the LP relaxation leads to significantly faster solving time, with an overall decreased solution quality. That being said, for capital budgeting in particular, NEUR2RO with the LP relaxation still achieves a lower median RE than the baselines on larger instances, while being roughly five times faster than results without the relaxation.

4.H.3. PREDICTION TARGET

This section compares the prediction target. For capital budgeting, the coefficients of the first-stage decisions in the objective contain uncertainty. As such, this presents a choice of either predicting the sum of the first- and second-stage objectives, i.e., $\mathbf{c}(\boldsymbol{\xi})^\intercal \mathbf{x} + \min_{\mathbf{y}\in\mathscr{Y}} \{\mathbf{d}(\boldsymbol{\xi})^\intercal \mathbf{y} : W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi}) - T(\boldsymbol{\xi})\mathbf{x}\}$, or only the second-stage objective, i.e., $\min_{\mathbf{y}\in\mathscr{Y}} \{\mathbf{d}(\boldsymbol{\xi})^\intercal \mathbf{y} : W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi}) - T(\boldsymbol{\xi})\mathbf{x}\}$. Specifically, we compare the downstream optimization performance with respect to the resulting formulations. The formulation for predicting the sum of the first- and second-stage objectives is presented in Section 4.4.

Table 4.5: argmax and max formulations on knapsack instances. For each row, the median RE and solving time are computed over 18 instances. All times in seconds.

Correlation Type	# items	Median RE		Times	
		argmax	max	argmax	max
Uncorrelated	20	**0.000**	1.167	**5**	11
	30	**0.000**	0.945	**7**	14
	40	**0.000**	1.931	**9**	24
	50	**0.000**	1.634	**10**	33
	60	**0.000**	0.452	**17**	29
	70	**0.000**	0.801	**19**	28
	80	**0.000**	2.227	**13**	35
Weakly Correlated	20	**0.000**	3.515	**6**	13
	30	**0.000**	2.405	**11**	22
	40	**0.000**	0.502	**26**	42
	50	**0.000**	0.254	**24**	39
	60	**0.000**	1.528	77	**58**
	70	**0.000**	1.769	**18**	35
	80	**0.000**	3.492	**27**	75
Almost Strongly Correlated	20	**0.000**	2.042	**5**	12
	30	**0.000**	1.433	**6**	14
	40	**0.000**	1.739	**11**	33
	50	**0.000**	3.161	**8**	20
	60	**0.000**	2.449	**15**	30
	70	**0.000**	2.497	**18**	35
	80	**0.000**	1.824	**17**	30
Strongly Correlated	20	**0.000**	1.154	**5**	11
	30	**0.000**	0.967	**7**	15
	40	**0.000**	1.928	**16**	28
	50	**0.000**	3.613	**10**	21
	60	**0.000**	2.005	**20**	26
	70	**0.000**	2.657	**16**	33
	80	**0.000**	2.051	**16**	28

For predicting the second-stage objective only, the MP is given by

$$\min_{\mathbf{x}\in\mathscr{X},\mathbf{y}\in\mathscr{Y},\boldsymbol{\xi}_a\in\Xi} \quad \mathbf{c}(\boldsymbol{\xi}_a)^\top\mathbf{x}+\mathbf{d}(\boldsymbol{\xi}_a)^\top\mathbf{y} \tag{4.15a}$$

$$\text{s.t.} \quad W(\boldsymbol{\xi}_a)\mathbf{y}+T(\boldsymbol{\xi}_a)\mathbf{x}\leq\mathbf{h}(\boldsymbol{\xi}_a), \tag{4.15b}$$

$$\boldsymbol{\xi}_a\in\operatorname{argmax}_{\boldsymbol{\xi}\in\Xi'}\left\{\mathbf{c}(\boldsymbol{\xi})^\top\mathbf{x}+\mathrm{NN}_\Theta(\mathbf{x},\boldsymbol{\xi})\right\}, \tag{4.15c}$$

and the AP is given by

$$\max_{\boldsymbol{\xi}\in\Xi}\ \mathbf{c}(\boldsymbol{\xi})^\top\mathbf{x}^\star+\mathrm{NN}_\Theta(\mathbf{x}^\star,\boldsymbol{\xi}). \tag{4.16}$$

The main difference with this formulation is that the objective coefficients $\mathbf{c}(\boldsymbol{\xi})$ can be utilized directly rather than requiring the ML model to predict them. Table 4.9 compares the two approaches on the capital budgeting instances wherein the RE is computed with respect to the baselines. Empirically, we can see that predicting the sum of the first- and second-stage objectives yields significantly better solutions. On the methodological side, when only the second stage is predicted each node in the branch-and-bound tree being explored by a MIP solver will contain the exact first-stage and the predicted second-stage objectives. As such, we speculate that the LP relaxation at each node will consist of two components that are on entirely different scales. Specifically, the first-stage objective will be tight as it is being represented exactly while the second-stage

Table 4.6: Adversarial and sampling-based approaches for worst-case scenario acquisition on knapsack instances. For each row, the median RE and solving time are computed over 50 instances. All times in seconds.

Correlation Type	# items	Median RE		Times	
		adversarial	sampling	adversarial	sampling
Uncorrelated	20	**0.000**	**0.000**	5	**2**
	30	**0.000**	**0.000**	7	**4**
	40	0.560	**0.000**	9	**4**
	50	0.723	**0.000**	10	**5**
	60	0.066	**0.000**	17	**6**
	70	0.150	**0.000**	19	**8**
	80	0.395	**0.000**	13	**9**
Weakly Correlated	20	**0.000**	0.074	6	**3**
	30	**0.000**	0.444	11	**4**
	40	**0.000**	0.093	26	**5**
	50	0.441	**0.000**	24	**7**
	60	0.119	**0.065**	77	**9**
	70	**0.000**	0.185	18	**8**
	80	**0.000**	0.536	27	**9**
Almost Strongly Correlated	20	**0.000**	**0.000**	**5**	5
	30	**0.000**	**0.000**	**6**	6
	40	**0.000**	**0.000**	11	**10**
	50	**0.000**	**0.000**	8	**7**
	60	**0.000**	**0.000**	15	**14**
	70	**0.000**	**0.000**	18	**13**
	80	**0.000**	**0.000**	17	**12**
Strongly Correlated	20	**0.000**	**0.000**	5	**5**
	30	**0.000**	**0.000**	7	**7**
	40	**0.000**	**0.000**	16	**11**
	50	**0.000**	**0.000**	10	**8**
	60	**0.000**	**0.000**	20	**13**
	70	**0.000**	**0.000**	16	**14**
	80	**0.000**	**0.000**	16	**13**

Table 4.7: Median RE and solving times for knapsack instances with LP relaxation. For each row, the median RE and average solving time are computed over 18 instances. All times in seconds. The smallest (best) values in each row/metric are in bold.

Correlation Type	# items	Median RE		Times	
		standard	LP relaxation	standard	LP relaxation
Uncorrelated	20	**1.417**	1.673	4	**1**
	30	1.188	**1.167**	6	**1**
	40	1.614	**1.387**	9	**2**
	50	1.814	**1.660**	9	**2**
	60	**1.146**	**1.146**	14	**1**
	70	1.408	**1.166**	16	**2**
	80	0.986	**0.970**	11	**2**
Weakly Correlated	20	1.582	**1.454**	5	**1**
	30	2.236	**2.034**	11	**1**
	40	**1.595**	2.733	20	**2**
	50	1.757	**1.126**	19	**2**
	60	**0.695**	0.729	77	**3**
	70	**0.165**	0.243	15	**3**
	80	**0.000**	0.316	21	**9**

Correlation Type	# items	Median RE		Times	
		standard	LP relaxation	standard	LP relaxation
Almost Strongly Correlated	20	1.439	**1.211**	5	**1**
	30	0.782	**0.665**	6	**1**
	40	**0.497**	0.927	10	**2**
	50	**0.019**	1.884	7	**2**
	60	**0.000**	1.079	14	**2**
	70	**0.017**	0.025	13	**4**
	80	**0.000**	1.775	12	**4**
Strongly Correlated	20	1.604	**1.368**	5	**1**
	30	**0.610**	0.796	7	**2**
	40	**0.443**	1.375	11	**3**
	50	**0.073**	2.333	9	**2**
	60	**0.042**	0.510	11	**4**
	70	**0.020**	0.623	16	**3**
	80	**0.000**	1.097	13	**3**

objective requires the relaxation of the prediction model which will not be tight due to the big-M constraints. This means that the maximization problem in the AP favors the second stage. This mismatch could lead to inaccurate scenarios and undesirable downstream effects within branch-and-bound.

Table 4.8: Median RE and solving times for capital budgeting instances with LP relaxation. For each row, the median RE and average solving time are computed over 50 instances. All times in seconds. The smallest (best) values in each row/metric are in bold.

# items	Median RE		Times	
	standard	LP relaxation	standard	LP relaxation
10	**1.105**	2.663	59	**4**
20	**0.000**	0.060	324	**142**
30	0.109	**0.071**	602	**141**
40	0.009	**0.007**	739	**226**
50	**0.001**	0.001	1,032	**231**

Table 4.9: Sum and second-stage only predictions for capital budgeting instances. For each row, the median RE and solving time are computed over 50 instances. Note that in these results, the RE is calculated with respect to the K-adaptability and each respective ML-approach. All times in seconds.

# items	Median RE		Times	
	sum	second only	sum	second-only
10	**1.105**	2.424	**20**	233
20	**0.000**	0.192	**324**	1,823
30	**0.109**	0.151	**602**	3,823
40	**0.009**	0.010	**739**	4,062
50	**0.001**	0.005	**1,032**	7,424

4.H.4. BASELINE SOLUTION QUALITY AT NEUR2RO TERMINATION TIME

In this section, we report the objective quality, i.e., the median relative error, for K-adaptability baseline at the termination time of NEUR2RO in Table 4.10. From the table, we can see that the performance is median RE of NEUR2RO is marginally better than when K-adaptability is given 3 hours, except $n = 20, 40$. Note that these tables are only be reproduced for capital budgeting as we do not have the knapsack results throughout the solving process, given only the final objective values are reported in Arslan and Detienne [6].

Table 4.10: Median RE for capital budgeting at 3 hour time limit and NEUR2RO termination time. For each row, the median RE and average solving time are computed over 50 instances. All times in seconds. The smallest (best) values in each row/metric are in bold.

# items	Median RE at 3 hours				Median RE at NEUR2RO termination			
	NEUR2RO	$K = 2$	$K = 5$	$K = 10$	NEUR2RO	$K = 2$	$K = 5$	$K = 10$
10	1.105	1.140	**0.000**	**0.000**	0.809	1.559	**0.267**	0.359
20	**0.000**	0.196	0.112	0.064	**0.011**	0.240	0.098	0.084
30	0.109	**0.020**	0.073	0.032	0.102	0.067	0.093	**0.029**
40	**0.009**	0.074	0.011	0.019	**0.013**	0.079	0.058	0.019
50	**0.001**	0.033	0.039	0.020	**0.002**	0.035	0.006	0.008

4.I. MACHINE-LEARNING MODEL DETAILS

4.I.1. FEATURES

Here we provide the features for each of the problems. In both cases, set-based architectures [226] with parameter sharing utilized, so we report the features for a single dimension of the first-stage decision and scenario accordingly. Table 4.11 reports all of the features for each instance.

Table 4.11: Features for first-stage decision and scenario embedding networks.

Problem	First-Stage Features	Scenario Features
Knapsack	$x_i, f_i, \bar{p}_i, \hat{p}_i, r_i, c_i, t_i, C$	$\xi_i, f_i, \bar{p}_i, \hat{p}_i, r_i, c_i, t_i, C$
Capital budgeting	x_i, r_i, c_i	$\left(1+\mathbf{\Phi}_i^\intercal \boldsymbol{\xi}/2\right)_i, \left(1+\mathbf{\Psi}_i^\intercal \boldsymbol{\xi}/2\right)_i, r_i, c_i$

4.I.2. MODEL HYPERPARAMETERS

This section reports the hyperparameters for the neural networks for each problem. For both problems, we have the same architecture with slightly different hyperparameters. As the objective of NEUR2RO is to enable efficient optimization, we train small networks that can achieve a low mean absolute error value to ensure that the main and adversarial problems are tractable. For this reason, no systematic hyperparameter tuning was done. Hyperparameter optimization would likely only further improve the already strong numerical results. For both problems, we train a model for 500 epochs and compute the mean absolute error on a validation set every 10 epochs. We then use the model with the lowest reported mean absolute validation error during training for evaluation.

Table 4.12 reports the hyperparameters for each model. As our model generalizes across instances, which requires invariance to the order and number of decision variables, both the first-stage and scenario embedding networks are set-based architectures [226]. We refer to Figure 4.3 for a refresher on the overall architecture which has the following hyperparameters. The hyperparameters "$\hat{\Phi}_{\mathbf{x}}$ dimensions" and "$\Phi_{\mathbf{x}}$ dimensions" correspond to the hidden and embedding dimensions of the first-stage embedding network. Specifically, "$\hat{\Phi}_{\mathbf{x}}$ dimensions" corresponds to the network with shared parameters that embed the representation for each first-stage decision. The last dimension of "$\hat{\Phi}_{\mathbf{x}}$ dimensions" is that of the aggregated vector. The hyperparameter "$\Phi_{\mathbf{x}}$ dimensions" corresponds to the network that takes the aggregated first-stage embedding vector as input. The last dimension of "$\Phi_{\mathbf{x}}$ dimensions" specifies the embedding dimension of the first-stage embedding network. "$\hat{\Phi}_{\boldsymbol{\xi}}$ dimensions" and "$\Phi_{\boldsymbol{\xi}}$ dimensions" are analogous for the scenario embedding network. "Φ dimensions" correspond to the hidden dimensions of the value network. Finally, "aggregation type" specifies the type of aggregation that combines the first-stage/scenario embeddings.

4.I.3. TRAINING CURVES

Figures 4.9-4.10 plot the mean absolute error at every 10 epochs during training for the training and validation data. Generally, the training and validation mean absolute error is very close, and in both problems, a relatively low mean absolute error is achieved.

Table 4.12: Hyperparameters for neural networks.

Hyperparameter	Knapsack	Capital budgeting
Feature scaling	min-max	min-max
Label scaling	min-max	min-max
# epochs	500	500
Batch size	256	256
Learning rate	0.001	0.001
Dropout	0	0
Loss function	MSELoss	MSELoss
Optimizer	Adam	Adam
$\hat{\Phi}_{\mathbf{x}}$ dimensions	[32, 16]	[16, 4]
$\Phi_{\mathbf{x}}$ dimensions	[64, 8]	[32, 8]
$\hat{\Phi}_{\boldsymbol{\xi}}$ dimensions	[32, 16]	[16, 4]
$\Phi_{\boldsymbol{\xi}}$ dimensions	[64, 8]	[32, 8]
Φ dimensions	[8]	[8]
Aggregation type	sum	sum

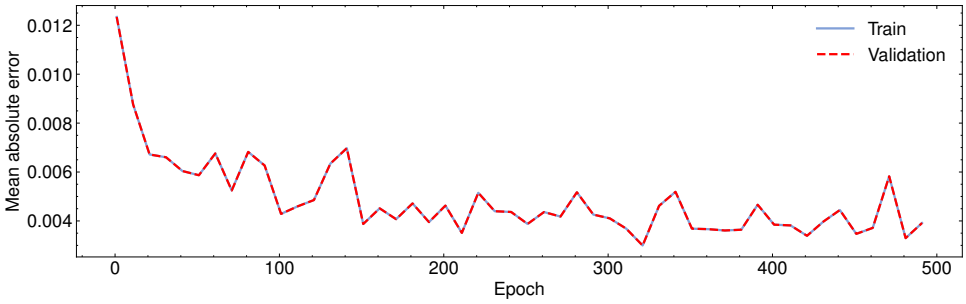


Figure 4.9: Training curve for knapsack.

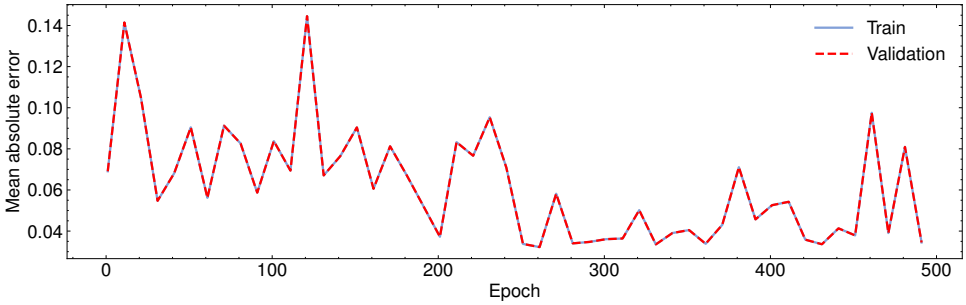


Figure 4.10: Training curve for capital budgeting.

5

MACHINE LEARNING FOR K-ADAPTABILITY

Two-stage robust optimization problems constitute one of the hardest optimization problem classes. One of the solution approaches to this class of problems is K-adaptability. This approach simultaneously seeks the best partitioning of the uncertainty set of scenarios into K subsets, and optimizes decisions corresponding to each of these subsets. In general case, it is solved using the K-adaptability branch-and-bound algorithm, which requires exploration of exponentially-growing solution trees.

Proposed data-driven algorithm: *To accelerate finding high-quality solutions in such trees, we propose a machine learning-based node selection strategy. In particular, we construct a feature engineering scheme based on general two-stage robust optimization insights such that a model can be applied as-is to problems of different sizes and/or types.*

Data generation scheme: *To sample accurate data points seen during the algorithm execution, the top part of B&B trees of multiple instances are resolved, and labeled using Monte-Carlo Tree Search-like approaches. To construct the partial exhaustive tree, single-level problems are solved, with each of them giving rise to one data point.*

Experimental results: *We experimentally show that using our learned node selection strategy outperforms a vanilla, random node selection strategy when tested on problems of the same type as the training problems, also in case the K-value or the problem size differs from the training ones.*

This chapter is based on Julien et al. [111] published in **INFORMS Journal on Computing**. In collaboration with Krzysztof Postek and Ş. İlker Birbil. The code is available at `https://github.com/estherjulien/KAdaptNS`.

5.1. INTRODUCTION

Many optimization problems are affected by data uncertainty caused by errors in the forecast, implementation, or measurement. Robust optimization (RO) is one of the key paradigms to solve such problems, where the goal is to find an optimal solution among the ones that remain feasible for all data realizations within an *uncertainty set* [21]. This set includes all *reasonable* data outcomes.

A specific class of RO problems comprises two-stage robust optimization (2RO) problems in which some decisions are implemented *before* the uncertain data is known (here-and-now decisions), and other decisions are implemented *after* the data is revealed (wait-and-see decisions). Such a problem can be formulated as

$$\min_{\mathbf{x}\in\mathscr{X}} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\mathscr{Y}} \left\{\mathbf{c}(\boldsymbol{\xi})^\top\mathbf{x}+\mathbf{d}(\boldsymbol{\xi})^\top\mathbf{y} : T(\boldsymbol{\xi})\mathbf{x}+W(\boldsymbol{\xi})\mathbf{y}\leq\mathbf{h}(\boldsymbol{\xi}),\ \forall\boldsymbol{\xi}\in\Xi\right\}, \tag{5.1}$$

where $\mathbf{x}\in\mathscr{X}\subseteq\mathbb{R}^{N_x}$ and $\mathbf{y}\in\mathscr{Y}\subseteq\mathbb{R}^{N_y}$ are the here-and-now and wait-and-see decisions, respectively, and $\boldsymbol{\xi}$ is the vector of initially unknown data belonging to the uncertainty set $\Xi\subseteq\mathbb{R}^{N_z}$. Solving problem (5.1) is difficult in general, since Ξ might include an infinite number of scenarios, and hence different values of $\mathbf{y}$ might be optimal for different realizations of $\boldsymbol{\xi}$. In fact, finding optimal $\mathbf{x}$ is an NP-hard problem [22]. Several approaches have been proposed to address this difficulty. The first one is to use so-called decision rules which explicitly formulate the second-stage decision $\mathbf{y}$ as a function of $\boldsymbol{\xi}$, and hence the function parameters become first-stage decisions next to $\mathbf{x}$; see [22]. Another approach is to partition Ξ into subsets and to assign a separate copy of $\mathbf{y}$ to each of the subsets. The partitioning is then iteratively refined, and the decisions become increasingly *customized* to the outcomes of $\boldsymbol{\xi}$.

In this chapter, we consider a third approach to (5.1) known as K-adaptability. There, at most K possible wait-and-see decisions $\mathbf{y}_1,\ldots,\mathbf{y}_K$ are allowed to be constructed, and the decision maker must select one of those. The values of the possible $\mathbf{y}_k$'s become the first-stage variables, and the problem boils down to

$$\min_{\mathbf{x}\in\mathscr{X},\mathbf{y}\in\mathscr{Y}^K} \max_{\boldsymbol{\xi}\in\Xi} \min_{k\in\mathscr{K}} \left\{\mathbf{c}(\boldsymbol{\xi})^\top\mathbf{x}+\mathbf{d}(\boldsymbol{\xi})^\top\mathbf{y}_k : T(\boldsymbol{\xi})\mathbf{x}+W(\boldsymbol{\xi})\mathbf{y}_k\leq\mathbf{h}(\boldsymbol{\xi})\right\}, \tag{5.2}$$

where $\mathscr{K}=\{1,\ldots,K\}$ and $\mathscr{Y}^K=\bigtimes_{k=1}^K\mathscr{Y}$. Although the solution space of (5.2) is finite-dimensional, it remains an NP-hard problem [28]. For certain cases, (5.2) can be equivalently rewritten as a mixed integer linear programming (MILP) model [95].

The above formulation requires that for given $\mathbf{x}\in\mathscr{X}$ and $\boldsymbol{\xi}\in\Xi$, there is at least one decision $\mathbf{y}_k$, $k\in\mathscr{K}$ satisfying $T(\boldsymbol{\xi})\mathbf{x}+W(\boldsymbol{\xi})\mathbf{y}_k\leq\mathbf{h}(\boldsymbol{\xi})$, and among those one (or more) minimizing the objective. Looking at (5.2) from the point of view $\mathbf{y}_k$, we can say that for each $\mathbf{y}_k$, we can identify a subset Ξ_k of Ξ for which a given $\mathbf{y}_k$ is optimal among K selected recourses. The union of sets Ξ_k, $k\in\mathscr{K}$ is equal to Ξ although they need not be mutually disjoint (but a mutually disjoint partition of Ξ can be constructed). Consequently, solving (5.2) involves implicitly (i) clustering Ξ, and (ii) optimizing the per-cluster decision so that the objective function corresponding to *the most difficult cluster* is minimized. Such simultaneous clustering and per-cluster optimization also occur, for example in retail. A line of K products is to be designed to attract the largest possible group of customers. The customers are clustered into K groups, and the nature of the products is guided by the cluster characteristics.

In this chapter, we focus on the general MILP K-adaptability case for which the only existing solution approach is the K-adaptability branch-and-bound (K-B&B) algorithm of Subramanyam et al. [189]. Other methods to solve the K-adaptability problem have been proposed by Hanasusanto et al. [95] that deals with binary decisions, and Ghahtarani et al. [83] that assumes integer first-stage decisions. The K-B&B algorithm, as opposed to the top-down partitioning of Ξ of Bertsimas and Dunning [30] or Postek and den Hertog [170], proceeds by gradually building up discrete subsets Ξ'_k of scenarios. In most practical cases, a solution to (5.2), where $\mathbf{y}_1,\ldots,\mathbf{y}_K$ are feasible for large $\Xi'_1,\ldots,\Xi'_K$, is also feasible to the original problem. The problem, however, lies in knowing which scenarios should be grouped together. In other words, a decision needs to be made on which scenarios of Ξ should be responded to with the same decision. How well this question is answered, determines the (sub)optimality of $\mathbf{y}_1,\ldots,\mathbf{y}_K$. In Subramanyam et al. [189], a search tree is used to determine the best collection (see Section 5.2 for details). However, this approach suffers from exponential growth.

We introduce a method for learning the best strategy to explore this tree. In particular, we learn which nodes to evaluate next in depth-first search *dives* to obtain good solutions faster. These predictions are made using a supervised machine learning (ML) model. As the input does not range for different instance sizes, or values of K, as will be explained in future sections, the ML model does not require difficult architectures. Standard ML models, such as feed-forward neural networks, random forests, or support vector machines can be used for this work.

Due to the supervised nature, some *oracle* is required to be imitated. In the design of this oracle, we are partly inspired by Monte Carlo tree search (MCTS) [44], which is often used for exploring large trees. Namely, the training data is obtained by exploring K-B&B trees via an adaptation of MCTS (see Section 5.3.4). The scores given to the nodes in the MCTS-like exploration are stored and used as labels in our training data. In the field of solving MILPs, learning node selection to speed up exploring the B&B tree has been done, *e.g.*, by [98]. Here, a node selection policy is designed by imitating an oracle. This oracle is constructed using the optimal solutions of various MILP data sets. More recently, Khalil et al. [115] used a graph neural network to learn node selection. Our method distinguishes itself from these approaches as we specifically use the nature of our problem. Namely, in the design of the node selection strategy, we use the actual meaning of selecting a node; adding a scenario to a subset. Therefore, we try to learn what characteristics (or features) scenarios that should belong to the same subset have. For an overview of ML for learning branching policies in B&B, see Bengio et al. [23]. There has also been a vast amount of research on applying MCTS directly to solving combinatorial problems. In Sabharwal et al. [173] a special case of MCTS called Upper Confidence bounds for Trees (UCT), is used for designing a node selection strategy to explore B&B trees (for MIPs). In Khalil et al. [116] MCTS is used to find the best backdoor (i.e., a subset of variables used for branching) for solving MIPs. Loth et al. [141] have used MCTS for enhancing constraint programming solvers, which naturally use a search tree for solving combinatorial problems. For an elaborate overview on modifications and applications of MCTS, we refer to Świechowski et al. [191].

The remainder of the chapter is structured as follows. In Section 5.2 we describe the inner workings of the K-adaptability branch-and-bound to set the stage for our contri-

bution. In Section 5.3 we outline our ML methodology along with the data generation procedure. Section 5.4 discusses the results of a numerical study, and Section 5.5 concludes with some remarks on future works.

5.2. PRELIMINARIES

It is instructive to conceptualize a solution to (5.2) as a solution to a nested clustering and optimization-for-clusters methodology. As already mentioned in Section 5.1, a feasible solution to (5.2) can be used to construct a partition of the uncertainty set into subsets $\Xi_1, \ldots, \Xi_K$ such that $\bigcup_{k=1}^{K} \Xi_k = \Xi$. Here, decision $\mathbf{y}_k$ is applied in the second stage if $\boldsymbol{\xi} \in \Xi_k$. The decision framework associated with a given solution is illustrated in Figure 5.1.

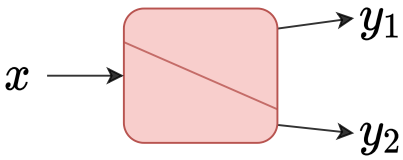


Figure 5.1: A framework of the K-adaptability problem, where we split the uncertainty set (red box) in $K = 2$ parts. Here, $\mathbf{x}$ represents the first-stage decisions, and $\mathbf{y}_1$ with $\mathbf{y}_2$ those of the second-stage.

For such a *fixed partition* the corresponding optimization problem becomes

$$\min_{\mathbf{x}\in\mathscr{X}, \mathbf{y}\in\mathscr{Y}^K} \max_{k\in\mathscr{K}} \max_{\boldsymbol{\xi}\in\Xi_k} \{\mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x} + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y}_k\} \tag{5.3a}$$

$$\text{s.t.} \quad T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y}_k \leq \mathbf{h}(\boldsymbol{\xi}). \tag{5.3b}$$

The optimal solution to (5.2) also corresponds to an optimal partitioning of Ξ, and the optimal decisions of (5.3) with that partitioning. Finding an optimal partition and the corresponding decisions has been shown to be NP-hard by Bertsimas and Caramanis [28]. For that reason, Subramanyam et al. [189] have proposed the K-B&B algorithm. There, the idea is to gradually build up a collection of finite subsets $\Xi'_1, \ldots, \Xi'_K$, such that for each $k \in \mathscr{K}$ an optimal solution to (5.3) with $\Xi_k = \Xi'_k$ is also an optimal solution to (5.2).

The algorithm follows a main-subproblem approach. The main problem solves (5.2) with K finite subsets of scenarios. The subproblem finds the scenario for which the current main solution is not robust. The number K of possible assignments of this new scenario to one of the existing subsets gives rise to using a search tree. Each tree node corresponds to a partition of all scenarios found so far into K subsets. The goal is to find the node with the best partition. An illustration of the search tree is given in Figure 5.2. The tree grows exponentially and thus only (very) small-scale problems can be solved in reasonable time. The method we propose in the next section learns a good node selection strategy with the goal of converging to the optimal solution much faster than K-B&B.

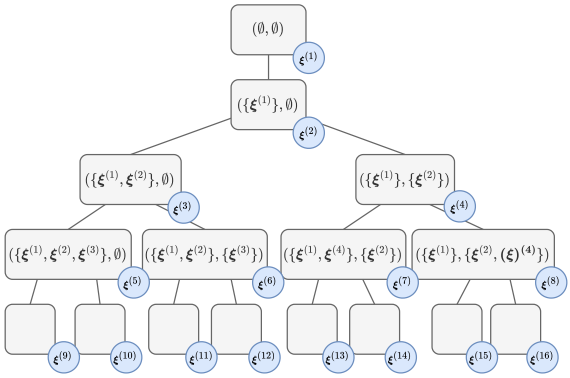


Figure 5.2: Search tree for K-adaptability branch-and-bound ($K = 2$).

5

Main problem. This problem solves the K-adaptability problem (5.2) with respect to the currently found scenarios grouped into $\Xi'_k \subset \Xi$ for all $k \in \mathcal{K}$. For a collection $\Xi'_1, \ldots, \Xi'_K$, the problem formulation is defined as follows:

$$\min_{\theta \in \mathbb{R}, \mathbf{x} \in \mathcal{X}, \mathbf{y} \in \mathcal{Y}^K} \theta \tag{5.4a}$$

$$\text{s.t.} \quad \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x} + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y}_k \leq \theta, \qquad \forall \boldsymbol{\xi} \in \Xi'_k, \forall k \in \mathcal{K}, \tag{5.4b}$$

$$T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y}_k \leq \mathbf{h}(\boldsymbol{\xi}), \qquad \forall \boldsymbol{\xi} \in \Xi'_k, \forall k \in \mathcal{K}, \tag{5.4c}$$

where θ is the current estimate of the objective function value. We denote the optimal solution of (5.4) by the triplet $(\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star)$.

Subproblem. The subproblem aims to find a scenario $\boldsymbol{\xi}$ for which the current main solution is infeasible. That is, a scenario is found such that for each k, at least one of the following is true:

- the current estimate of $\theta^\star$ is too low, i.e., $\mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y}^\star_k > \theta^\star$;
- at least one of the original constraints is violated, i.e., $T(\boldsymbol{\xi})\mathbf{x}^\star + W(\boldsymbol{\xi})\mathbf{y}^\star_k > \mathbf{h}(\boldsymbol{\xi})$.

If no such scenario exists, we define the solution $(\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star)$ as a robust solution. When such a scenario $\boldsymbol{\xi}^\star$ does exist, the solution is not robust and the newly-found scenario is assigned to one of the sets $\Xi'_1, \ldots, \Xi'_K$.

Definition 5.1. *A solution* $(\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star_1, \ldots, \mathbf{y}^\star_K)$ *to* (5.4) *is robust if*

$$\forall \boldsymbol{\xi} \in \Xi, \exists k \in \mathcal{K} : \; T(\boldsymbol{\xi})\mathbf{x}^\star + W(\boldsymbol{\xi})\mathbf{y}^\star_k \leq \mathbf{h}(\boldsymbol{\xi}), \; \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y}^\star_k \leq \theta^\star.$$

Example 5.1. *Consider the following main problem*

$$\begin{aligned}
\min_{\theta,\mathbf{x},\mathbf{y}} \quad & \theta \\
s.t. \quad & \theta \in \mathbb{R}, \mathbf{x} \in \{0,1\}^2, \mathbf{y} \in \{0,1\}^2, \\
& \boldsymbol{\xi}^\top \mathbf{x} + 2\boldsymbol{\xi}^\top \mathbf{y}_k \leq \theta, && \forall \boldsymbol{\xi} \in \Xi'_k, \forall k \in \mathscr{K}, \\
& \mathbf{y}_k \geq \mathbf{1} - \boldsymbol{\xi}, && \forall \boldsymbol{\xi} \in \Xi'_k, \forall k \in \mathscr{K}, \\
& \mathbf{x} + \mathbf{y}_k \geq \mathbf{1}, && \forall k \in \mathscr{K},
\end{aligned}$$

where $\boldsymbol{\xi} \in \{-1,0,1\}^2$ *and* $\mathscr{K} = \{1,2\}$*. We look at three solutions for different groupings of* $\boldsymbol{\xi}_1 = [1,1]^\top$*,* $\boldsymbol{\xi}_2 = [0,1]^\top$*, and* $\boldsymbol{\xi}_3 = [0,0]^\top$*. One of them is not robust, and the other ones are but have a different objective value due to the partition. The first partition we consider is* $\Xi'_1 = \{\boldsymbol{\xi}_1\}$*,* $\Xi'_2 = \{\boldsymbol{\xi}_2\}$*. The corresponding solution is* $\theta^\star = 2$*,* $\mathbf{x}^\star = [1,1]^\top$*,* $\mathbf{y}_1^\star = [0,0]^\top$*, and* $\mathbf{y}_2^\star = [1,0]^\top$*. This solution is not robust, since for* $\boldsymbol{\xi}_3 = [0,0]^\top$ *the following constraint is violated for all* $k \in \mathscr{K}$*:*

$$\mathbf{y}_1 \not\geq \mathbf{1} - \boldsymbol{\xi}_3 \iff \begin{bmatrix} 0 \\ 0 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad \mathbf{y}_2 \not\geq \mathbf{1} - \boldsymbol{\xi}_3 \iff \begin{bmatrix} 1 \\ 0 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Next, consider $\Xi'_1 = \{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2\}$*,* $\Xi'_2 = \{\boldsymbol{\xi}_3\}$*, with solution* $\theta^\star = 3$*,* $\mathbf{x}^\star = [0,1]^\top$*,* $\mathbf{y}_1^\star = [1,0]^\top$*,* $\mathbf{y}_2^\star = [1,1]^\top$*. There is no* $\boldsymbol{\xi}$ *that violates this solution, which makes it robust. The third partition is* $\Xi'_1 = \{\boldsymbol{\xi}_1, \boldsymbol{\xi}_3\}$*,* $\Xi'_2 = \{\boldsymbol{\xi}_2\}$*, with solution* $\theta^\star = 4$*,* $\mathbf{x}^\star = [0,0]^\top$*,* $\mathbf{y}_1^\star = [1,1]^\top$*, and* $\mathbf{y}_2^\star = [1,1]^\top$*. This solution is also robust. Their objective values are 3 and 4, which shows that different partitions can significantly influence solution quality.*

Mathematically, we formulate the subproblem using the big-M reformulation:

$$\max_{\zeta,\boldsymbol{\xi},\gamma} \quad \zeta \tag{5.5a}$$

$$\text{s.t.} \quad \zeta \in \mathbb{R}, \quad \boldsymbol{\xi} \in \Xi, \quad \gamma_{kl} \in \{0,1\}, \quad (k,l) \in \mathscr{K} \times \mathscr{L}, \tag{5.5b}$$

$$\sum_{l \in \mathscr{L}} \gamma_{kl} = 1, \qquad \forall k \in \mathscr{K}, \tag{5.5c}$$

$$\zeta + M(\gamma_{k0} - 1) \leq \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y}_k^\star - \theta^\star, \qquad \forall k \in \mathscr{K}, \tag{5.5d}$$

$$\zeta + M(\gamma_{kl} - 1) \leq \boldsymbol{t}_l(\boldsymbol{\xi})^\top \mathbf{x}^\star + \boldsymbol{w}_l(\boldsymbol{\xi})^\top \mathbf{y}_k^\star - h_l(\boldsymbol{\xi}), \qquad \forall l \in \mathscr{L}, \forall k \in \mathscr{K}, \tag{5.5e}$$

where M is some big scalar and $\mathscr{L}$ is the index set of constraints. When $\zeta \leq 0$, we have not found any violating scenario, and the main solution is robust. Otherwise, the found $\boldsymbol{\xi}^\star$ is added to one of $\Xi'_1, \ldots, \Xi'_K$. Then, the main problem is re-solved, so that a new solution $(\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star)$, which is guaranteed to be feasible for $\boldsymbol{\xi}^\star$ as well, is found.

Throughout the process, the key issue is to which of $\Xi'_1, \ldots, \Xi'_K$ to assign $\boldsymbol{\xi}^\star$. As this cannot be determined in advance, all options have to be considered. Figure 5.2 illustrates that from each tree node we create K child nodes, each corresponding to adding $\boldsymbol{\xi}^\star$ to the k-th subset:

$$\tau^k = \{\Xi'_1, \ldots, \Xi'_k \cup \{\boldsymbol{\xi}^\star\}, \ldots, \Xi'_K\}, \quad \forall k \in \mathscr{K},$$

where τ^k is the partition corresponding to the k-th child node of node τ. If a node is found such that $\theta^\star$ is greater than or equal to the value of another robust solution, then

this branch can be pruned as the objective value of the main problem cannot improve if more scenarios are added.

The pseudocode of K-B&B is given in Algorithm 5.1. The search tree becomes very deep for 2RO problems where many scenarios are needed for a robust solution. Moreover, the tree becomes wider when K increases. Due to these issues, solving the problem to optimality becomes computationally intractable in general. Our goal is to investigate if ML can be used to make informed decisions regarding the assignment of newly found $\boldsymbol{\xi}^\star$ to the discrete subsets, so that a smaller search tree has to be explored in a shorter time before a high-quality robust solution is found.

In our implementation of this algorithm, we apply a depth-first search strategy and select a random node of $\mathcal{N}$ instead of the first one. The reasons for these choices are given in the experiments section (Section 5.4.1)

Algorithm 5.1: K-B&B

Input : Problem instance $\mathscr{P}(N)$ with size N, number of partitions K
Output : Objective value θ, first-stage decisions $\mathbf{x}$, second-stage decisions $\mathbf{y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_K\}$, subsets with scenarios Ξ'_k for all $k \in \{1, \ldots, K\}$
Initialization : Incumbent partition: $\tau^i := \{\Xi'_1, \ldots, \Xi'_K\}$, where $\Xi'_k = \emptyset$ for all $k \in \mathcal{K}$, set containing all node partitions yet to explore: $\mathcal{N} := \{\tau^i\}$, incumbent solutions: $(\theta^i, \mathbf{x}^i, \mathbf{y}^i) := (\infty, \emptyset, \emptyset)$

1 **while** $\mathcal{N}$ *not empty* **do**
2 Select the first node with partition $\tau = \{\Xi'_1, \ldots, \Xi'_K\}$ from $\mathcal{N}$, then $\mathcal{N} \leftarrow \mathcal{N} \setminus \{\tau\}$
3 $(\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star) \leftarrow$ *main problem*(τ)
4 **if** $\theta^\star > \theta^i$ **then**
5 Prune tree since current objective is worse than best solution found.
6 Continue to line 2.
7 **end**
8 $(\boldsymbol{\xi}^\star, \zeta^\star) \leftarrow$ *subproblem*$(\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star)$
9 **if** $\zeta^\star > 0$ **then**
10 Solution not robust, create K new branches.
11 **for** $k \in \{1, \ldots, K\}$ **do**
12 $\tau^k := \{\Xi'_1, \ldots, \Xi'_k \cup \{\boldsymbol{\xi}^\star\}, \ldots, \Xi'_K\}$
13 $\mathcal{N} \leftarrow \mathcal{N} \cup \{\tau^k\}$
14 **end**
15 **else**
16 Current solution robust, prune tree.
17 $(\theta^i, \mathbf{x}^i, \mathbf{y}^i, \tau^i) \leftarrow (\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star, \tau)$
18 **end**
19 **end**
20 **return** $(\theta^i, \mathbf{x}^i, \mathbf{y}^i, \tau^i)$

5.3. ML METHODOLOGY

We propose a method that enhances the node selection strategy of the K-B&B algorithm. It relies on four steps:

1. ***Decision on what and how we want to predict:*** Section 5.3.1.
2. ***Feature engineering:*** Section 5.3.2.
3. ***Label construction:*** Section 5.3.3.
4. ***Using partial K-B&B trees for training data generation:*** Section 5.3.4.

All these steps are combined into the K-B&B-NODESELECTION algorithm (Section 5.3.5).

5.3.1. LEARNING SETUP

As there is no clearly well-performing node selection strategy for K-B&B, we cannot simply try to imitate one. Instead, we investigate what choices a good strategy would make. We will focus on learning how to make informed decisions about the order of inspecting the children of a given node. The scope of our approach is illustrated with rounded-square boxes in Figure 5.3.

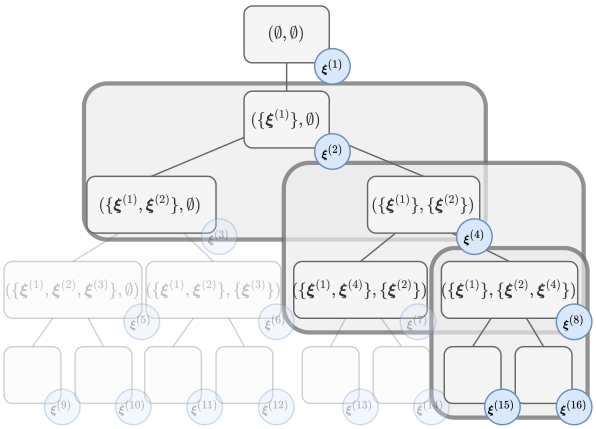


Figure 5.3: The scope of node selection

To rank the child nodes in the order they ideally be explored, one of the ways is to have a certain form of child node information whether selecting this node is *good* or *bad*. In other words, how likely is a node to guide us towards a high-quality robust solution fast. Indeed, this shall be exactly the quantity that we predict with our model. To train such a model, we will construct a dataset consisting of the following input-output pairs:

– **Input.** feature vector F of the decision to insert a scenario to a subset (Section 5.3.2)

– **Output.** $[0,1]$ label that informs how good a given insertion was, based on an ex-post constructed strategy; 'what would have been the best node selection strategy, had we known the entire tree?' (Section 5.3.3).

We gather this data by creating a proxy for an oracle (see Section 5.3.4). Once the predictive model is trained, we can apply it to the search tree where in each iteration we predict for all K child nodes a score μ, in the interval $[0,1]$. The child node with the highest score is explored first (see Figure 5.4). Formally, the working order of the trained

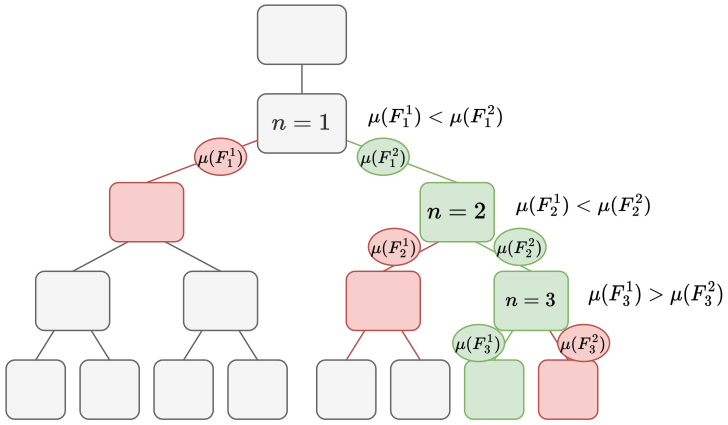


Figure 5.4: Node selection with ML predictions. Node selections are decided by the prediction of the function μ with input features F_n^k, where n is the node from which a selection is made and k relates to its k-th child node.

method shall be as follows:

1. **Process node.** Solve main and subproblem in current node n. If prune conditions hold, then select a new node. Otherwise, continue to Step 2.
2. **Compute features.** For each one of the K child nodes, generate a vector of features $F_n^1, \ldots, F_n^K$ (see Section 5.3.2 for details).
3. **Predict.** For each child node k, predict the goodness score $\mu(F_n^k)$.
4. **Node selection.** Select the child node with the highest score.

It is desirable for an ML tool to be applicable to data that is different from the one which it is trained on. In the context of an ML model constructed to solve optimization problems, this means the potential to use a given trained method on various optimization problems of various sizes, with different values of K. Indeed, we shall demonstrate the generality of our method. First, we note that each parent-child node pair corresponds to one data point, and hence, training on a problem with a certain K does not prevent us from using it for different K. Next, we construct our training dataset so that the model becomes independent of (i) the instance size, and (ii) the type of objective function and constraints. This will be explained in Section 5.3.2.

5.3.2. FEATURE ENGINEERING

If we take a look at a single rounded box in Figure 5.3, the ML model we are about to train is going to give a goodness score which will depend on the parent node and the scenario. Therefore, we need to design features which we group into (i) state features that describe the main problem and the subproblem solved in the parent node, (ii) scenario features that describe the assignment of a newly-found scenarios to one of the subsets Ξ'_k. In what follows, we present the feature list:

1. **State features.** This input describes the parent node n, i.e., the current state of the algorithm. Different states might benefit from different strategies. Hence, information on the current node might increase the prediction performance. This also means that all the child nodes have the same state features: $\mathbf{s}_n^k = \mathbf{s}_n$ for all $k \in \mathscr{K}$. To scale the features, we always initialize a tree search with a so-called *initial run*. This is a dive with random child-node selections, where we stop until robustness is reached. The following values are taken from this initial run:

   - θ^0: the objective function value of the robust solution found in the initial run.
   - ζ^0: the violation of the root node.
   - κ^0: depth reached in the initial run.

   In the experiments, multiple initial runs are done. The averages of θ^0, ζ^0, and κ^0 over the dives are then used for scaling. Additional meta-features for K and the dimensions of Ξ, $\mathscr{X}$, and $\mathscr{Y}$ could be added. We omitted these as in our experimental setup we do not mix training data for different combinations of these values.

2. **Scenario features.** Intuitively, scenarios contained in the same group should have similar characteristics. Therefore, the features for each node are constructed in the following way: each newly found scenario $\boldsymbol{\xi}^\star$ is assigned a set of characteristics, or attributes. Based on the attributes of the new scenario, and the attributes of the scenarios already grouped into the K subsets, we formulate the input of one data point. Some of the scenario attributes can be directly determined from the main problem. Others are extracted from easily solvable optimization problems: the *deterministic problem* and the *static problem*. The deterministic problem is a version of the problem where $\boldsymbol{\xi}^\star$ is the only scenario. The solution to this problem gives information on the optimal objective and decisions found, in the most naive sense. One would expect that for good-performing subset formations, the optimal decisions of its scenarios would have similarities. For the static problem, we first solve for a single $\mathbf{y}$ (no adaptability) for all $\boldsymbol{\xi} \in \Xi$. Then, for the obtained $\mathbf{x}^\star$ and for a given $\boldsymbol{\xi}^\star$, we solve for the best $\mathbf{y}$. The solution to this problem gives additional information as it has a sense of embedded robustness.

As our goal is to use the model also on different problems, we engineer the features to keep them independent from the problem size or type. In Tables 5.1-5.2, we give an outline of the two feature types that we construct. For a detailed description of how they are computed, we refer the reader to Appendix 5.A.

Table 5.1: State features. θ^0, ζ^0, and κ^0 are as defined above. θ^p is the objective value of the parent node, ζ^p is the violation of the parent node, and κ is the depth of the current node.

Num.	State feature name	Description	Calculation
1	Objective	Relative objective value of this node to the first robust solution	θ/θ^0
2	Objective difference	Ratio of objective to that of the parent node	θ/θ^p
3	Violation	Relative violation with respect to the first violation found	ζ/ζ^0
4	Violation difference	Ratio of violation to that of the parent node	ζ/ζ^p
5	Depth	Relative depth of this node to the depth of the fist robust solution	κ/κ^0

Table 5.2: Attributes assigned to scenario $\boldsymbol{\xi}^\star$.

	Num.	Attribute name	Description
	1	Scenario values	Vector of scenario values $\boldsymbol{\xi}^\star$
Main problem	2	Constraint distance	A measure for the change of the feasibility region when $z^\star$ is added to a subset. We look at the distance between the constraints already in the main problem, and the one to be added.
	3	Scenario distance	With this attribute we measure how far away $\boldsymbol{\xi}^\star$ is from *not* being a violating scenario, for each of the k subsets. This is done by looking at the constraints in the space of Ξ, given the current solutions $\mathbf{x}$ and $\mathbf{y}^k$.
	4	Constraint slacks	The slack values of the uncertain constraint per subset decisions.
Deterministic problem	5	Objective value	The objective value of the deterministic problem
	6	First-stage decisions	First-stage decisions of deterministic problem
	7	Second-stage decisions	Second-stage decisions of deterministic problem
Static problem	8	Objective value	The objective value of the static problem
	9	second-stage decisions	Second-stage decisions of static problem

The state features in Table 5.1 are readily problem-independent. However, this is not the case for the scenario attributes in Table 5.2. They are, for example, dependent on the instance size. To deal with the size dependency, and to adopt the actual meaning of node selection (*i.e.*, placing a scenario to a group of other scenarios), we introduce the *attribute distance* as a feature to our model. This is the distance between scenario-subset pairs in terms of the attributes attached to the scenarios, as a proxy to the unknown implications the addition of the scenario would have on the solution of the problem. For each attribute, the scenario-subset distance is taken by the Euclidean distance from the attribute of the new scenario to the average of the attribute of the scenarios already in the subset. This feature is described as:

$$\delta_f^k = \frac{\|\mathbf{a}_f^{k,\boldsymbol{\xi}^\star} - \frac{1}{|\Xi'_k|}\sum_{\boldsymbol{\xi}\in\Xi'_k}\mathbf{a}_f^{k,\boldsymbol{\xi}}\|_2}{\text{length}(\mathbf{a}_f^{k,\boldsymbol{\xi}^\star})}, \qquad \forall k \in \mathcal{K}, \forall f \in \{1,\dots,9\}, \tag{5.6}$$

where δ_f^k is the attribute distance of the new scenario $\boldsymbol{\xi}^\star$ to subset Ξ'_k and $\mathbf{a}_f^{k,\boldsymbol{\xi}}$ is the data vector related to the f-th attribute (of Table 5.2) for the k-th child node. The attribute distance is scaled by the length of the attribute vector, denoted by 'length' in the denominator of (5.6), to control for varying attribute vectors in the feature value. Larger attribute vectors would otherwise result in higher feature values. Then, the *scenario feature* vector of the k-th child node is defined as $\mathbf{d}_n^k = [\delta_1^k,\dots,\delta_9^k]^\intercal$. Then, the input of the ML model for the k-th child of node n is given by $F_n^k = [\mathbf{s}_n \quad d_n^k]^\intercal$.

In Figure 5.5 we outline the feature generation procedure. Steps 2 and 5 indicate how new attributes need to be generated for every child node. In practice many attributes are the same for all subsets, and the attributes that do differ are the main problem-based ones, which are easily computed.

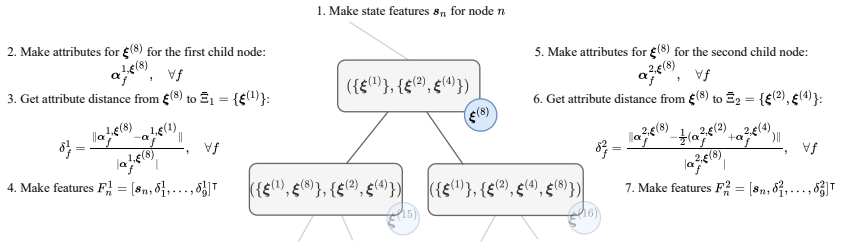


Figure 5.5: Example of a feature generation procedure for node n and its two child nodes.

5.3.3. LABEL CONSTRUCTION

To learn how to assign a new scenario to one of the subsets Ξ'_k, we need another piece of the input-output pairs in our database – labels that would indicate how good, ex post, it was to perform a given assignment, i.e., how likely a given assignment is to lead the search strategy towards a *good solution*. We shall assume that given a K-B&B tree, a good solution is a robust solution with an objective that belongs to the best α% of the

found robust solutions. We will now introduce a notion of scenario-to-set assignments and illustrate a method of constructing labels q.

We define p_v – the probability of node v leading to a good robust solution. If this node is selected, one of a finite, possibly large, number of *leaves* (i.e., terminal nodes) is reached with depth-first search. Then, taking g_v to be the number of good solutions and M_v as the number of leaves under node v, we define $p_v = g_v / M_v$ as the fraction of leaves with a good robust solution. We define a node v as good if the probability p_v of it leading to a good robust solution is higher than some threshold ϵ. This is computed by the quality value $q_v = \mathbf{1}_{\{p_v \geq \epsilon\}}$.

Example 5.2. *Consider the tree in Figure 5.6. For three nodes in the search tree, its subtree (consisting of all its successors) and leaves (coloured nodes) are shown. Red coloured nodes represent bad and green nodes represent good robust solutions. Then, if we pick* $\epsilon = \frac{1}{5}$ *as threshold, the corresponding success probabilities* p_v *and the quality values of the three nodes are:*

$$p_1 = \frac{1}{4},\ p_2 = 0,\ p_3 = \frac{2}{5},\ q_1 = \mathbf{1}_{\{p_1 \geq \frac{1}{5}\}} = 1,\ q_2 = \mathbf{1}_{\{p_2 \geq \frac{1}{5}\}} = 0,\ q_3 = \mathbf{1}_{\{p_3 \geq \frac{1}{5}\}} = 1.$$

The first and third nodes would have been good node selections, whereas the second selection would have been a bad one.

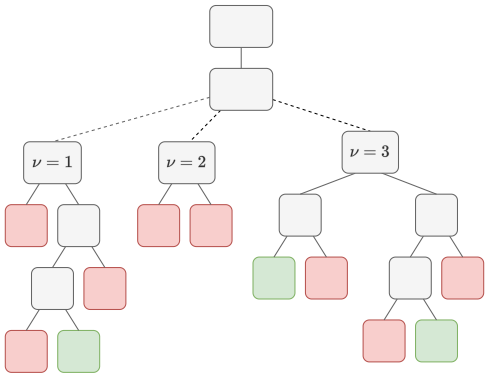


Figure 5.6: Example of a tree where three nodes are considered. The coloured nodes are leaves; green and red nodes are good and bad robust solutions, respectively.

In practice, we do not know for a node v how many underlying leaves are good and bad. This is the reason that in this method we will predict q_v with a model μ. We will also call this model the strategy model (or function) since it guides us in making node selections.

5.3.4. TRAINING DATA GENERATION

Our goal is to learn a supervised ML model $\mu(F_v) = \hat{q}_v$, where μ is the strategy function, F_v the input features, and $\hat{q}_v$ the prediction of the quality of moving to node v. To train this model, we first need to generate training data. Given a single tree, the difficulty of

generating data does not lie in the input, but in the output: an expert is needed to determine the correct values of $\hat{q}_v$ for its nodes. Consider the following; if nodes of the entire tree are *processed* (i.e., solving the main problem and the subproblem), we could easily find the paths from the root to good solutions. Then, all the nodes in these paths would get the values $\hat{q}_v = 1$, and all others $\hat{q}_v = 0$. Or equivalently, the success probabilities would be set to $p_v > \epsilon$. Since the trees grow exponentially, this approach is not practical.

As already mentioned in the introduction, a popular method for exploring intractable decision trees is Monte Carlo tree search (MCTS) [44]. By randomly running deep in the tree, we can gather information on the search space. In our method, we use the idea of random runs to mimic an expert, and hence, label the data points. Generating training data is done as follows (see Figure 5.7):

1. **Get instance.** Generate an instance of a 2RO problem.

2. **Initial run.** One (or multiple) dives are executed to gather feature information.

3. **Initialize search tree.** Process all nodes up to a predetermined level L of the tree. Generate features for each of the explored nodes.

4. **Downward pass.** Per node $v \in \{1,\dots,N_L\}$ of the L-th layer (N_l is the number of nodes of the l-th layer), perform dives for a total of R times.

5. **Probability of bottom nodes.** Set probability $p_{L,v}$ of each node $v \in \{1,\dots,N_L\}$ in layer L as $p_{L,v} = \frac{g_v}{R}$ where $g_v \in \{0,\dots,R\}$ is the number of good solutions from the samples of the v-th node in layer L.

6. **Upward pass.** Propagate the probabilities $p_{l,v}$ upwards through the tree, for all nodes $v \in \{1,\dots,N_l\}$, for all levels $l \in \{L-1,\dots,2\}$, as follows:

$$\begin{aligned} p_{l,v} &= \mathbb{P}(\text{at least one child node is successful}) \\ &= 1 - \mathbb{P}(\text{no successful child nodes}) \\ &= 1 - \prod_{k \in \mathcal{K}} (1 - p^k_{l,v}), \end{aligned}$$

   where $p^k_{l,v}$ is the k-th child of node v of layer l (while this child node is in the $(l+1)$-th level). Note that $p^k_{l,v} = p_{l+1,v'}$ for some $v \in \{1,\dots,N_l\}$ and $v' \in \{1,\dots,N_{l+1}\}$.

7. **Label nodes.** Determine the label $\hat{q}_{l,v}$ for nodes $v \in \{1,\dots,N_l\}$ for levels $l \in \{2,\dots,L\}$.

The advantage of this structure is that both bad and good decisions are well represented in the dataset. However, it only consists of input-output pairs of the top L levels. This is not necessarily a disadvantage as good decisions at start can be expected to be more important. The above generation method is applied per instance, thus it can be parallelized, like Step 4.

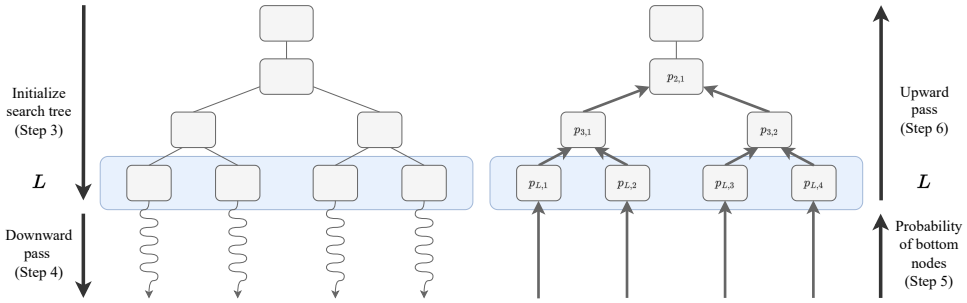


Figure 5.7: Downward and upward pass for generating training data. The blue layer represents the L-th layer in the tree from which random runs are made.

5.3.5. COMPLETE NODE SELECTION ALGORITHM

We now combine all the steps above into one algorithm which is, essentially, a variant of K-B&B enhanced with: (i) node selection, (ii) feature engineering, and (iii) training data generation (see Algorithm 5.2). Our K-B&B-NODESELECTION algorithm has two preprocessing steps:

1. STRATEGYMODEL (Procedure 5.3 in Appendix 5.B): Generate the data applying 1-4, and train the ML model.
2. INITIALRUN (Procedure 5.4 in Appendix 5.B): Start with a random dive through the tree to obtain θ^0, ζ^0, and κ^0 used to scale the features (see Table 5.1).

5.4. EXPERIMENTS

We now investigate if *it is possible to learn a node selection strategy that generalizes (i) to other problem sizes, (ii) to different values of K, and (iii) to various problems.* We answer this question by a detailed study of two problems: capital budgeting (with loans) and shortest path [189], whose formulations are given in Appendix 5.C. This section is set up as follows: First, the effectiveness of the original K-B&B is tested on the problems. Then, we compare K-B&B to K-B&B-NODESELECTION. We shall observe that the results obtained with our approach are very promising.

For solving the MILPs, Gurobi 9.1.1 [92] is used. All computations of generating training data, K-B&B, and K-B&B-NODESELECTION are performed on an Intel Xeon Gold 6130 CPU @ 2.1 GHz with 96 GB RAM. Training of the ML model is executed on an Intel Core i7-10610U CPU @ 1.8 GHz with 16 GB RAM. Our implementation along with the scripts to reproduce our results are available online[1].

[1] https://github.com/estherjulien/KAdaptNS

Algorithm 5.2: K-B&B-NODESELECTION

Input : Test instance $\mathscr{P}^{test}(N^{test})$, train instances $\mathscr{P}_1^{train}(N^{train}),\dots,\mathscr{P}_I^{train}(N^{train})$
number of partitions for training K^{train} testing K^{test},
level for training L^{train} and testing L^{test},
quality threshold ϵ, number of random dives per node R

Output : Objective value θ, first-stage decisions $\mathbf{x}$, second-stage decisions $\mathbf{y} = \{\mathbf{y}_1,\dots,\mathbf{y}_K\}$, subsets with scenarios Ξ'_k for all $k \in \{1,\dots,K\}$

Initialization : Incumbent partition: $\tau^i := \{\Xi'_1,\dots,\Xi'_K\}$, where $\Xi'_k = \emptyset$ for all $k \in \mathscr{K}$,
set containing all node partitions yet to explore: $\mathscr{N} := \{\tau^i\}$,
initial incumbent solution: $(\theta^i, \mathbf{x}^i, \mathbf{y}^i) := (\infty, \emptyset, \emptyset)$

`// Preprocessing`

1 *model* $\leftarrow$ STRATEGYMODEL($\mathscr{P}_1^{train}(N^{train}),\dots,\mathscr{P}_I^{train}(N^{train}), K^{train}, L^{train}, \epsilon, R$)
2 *scaling info* $\leftarrow$ INITIALRUN($\mathscr{P}^{test}(N^{test}), K^{test}$)
`// Tree search`
3 **while** $\mathscr{N}$ *not empty* **do**
4 **if** *no solution yet or previous node pruned* **then**
5 Select a random node with partition $\tau = \{\Xi'_1,\dots,\Xi'_K\}$ from $\mathscr{N}$, then $\mathscr{N} \leftarrow \mathscr{N} \setminus \{\tau\}$
6 **else**
7 $\tau := \{\Xi'_1,\dots,\Xi'_{k^\star} \cup \{\boldsymbol{\xi}^\star\},\dots,\Xi'_K\}$
8 **end**
9 $(\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star) \leftarrow$ *main problem*(τ)
10 **if** $\theta^\star > \theta^i$ **then**
11 Prune tree since current objective is worse than best solution found, continue to line 4.
12 **end**
13 $(\boldsymbol{\xi}^\star, \zeta^\star) \leftarrow$ *subproblem*($\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star$)
14 **if** $\zeta^\star > 0$ **then**
15 Solution not robust. Create K new branches.
16 **if** *current level is more than* L^{test} **then**
17 $k^\star \leftarrow$ *random uniform sample*($[1, K]$)
18 **else**
19 Create feature vectors for K child nodes.
20 $(D_1,\dots,D_K) \leftarrow$ *generate features*(*scaling info*, θ, ζ) `// steps of Section 5.3.2`
21 $k^\star \leftarrow$ *predict node qualities*($D_1,\dots,D_K$, *model*)
22 **end**
23 Make K new branches, of which the $k^\star$-th is selected.
24 **for** $k \in \{1,\dots,K\} \setminus \{k^\star\}$ **do**
25 $\tau^k := \{\Xi'_1,\dots,\Xi'_k \cup \{\boldsymbol{\xi}^\star\},\dots,\Xi'_K\}$
26 $\mathscr{N} \leftarrow \mathscr{N} \cup \{\tau^k\}$
27 **end**
28 **else**
29 Current solution robust, prune tree.
30 $(\theta^i, \mathbf{x}^i, \mathbf{y}^i, \tau^i) \leftarrow (\theta^\star, \mathbf{x}^\star, \mathbf{y}^\star, \tau)$
31 **end**
32 **end**
33 **return** $(\theta^i, \mathbf{x}^i, \mathbf{y}^i, \tau^i)$

5.4.1. PERFORMANCE OF K-B&B

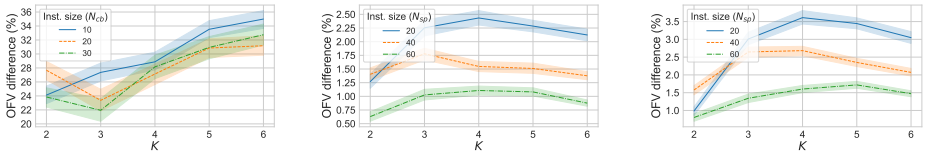
In our experiments, we investigate the potential of improving the node selection strategy with ML. For that reason, it is important to identify problems on which such an improvement matters, i.e., where different partitions give varying outcomes and are nontrivial. To identify such problems, we run K-B&B on several problems. Compared to the algorithm of Subramanyam et al. [189], we made some minor changes in K-B&B:

- Instead of a breadth-first search, depth-first dives are performed with random node selection. Depth-first search returns an incumbent solution early on, which is used for pruning nodes. This is increasingly vital when K grows, as each node branches on K other nodes. Likely a combination of breadth- and depth-first search would be preferred. This would require additional bookkeeping of the features for K-B&B-NODESELECTION.

- In the starting node selection step (Step 2, Algorithm 5.1), a random node is taken from $\mathcal{N}$ instead of the first one. This step is in line with random node selection.

The quantity we are interested in is the relative change in the objective function value (OFV) – the OFV of the first robust solution divided by the best one after 30 minutes. The higher the value, the more potential for a smart node selection strategy.

First, we consider the capital budgeting and the shortest path problems, only mentioned now and formally described later, for which the results are in Figure 5.8. We observed that the objective function values of the capital budgeting instances are changing more than those of shortest path (in which nodes of a graph are located on a 2D plane). This is why we implemented another instance type for shortest path: graph with nodes located on a 3D sphere (see the Appendix). The OFV differences for these instances are still smaller than those of capital budgeting, but more than for the 'normal' type. Therefore, further experiments on shortest path were conducted with the sphere instances alone.



(a) Capital budgeting (b) Shortest path (normal) (c) Shortest path (sphere)

Figure 5.8: Objective function value (OFV) difference (in %) for capital budgeting and shortest path with 'normal' and 'sphere' instances, within 30 minutes using K-B&B. For each problem type, experiments were done for 100 instances per $K \in \{2,\ldots,6\}$ and instance size. The 75% confidence interval (CI) is also given as shaded strips along the curves.

Another problem on which we ran K-B&B was the knapsack problem parameterized by the combination of the capacity (c), the number of items (N_{ks}), and the maximum deviation of the profit of items (i.e., the uncertainty parameter γ). For this problem, we considered instances of a similar size as the earlier problems, fixing $K = 4$ and $N_{ks} = 100$, and tested K-B&B on 16 instances for all combinations of $c \in \{0.05, 0.15, 0.35, 0.5\}$

and $\gamma \in \{0.05, 0.15, 0.35, 0.5\}$. The OFV differences (in percentages) are given in Table 5.3, where it is visible that different partitions barely play a role. Any random robust solution seems to be performing well. Thus, node selection will most likely not enhance K-B&B for knapsack and will therefore not be tested.

Table 5.3: The OFV difference (in %) of the knapsack problem, within 30 minutes, for different values of the capacity (c) and uncertainty (γ) parameters. $K = 4$ and $N = 100$ are fixed.

c	γ			
	0.05	0.15	0.35	0.5
0.05	0.002	0.086	0.000	0.000
0.15	0.002	0.011	0.040	0.000
0.35	0.001	0.001	0.019	0.038
0.5	0.001	0.003	0.012	0.027

Solving problem instances with K-B&B takes a long time, where we have to think in the range of multiple hours until optimality is proven. This also holds for small-size instances. After having studied the convergence over runtime, we have fixed the time limit to 30 minutes for all the problem instances. For capital budgeting with $(N, K) = (10, 2)$ all instances could be solved within 30 minutes. For the same instance size and $K = 3$, 25 out of 100 are solved, only one for $K = 4$, and none for larger K. For the 3D shortest path problem with the smallest instance size and $K = 2$, only six instances are solved.

5.4.2. EXPERIMENTAL SETUP K-B&B-NODESELECTION

In our experiments, we also investigate what would be a good node selection strategy in K-B&B that would perform well after it is trained on a selection of problems and then applied to other problems. Naturally, we are interested in generalization of trained tools to different instance sizes, K, and problem types. To this end, we have performed an ablation-study described in Table 5.4, where each row informs about the similarity of the testing problem instances, compared to the training ones.

Table 5.4: Different types of experiments (EXP), where the instance size N, K, and the problem itself can be different for training and testing.

Type	**Instance size** N	K	**Problem**
EXP1	Same	Same	Same
EXP2	Same	Different	Same
EXP3	Different	Same	Same
EXP4	Different	Different	Same
EXP5	Different	Same	Different
EXP6	Different	Different	Different

The problems we study are the capital budgeting problem with loans and the shortest path problem on a sphere (see the Appendix). We now describe the design choices we made regarding the ML model and data generation. As our focus lies in how ML is used for optimization and not in differences between ML models, we select a frequently-used

model: random forest of `scikit-learn` [166] with default settings. We note that the training times do not exceed a couple of minutes for different data sets.

As for the data generation process, it is governed by STRATEGYMODEL (Procedure 5.3), driven by the following parameters: I (number of training instances), L^{train} (depth level used in training data), and R (number of dives per node). In these experiments, we instead made them depend on T – the total duration in hours, and ι – the time per training instance in minutes. First, we set $I = 60T/\iota$. Selection of the right L^{train} value is more challenging since for some problem instances the main problem takes a lot more time or deep trees are needed. Therefore, to get sufficiently many random dives for each node in L^{train} within the time limit ι, we make L^{train} depend negatively on the total duration of the initial run (INITIALRUN, Procedure 5.4). This means that if a random dive takes a long time, the number of starting nodes should be lower, and therefore, the value of L^{train} should also decrease. Finally, the number of dives per node (R) depends on how many nodes we have in level L^{train} and the time we have for the instance (ι).

First, the experiments of EXP1-EXP4 are run on the capital budgeting and shortest path problem. Then, the experiments where the training and testing problems are mixed, are conducted (EXP5 and EXP6). We only discuss a representative selection of the results in the main body, referring the reader to Appendix 5.E for a complete overview. Finally, we discuss the feature importance scores we obtained with our random forests.

5.4.3. CAPITAL BUDGETING

The capital budgeting problem is a 2RO problem where investments in a total of N_{cb} projects can be made in two time periods. In the first period, the cost and revenue of these projects are uncertain. In the second period, these values are known but an extra penalty needs to be paid for postponement. The MILP formulation of capital budgeting has uncertain objective function, fixed number of uncertain constraints, and the dimension of $\mathcal{Z}$ is fixed to 4 for all instance sizes. For the full description, see Appendix 5.C.1. Recall that K-B&B-NODESELECTION takes more parameters than the ones being tested in the ablation study: L^{test} (level up to where node selection is performed), ϵ (node quality threshold), and T and ι for generating training data. These parameters will be tuned in the first part of the experiments.

PARAMETER TUNING (WITH EXP1)

For this type of experiment, we use the same K and N for testing and training, applied to the smallest instance size: $N_{cb} = 10$, but for all $K \in \{2,3,4,5,6\}$. For different values of K, we noticed that different hyperparameters for generating training data were performing well. In Appendix 5.D.1, the tuning is performed of parameter ι (minutes spent per training instance) based on the number of data points obtained in total, together with some other information. The testing accuracy scores for different data sets we trained on, range between 0.92-0.99.

We next tune the values of L^{test} (level up to which node selection is performed) and ϵ (quality threshold) using the sets $L^{test} \in \{5,10,20,30,40,50\}$ and $\epsilon \in \{0.05,0.1,0.2,0.3,0.4\}$. Since there are 30 combinations per K value, we only considered $K = 6$. The results are shown in Figure 5.9, where using a low value for ϵ and high L^{test} gives the best results for both values of K. When a node n has success probability p_n larger than zero, this is

already considered a good quality node. For further experiments of capital budgeting, we fix $\epsilon = 0.05$. Since high values of L^{test} outperform lower ones, we also consider the possibility of applying the strategy always ($L^{test} = \infty$), with fixed $\epsilon = 0.05$. This option is analyzed in the Appendix. We noticed that choosing $L^{test} = \infty$ performs well for $K = 6$ but for lower values of K, choosing $L^{test} = 40$ gives even better solutions. Therefore, we continue with $L^{test} = 40$.

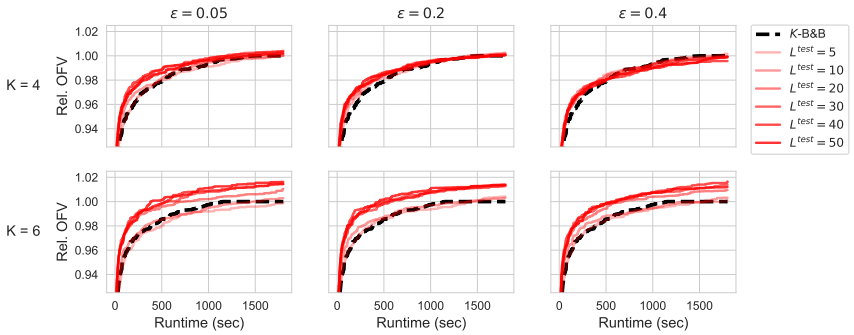


Figure 5.9: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K, L^{test} and ϵ. The plots show the average relative objective value over the runtime of 100 instances for the capital budgeting problem, for $N = 10$.

Another important parameter is the number of hours spent on generating training data (T). In Figure 5.10, results are shown for $T \in \{1,2,5,10\}$ and $K \in \{3,4,5,6\}$. Note that only for $K = 3$ higher values of T result in a better performance of K-B&B-NODESELECTION. For the other values of K the performance is similar, or even worse, for higher values of T compared to lower ones. For further experiments, we shall use $T = 2$.

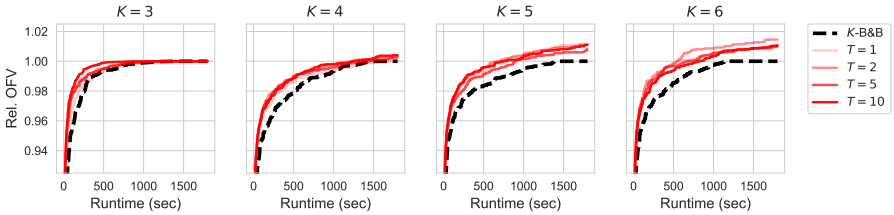


Figure 5.10: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K and T. The plots show the average objective over THE runtime of 100 instances for the capital budgeting problem, for $N = 10$.

ALTERNATIVE SCALING

The step taken to obtain scales for the objective (θ^0), violation (ζ^0), and depth (κ^0) in INITIALRUN seems to be avoidable if we decide to take other values for these parameters. In this section, we will describe a small experimental study of different scaling methods:

(i) INITIALRUN, (ii) Alternative, and (iii) No scaling, where the latter corresponds to setting $\theta^0 = \zeta^0 = \kappa^0 = 1$. The Alternative setting takes as scale for θ^0 the objective of the root node, and $\kappa^0 = K^{\log(\dim(\mathcal{Z}))}$. Next to these 'internal' scaling approaches, one can also use min/max scaling of all feature values. The scales are taken from the training data set and then used in the testing phase. These two scaling approaches can be used in combination. The results obtained for the capital budgeting problem, with $N = 10$ and $K^{test} = K^{train} = 6$, are given in Table 5.5. Here, for the first time, we use performance statistics to measure the effectiveness of the method. From the results presented in the table, we can conclude that, on average, solutions found after 30 minutes are best when INITIALRUN is used. Additional min/max scaling on top of the different scaling approaches gives mixed results.

Table 5.5: The results for a variety of scaling methods tested on capital budgeting problem, with $N = 10$, $K^{test} = K^{train} = 6$, on 16 instances. Different 'internal' scaling methods: 'INITIALRUN', 'Alternative', and 'No scaling'. These internal scaling methods can be combined with min/max scaling. The results for all combinations is given. Four performance statistics are given: (i) 'OFV 30m': Difference (in %) in relative OFV found after 30 minutes from K-B&B to K-B&B-NODESELECTION (higher is better). (ii) 'OFV 1m': is the relative OFV from the two methods after one minute. (iii): 'to OFV=1': Runtime speedup (in %) to reach a relative OFV of 1 for K-B&B-NODESELECTION compared to K-B&B (higher is better). (iv) 'NS to OFV=1': Number of instances of K-B&B-NODESELECTION that reached a relative OFV of 1 (higher is better).

Statistic	**min/max**	**Initial dive**		**Alternative**		**No scaling**	
		NO	YES	NO	YES	NO	YES
OFV 30m		2.10	1.98	1.28	1.11	1.33	1.50
OFV 1m		2.14	2.39	1.49	1.93	2.03	1.39
to OFV=1		44.10	42.53	52.58	38.08	51.72	44.29
NS to OFV=1		16	16	13	14	13	14

RESULTS

We compare K-B&B to K-B&B-NODESELECTION for all combinations of $K^{test}, K^{train} \in \{2,3,4,5,6\}$, and $N_{test} \in \{10,20,30\}$. The results of the full set of combinations are displayed in Appendix 5.E.1. Table 5.6 lists the results using four performance statistics of EXP1 and EXP2 (described in the caption). The table shows that K-B&B-NODESELECTION outperforms K-B&B, with speedups ranging from on average 2 to 50%, except for $K^{train} = 2$ and the instances of $K^{test} = 2$. Higher values of K^{train} generally perform better both in terms of relative OFV and speedup.
In the next sections, we look at specific problem instances in more detail, where we also focus on the stability of the algorithm. Due to the many combinations for EXP3-EXP4, a table would be too convoluted.

EXP1 and EXP2 results. For EXP1, the results for $K^{train} = K^{test} \in \{3,4,5\}$ are shown in Figure 5.11. As in Figure 5.8, the shaded strips around the solid curves show the confidence interval. We can see that K-B&B-NODESELECTION outperforms K-B&B for all K. Moreover, the convergence is steeper: good solutions are found earlier. For $K \in \{4,5\}$ the final solution is also better when node selection is guided by ML predictions. Then for EXP2, where we also apply ML models that are trained with $K^{train} \neq K^{test}$, we see the

Table 5.6: Combined results of EXP1 (along diagonal) and EXP2 for the capital budgeting problem. The four statistics described in the caption of Table 5.5 are given: (i) 'OFV 30m' (higher is better), (ii) 'OFV 1m', (iii) 'to OFV=1' (higher is better), and (iv) 'NS to OFV=1' (higher is better).

K^{test}	**Statistic**	K^{train} 2	3	4	5	6
2	OFV 30m	-0.00	-0.00	-0.00	-0.00	-0.00
	OFV 1m	-0.13	-0.12	-0.09	0.02	-0.09
	to OFV=1	-101.47	-78.31	-60.72	-54.80	-33.46
	NS to OFV=1	47	46	56	50	55
3	OFV 30m	-0.08	-0.13	0.05	-0.06	0.12
	OFV 1m	0.56	2.07	2.38	2.31	2.74
	to OFV=1	3.13	18.67	26.95	12.00	24.38
	NS to OFV=1	47	44	49	56	64
4	OFV 30m	-0.47	-0.02	0.34	0.66	0.95
	OFV 1m	-0.19	1.58	1.98	2.22	2.33
	to OFV=1	21.19	35.19	43.33	45.80	49.88
	NS to OFV=1	33	45	62	74	78
5	OFV 30m	-0.08	0.43	0.66	1.15	1.55
	OFV 1m	-0.70	1.36	1.34	1.38	1.79
	to OFV=1	-0.57	35.43	40.61	48.15	52.80
	NS to OFV=1	53	66	69	79	88
6	OFV 30m	-0.02	0.74	0.88	1.18	1.74
	OFV 1m	0.07	1.49	1.68	1.68	2.33
	to OFV=1	13.50	30.99	24.11	27.75	31.75
	NS to OFV=1	49	69	76	78	89

performance of K-B&B-NODESELECTION improving when K^{train} increases. This indicates that data obtained with higher values of K^{train} are more informative than those with lower values. See Figure 5.12 for an illustration with $K^{test} = 5$ and $K^{train} \in \{4,5,6\}$.

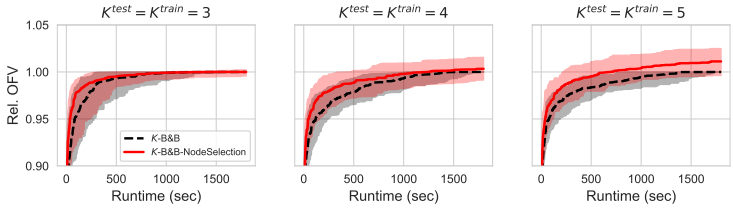


Figure 5.11: EXP1 results for $K^{train} = K^{test} \in \{3,4,5\}$ on 100 instances. The black line gives the average relative objective function value (Rel. OFV) over the runtime (in seconds) of K-B&B, with a 30 minute time limit. The red line is the Rel. OFV trajectory of K-B&B-NODESELECTION. The shaded area around the lines is their respective 75% confidence interval.

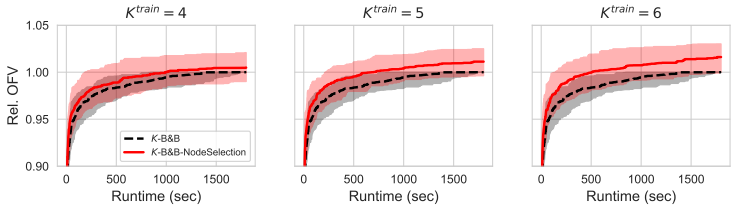


Figure 5.12: EXP2 results for $K^{test} = 5$ and $K^{train} \in \{4,5,6\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

EXP3 and EXP4 results. In Figure 5.13, we depict the results for experiments with same K but different instance sizes. On average, K-B&B-NODESELECTION performs better both in terms of speedup and final relative OFV found: for $N^{test} = 20$, the speedups are between 20-50% and relative OFV increase around 1.5%. For $N^{test} = 30$, we see speedups ranging from 11-47% and relative OFV increases of 1.6-2.1%. However, the confidence interval of $N^{test} = 30$ is larger than that of $N^{test} = 10$. This suggests that testing on higher values of N gives rise to a higher risk. Testing on higher instance sizes for different values of K (Figure 5.14) has a similar effect as before: higher values of K^{train} result in better performance (Rel. OFV ranging from 1.4-1.9%, and speedup 5-27%), although marginally for some parameter combinations.

5.4.4. SHORTEST PATH ON A SPHERE

The shortest path problem can be described as a 2RO problem where we make the planning of a route from source to target, for which the lengths of the N_{sp} arcs are uncertain. This problem only has second-stage decisions and an uncertain objective. The dimension of the uncertainty set grows with the number of arcs in the graph. For the full description, see Appendix 5.C.2. Since there is only a second stage, some attributes

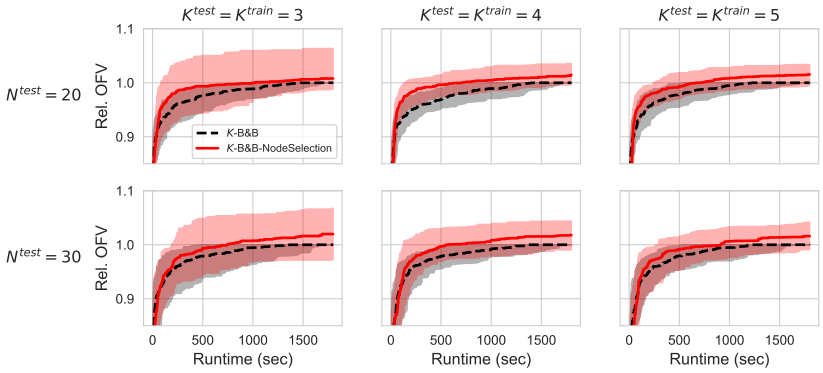


Figure 5.13: EXP3 results for $K^{train} = K^{test} \in \{3,4,5\}$ and $N^{test} \in \{20,30\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

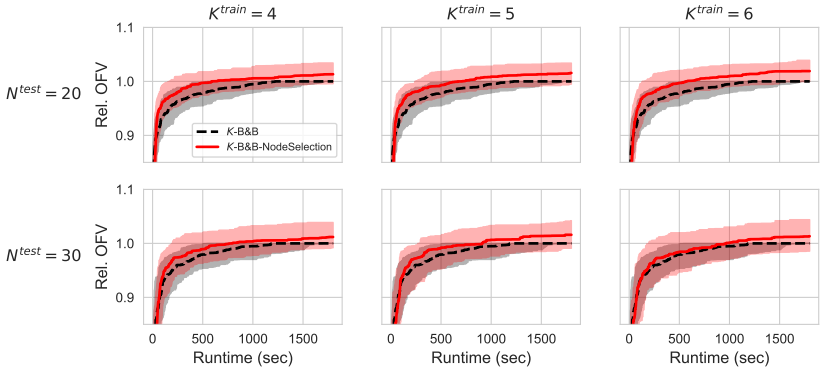


Figure 5.14: EXP4 results for $K^{test} = 5$, $K^{train} \in \{4,5,6\}$, and $N^{test} \in \{20,30\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

disappear for this problem: 'Deterministic first-stage' (attribute 6), and the static objective problem-related attributes (8 and 9). Hence, we are left with six attributes for this problem.

PARAMETER TUNING (WITH EXP1)

As we did for the capital budgeting problem, we first tune the parameter ι. The details of this parameter tuning are given in Appendix 5.D.2. The testing accuracy scores for different data sets we trained on is lower than for capital budgeting; between 0.88-0.97. Since the distribution of the probability success values p_n is very similar to the capital budgeting problem, the quality threshold is set to $\epsilon = 0.05$. In Figure 5.15, the level is tested again with $L^{test} \in \{5, 10, 20, 40, \infty\}$. Here we see that especially for smaller values of K, a higher level of L^{test} outperforms others. Therefore, we select $L^{test} = \infty$ for the remainder of the experiments.

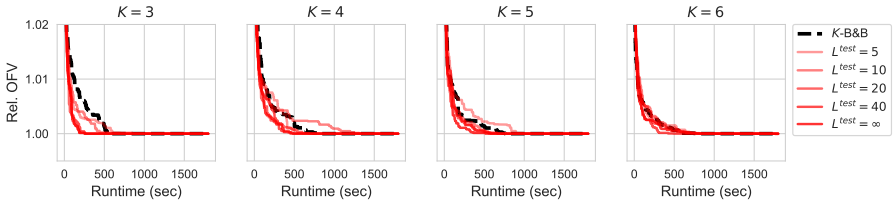


Figure 5.15: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K and L^{test}. The plots show the average objective over runtime of 100 instances for the shortest path problem, for $N = 20$.

Also note that when K grows, the performances of K-B&B and K-B&B-NODESELECTION are very similar. For the shortest path problem, we noticed that more training data points led to a substantial performance gain. See Figure 5.16 for these results. Therefore, for EXP1-EXP4, we select T for each K separately. For an overview of which parameters are chosen per K; see Appendix 5.D.2.

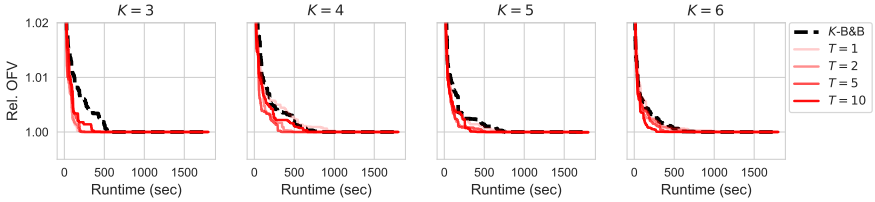


Figure 5.16: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K and T. The plots show the average objective over a runtime of 100 instances for the shortest path problem, for $N = 20$.

RESULTS

The entirety of the results for all combinations of K^{test}, K^{train}, and N^{test} for K-B&B versus K-B&B-NODESELECTION can be found in Appendix 5.E.2. Again, some performance statistics for EXP1-EXP2 are given, see Table 5.7. This table shows that the relative OFV found by K-B&B-NODESELECTION is marginally better than those of K-B&B

(0.01-0.12% when excluding $K^{train} = 2$), but that the speedups can be very significant: for $K = 6$, the average speedups range from 53-94%. For this problem class, the instances of $K = 2$ seem to benefit from ML enhancements.

Table 5.7: Combined results of EXP1 (along diagonal) and EXP2 for the shortest path problem. The four statistics described in the caption of Table 5.5 are given: (i) 'OFV 30m' (higher is better), (ii) 'OFV 1m', (iii) 'to OFV=1' (higher is better), and (iv) 'NS to OFV=1' (higher is better).

K^{test}	Statistic	K^{train} 2	3	4	5	6
2	OFV 30m	0.05	0.07	0.03	0.03	0.08
	OFV 1m	0.08	1.10	1.08	1.04	0.10
	to OFV=1	36.63	15.83	24.77	22.31	12.31
	NS to OFV=1	98	100	99	100	97
3	OFV 30m	0.01	0.14	0.13	0.11	0.12
	OFV 1m	0.17	0.45	0.51	0.19	0.52
	to OFV=1	60.44	46.93	34.30	49.02	29.06
	NS to OFV=1	99	100	99	100	100
4	OFV 30m	-0.12	0.10	0.12	0.11	0.12
	OFV 1m	0.89	0.41	0.41	0.47	0.65
	to OFV=1	55.90	63.34	75.11	45.12	45.79
	NS to OFV=1	99	100	100	100	100
5	OFV 30m	-0.07	0.12	0.09	0.15	0.10
	OFV 1m	-0.33	0.21	0.16	0.23	0.37
	to OFV=1	62.87	81.84	77.14	64.61	52.25
	NS to OFV=1	100	100	100.00	100	100
6	OFV 30m	-0.07	0.04	0.01	0.10	0.07
	OFV 1m	-0.07	0.11	0.17	0.20	0.23
	to OFV=1	52.26	67.37	67.55	88.17	93.68
	NS to OFV=1	100	100	100	100	100

EXP1 and EXP2 results. As is visible from Figure 5.17, even though no better solutions are found (probably because the optimal solution is found early on), it is still noticeable that K-B&B-NODESELECTION converges faster than K-B&B. This phenomenon is thus consistent over the two problems. The convergence of K-B&B over different values of K^{test} differs significantly (e.g., compare K^{test} equal to 3 and 5). The convergence of K-B&B-NODESELECTION is however quite stable across different values of K^{test}. Then, for EXP2, in Figure 5.18, we see that for $K^{test} = 4$, $K^{train} = 6$ performs best. For other values of K^{test}, $K^{train} = 6$ behaves well too, just as it did for the capital budgeting problem.

EXP3 and EXP4 results. We see in Figure 5.19 that the two algorithms behave very similarly. The average relative OFV ranges from 0.14-0.55%. Interestingly, as not easily visible from the figure, the average speedup values are quite high: ranging from 21-98%. The speedup of 98% is achieved for $K = 6$ and $N = 40$, where the line of K-B&B consistently lies slightly above the one of K-B&B-NODESELECTION. In terms of stability, the confidence interval of K-B&B-NODESELECTION grows with N^{test}. However, this does not necessarily mean that testing and training on different sizes is not stable: the CI of

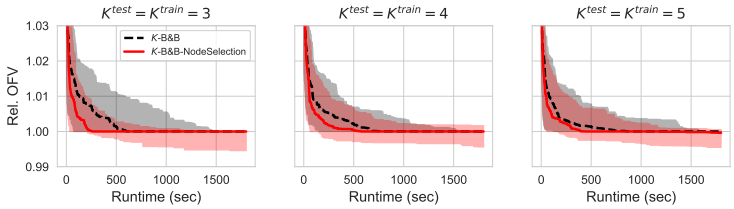


Figure 5.17: EXP1 results for $K^{train} = K^{test} \in \{3,4,5\}$ on 100 instances. The black line gives the average relative objective function value (Rel. OFV) over the runtime (in seconds) of K-B&B, with a 30 minute time limit. The red line is the Rel. OFV trajectory of K-B&B-NODESELECTION. The shaded area around the lines is their respective 80% confidence interval.

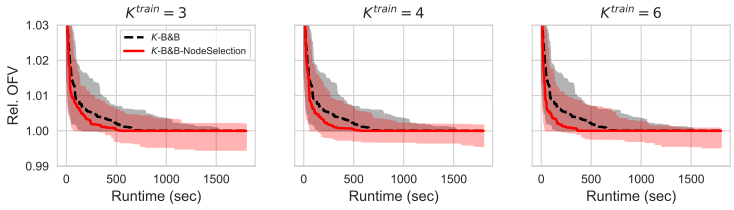


Figure 5.18: EXP2 results for $K^{test} = 4$ and $K^{train} \in \{3,4,6\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

K-B&B is also bigger. Moreover, the CI is mostly in the region below one, which indicates that we mainly have well-performing outliers. In Figure 5.20, the results of EXP4 are shown, where the average relative OFV improvement ranges from 0.29-0.51% and the speedups ($K = 4$ excluded) are between 32-77%. We see that for $N^{test} = 40$, $K^{train} = 6$ performs best, but not necessarily for the biggest instance size.

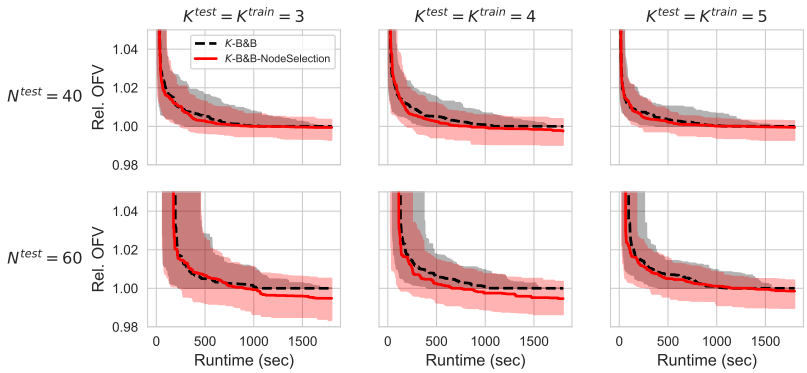


Figure 5.19: EXP3 results for $K^{train} = K^{test} \in \{3,4,5\}$ and $N^{test} \in \{40,60\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

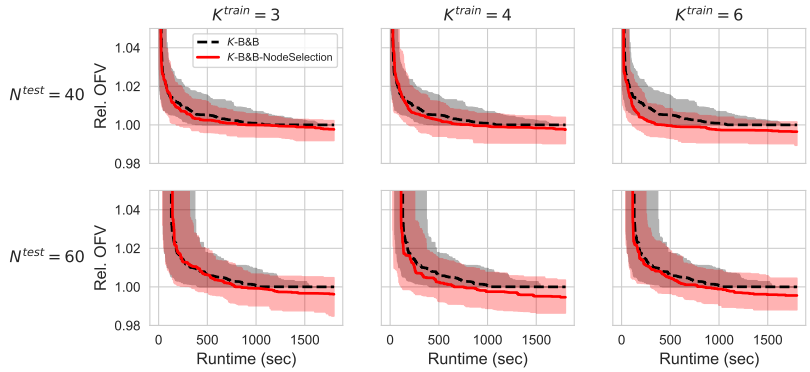


Figure 5.20: EXP4 results for $K^{test} = 4$, $K^{train} \in \{3,4,6\}$, and $N^{test} \in \{40,60\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

5

5.4.5. TRAINING AND TESTING ON DIFFERENT PROBLEMS

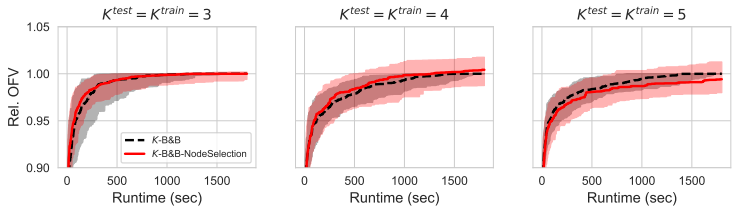
In this section, we show the results of EXP5 and EXP6, where we apply the node selection strategy to a different problem than it has been trained on. Note that the shortest path problem does not have first-stage decisions. This results in the features being slightly different than for the capital budgeting problem. Therefore, to create a model that can be trained by shortest path data, and used for the capital budgeting problem, the first-stage-related attributes are not constructed while running K-B&B-NODESELECTION. Full results of these experiments are given in Appendix 5.E.3.

Figure 5.21a shows the results of EXP5 for the capital budgeting problem. We can see that the performances of the two algorithms are very close. Recall that $K^{train} = 6$ previously resulted in (one of) the best solutions. Then, when we look at some of the solutions of EXP6 with $K^{train} = 6$ (see Figure 5.21b), we see this is not the case here.

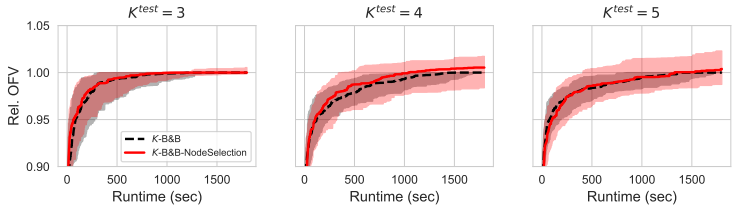
Now we show the results of the shortest path problem that uses a ML model trained on capital budgeting data. Due to the mismatch of features, we delete the three first-stage-related features not used by shortest path from the capital budgeting data. We can still use the generated capital budgeting data. For an illustration of EXP5, see Figure 5.22a. These plots illustrate that for two out of three values of K^{test}, K-B&B-NODESELECTION outperforms K-B&B, even though it is trained on data of another problem. More interesting is the following: the performance of different values of K^{test} with $K^{train} = 6$ gives very good results. They are as good as the ones trained on shortest path data. For an illustration of this, see Figure 5.22b.

5.4.6. FEATURE IMPORTANCE

A trained random forest model allows us to compute feature importance scores. For both problems tested on, these scores are given in Table 5.8 for $K^{train} \in \{2,3,4,5,6\}$. We observe that the relatively highest importance corresponds to the 'Objective' feature (a state feature), with the importance of the other features (both state and scenario ones), being of similar magnitude. Because of that, we cannot draw significant conclusions about the relative importance of the remaining features.

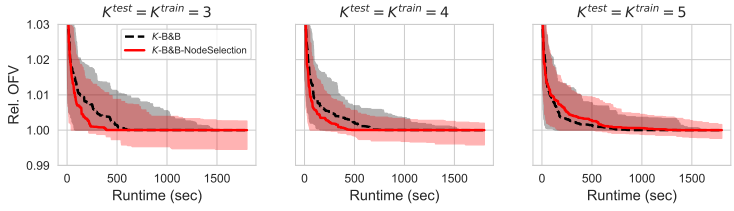


(a) EXP5

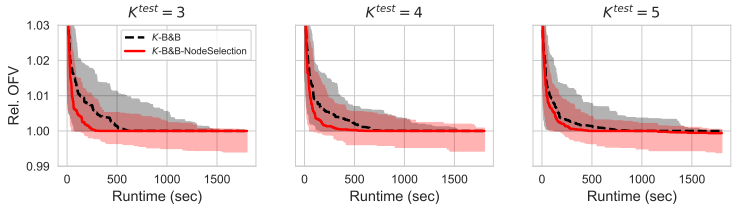


(b) EXP6

Figure 5.21: EXP5 results of capital budgeting for $K^{test} = K^{train} \in \{3,4,5\}$ and EXP6 results for $K^{train} = 6$ and $K^{test} \in \{3,4,5\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.



(a) EXP5



(b) EXP6

Figure 5.22: EXP5 results of shortest path for $K^{test} = K^{train} \in \{3,4,5\}$ and EXP6 results for $K^{train} = 6$ and $K^{test} \in \{3,4,5\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

Table 5.8: Feature importance scores of the selected random forest models for the two problems capital budgeting and shortest path, for each K^{train}. The bold scores are given to the scores that belong to the two highest ones of that model. The dashes correspond to the omitted features for the shortest path problem.

Feature name	Capital budgeting					Shortest path				
K^{train}	2	3	4	5	6	2	3	4	5	6
Objective	**0.25**	**0.25**	**0.24**	**0.28**	**0.25**	**0.10**	**0.11**	**0.24**	**0.13**	**0.15**
Objective difference	0.05	0.04	0.04	0.03	0.04	0.06	0.05	0.04	0.05	0.05
Violation	**0.12**	**0.10**	0.07	0.07	0.06	0.09	0.09	0.10	0.10	0.10
Violation difference	0.05	0.06	0.05	0.05	0.05	0.09	0.09	0.08	0.10	0.09
Depth	0.05	0.08	0.07	0.04	0.03	**0.12**	**0.13**	**0.12**	**0.11**	**0.10**
Scenario values	0.06	0.06	0.05	0.05	0.06	0.09	**0.11**	0.08	0.10	0.09
Constraint distance	0.08	0.06	0.05	0.05	0.05	0.10	0.10	0.08	0.10	**0.10**
Scenario distance	0.06	0.06	0.07	0.07	0.05	0.09	0.10	0.07	0.09	0.09
Constraint slacks	0.07	0.06	0.05	0.04	0.04	0.09	0.06	0.05	0.07	0.07
Det. Objective value	0.05	0.06	**0.11**	**0.12**	**0.12**	0.09	0.09	0.08	0.09	0.09
Det. First-stage decisions	0.05	0.06	0.05	0.06	0.07	-	-	-	-	-
Det. Second-stage decisions	0.00	0.00	0.00	0.00	0.00	0.08	0.07	0.06	0.06	0.06
Stat. Objective value	0.06	0.06	0.08	0.10	**0.12**	-	-	-	-	-
Stat. Second-stage decisions	0.06	0.05	0.05	0.05	0.05	-	-	-	-	-

5

5.5. CONCLUSION AND FUTURE WORK

We introduce an ML-based method to improve the K-B&B algorithm [189] for solving 2RO problems. K-B&B uses a search tree to optimally partition the uncertainty set into K parts and we propose to use a supervised ML model that learns the best node selection strategy to explore such trees faster.

For this, we designed a procedure for generating training data and formulated the ML features based on our knowledge of 2RO so that they are independent of the size, the value of K, and the type of problems on which the ML tool is trained. We experimentally show that our method outperforms K-B&B on the problems we test on. We see that when a problem is trained on a smaller instance size, and then applied to the same problem type with bigger instances, our method still outperforms K-B&B, although being less stable. Training and testing on entirely different problem types resulted in mixed results.

As K-B&B has a tree search structure, we believe that our work can be used to tackle other problems solved by a similar tree search structure, wherever expert knowledge can be used to construct meaningful problem size-independent features.

APPENDIX OF CHAPTER 5

5.A. ATTRIBUTE DESCRIPTIONS

Each scenario gets its own set of attributes. In total there are nine types: one is the scenario vector $\boldsymbol{\xi}$ itself ($\mathbf{a}_1^{\boldsymbol{\xi}} = \boldsymbol{\xi}$), three are determined by the solutions of the main problem, three are extracted from solving the *deterministic problem*, and two are taken from solving the *static problem*.

Main problem-based attributes. For these attributes, only little additional computation is needed. This is due to the fact that the values we use can be taken from the current solution of the main problem. Attributes 2-4 in Table 5.2 share this property:

2. **Constraint distance:** For the first attribute of the 'main-problem' type, we look at the constraints generated by the new scenario $\boldsymbol{\xi}^\star$. We call these constraints the '$\boldsymbol{\xi}^\star$-constraints'. When a constraint is added to a problem, the resulting feasible region will always be as large as, or smaller than, the feasible region we had before. When the feasible region is large, the objective value we find is often better than for smaller feasible regions (the optimal value found in the large region may be cut off in the small region). Each subset $\Xi'_1, \ldots, \Xi'_K$ consists of its own set of scenarios, which translates to its own set of constraints in the main problem. These are the 'Ξ'_k-constraints', for all $k \in \mathscr{K}$. These Ξ'_k-constraints form a feasibility region for the decision pair $(\mathbf{x}, \mathbf{y}_k)$. Ideally, we would calculate the volume of the feasible region whenever the $\boldsymbol{\xi}^\star$-constraints are added to the existing feasible regions. However, obtaining this result is computationally intractable. Therefore we only look at the distance between the constraints. We generate the cosine similarity between the $\boldsymbol{\xi}^\star$-constraints and the Ξ'_k-constraints, for each subset separately. When the cosine similarity is high, the distance between the constraints is low. Then, the attribute of the k-th child node $\mathbf{a}_2^{k,\boldsymbol{\xi}^\star}$ is the cosine similarity of the $\boldsymbol{\xi}^\star$-constraints and the Ξ'_k-constraints:

$$a_{2,c}^{k,\boldsymbol{\xi}^\star} = \max_{\boldsymbol{\xi} \in \Xi'_k} \frac{\gamma_c(\boldsymbol{\xi}^\star) \cdot \gamma_c(\boldsymbol{\xi})}{\|\gamma_c(\boldsymbol{\xi}^\star)\| \|\gamma_c(\boldsymbol{\xi})\|}, \qquad \forall c \in \{1, \ldots, C\}, \quad \forall k \in \mathscr{K},$$

where $a_{2,c}^{k,\boldsymbol{\xi}^\star} \in [-1, 1]$ and $\gamma_c : \Xi \to \mathscr{X} \times \mathscr{Y}$ is a function of the left-hand-side of constraint $c \in \{1, \ldots, C\}$ with input scenario $\boldsymbol{\xi}$. The first- and second-stage decisions are variable. Finally, the attribute of the k-th child node is formulated as $\mathbf{a}_2^{k,\boldsymbol{\xi}^\star} = [a_{2,1}^{k,\boldsymbol{\xi}^\star}, \ldots, a_{2,C}^{k,\boldsymbol{\xi}^\star}]$. Thus, the length of $\mathbf{a}_2^{k,\boldsymbol{\xi}^\star}$ is equal to the number of uncertain constraints of the MILP formulation of the problem.

3. **Scenario distance:** Per subset, we wish to know how far away the new scenario $\boldsymbol{\xi}^\star$ is from *not* being a violating scenario. We suspect that if the distance is small rather

than large, the current solution will not change much. Thus, will not become much worse. This attribute first takes the current solutions of the main problem. Then, for the k-th subset it determines the distance between each of the following planes (boundaries of constraints of) and the new scenario $\boldsymbol{\xi}^\star \in \Xi$:

$$\mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y}_k^\star - \theta^\star = 0, \tag{5.7a}$$
$$T_c(\boldsymbol{\xi})\mathbf{x}^\star + W_c(\boldsymbol{\xi})\mathbf{y}_k^\star - h_c(\boldsymbol{\xi}) = 0, \qquad \forall c \in \{2,\dots,C\}. \tag{5.7b}$$

Hence, we need to determine the distance between a plane and a point, for each constraint $c \in \{1,\dots,C\}$, where $c = 1$ corresponds to the objective function. The point-to-plane distance of the c-th plane for subset k is calculated by projecting $\boldsymbol{\xi}^\star$ on the normal vector of the plane as follows:

$$\chi_c^k = \frac{|\boldsymbol{\rho}_c(\mathbf{x}^\star, \mathbf{y}^k)^\top \boldsymbol{\xi}^\star|}{\|\boldsymbol{\rho}_c(\mathbf{x}^\star, \mathbf{y}^k)\|},$$

where $\boldsymbol{\rho}_c : \mathscr{X} \times \mathscr{Y} \to \Xi$ is a vector of coefficients of constraint c of (5.7). To compare the point-to-plane distance of the K subsets, we scale over the sum of the distance of the subsets. Then, the attribute is given as:

$$a_{3,c}^{k,\boldsymbol{\xi}^\star} = \frac{\chi_c^k}{\sum_{k' \in \mathscr{K}} \chi_c^{k'}}, \quad \forall c \in \{1,\dots,C\}, \quad \forall k \in \mathscr{K}.$$

4. **Constraint slacks:** This attribute takes the slack values of the uncertain constraints of the main problem for the new scenario $\boldsymbol{\xi}^\star$ with fixed first- and second-stage decisions $\mathbf{x}^\star$ and $\mathbf{y}^\star$. For the k-th subset and the constraints $c \in \{1,\dots,C\}$, with $c = 1$ the objective constraint, we get:

$$s_1^k = |\mathbf{c}(\boldsymbol{\xi}^\star)^\top \mathbf{x}^\star + \mathbf{d}(\boldsymbol{\xi}^\star)^\top \mathbf{y}_k^\star - \theta^\star|,$$
$$s_c^k = |T_c(\boldsymbol{\xi}^\star)\mathbf{x}^\star + W_c(\boldsymbol{\xi}^\star)\mathbf{y}_k^\star - h_c(\boldsymbol{\xi}^\star)|, \qquad \forall c \in \{2,\dots,C\}.$$

Similarly as with the previous attribute, we compare the slack values of the subsets by scaling over the sum of the slacks of all subsets:

$$a_{4,c}^{k,\boldsymbol{\xi}^\star} = \frac{s_c^k}{\sum_{k' \in \mathscr{K}} s_c^{k'}}, \quad \forall c \in \{1,\dots,C\}, \quad \forall k \in \mathscr{K}.$$

Deterministic problem-based attributes. This problem solves the 2RO problem for where the newly found scenario $\boldsymbol{\xi}^\star$ is the only scenario considered. We call this a deterministic problem, since we no longer deal with uncertainty. The problem is formulated as follows:

$$\min_{\theta^n, \mathbf{x}^n, \mathbf{y}^n} \quad \theta^n \tag{5.8a}$$
$$\text{s.t.} \quad \theta^n \in \mathbb{R}, \mathbf{x}^n \in \mathscr{X}, \mathbf{y}^n \in \mathscr{Y}, \tag{5.8b}$$
$$\mathbf{c}(\boldsymbol{\xi}^\star)^\top \mathbf{x}^n + \mathbf{d}(\boldsymbol{\xi}^\star)^\top \mathbf{y}^n \le \theta^n, \tag{5.8c}$$
$$T(\boldsymbol{\xi}^\star)\mathbf{x}^n + W(\boldsymbol{\xi}^\star)\mathbf{y}^n \le \mathbf{h}(\boldsymbol{\xi}^\star). \tag{5.8d}$$

By solving this problem we obtain Attributes 5-7:

5. **Deterministic objective** function value θ^n,
6. **Deterministic first-stage** decisions $\mathbf{x}^n$,
7. **Deterministic second-stage** decisions $\mathbf{y}^n$.

Static problem-based attributes. A straightforward method for approximately solving the 2RO problem is first to solve the first-stage decision for all scenarios in the uncertainty set. Then, after a realization of uncertainty, we combine this first-stage decision with the scenario to determine the second-stage decision. This is a naive way of solving 2RO, thus not optimal for all scenarios. But, it could give us some information on the approximate solutions to scenarios in the problem. Solving the static problem consists of two steps: First, we obtain the static robust first-stage decisions $\bar{\mathbf{x}}$ by solving

$$\begin{aligned}
\min_{\theta,\mathbf{x},\mathbf{y}} \quad & \theta && && \text{(5.9a)}\\
\text{s.t.} \quad & \theta \in \mathbb{R}, \mathbf{x} \in \mathscr{X}, \mathbf{y} \in \mathscr{Y}, && && \text{(5.9b)}\\
& \mathbf{c}(\boldsymbol{\xi})^\top \mathbf{x} + \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y} \leq \theta, && \forall \boldsymbol{\xi} \in \Xi, && \text{(5.9c)}\\
& T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi}), && \forall \boldsymbol{\xi} \in \Xi. && \text{(5.9d)}
\end{aligned}$$

This problem can be reformulated via the mathematically tractable formulation [21]. Secondly, by fixing $\mathbf{x}$ to $\bar{\mathbf{x}}$, we obtain the objective value θ^s and second-stage decisions $\mathbf{y}^s$ by solving

$$\begin{aligned}
\min_{\theta^s,\mathbf{y}^s} \quad & \theta^s && \text{(5.10a)}\\
\text{s.t.} \quad & \theta^s \in \mathbb{R}, \mathbf{y}^s \in \mathscr{Y}, && \text{(5.10b)}\\
& \mathbf{c}(\boldsymbol{\xi}^\star)^\top \bar{\mathbf{x}} + \mathbf{d}(\boldsymbol{\xi}^\star)^\top \mathbf{y}^s \leq \theta^s, && \text{(5.10c)}\\
& T(\boldsymbol{\xi}^\star)\bar{\mathbf{x}} + W(\boldsymbol{\xi}^\star)\mathbf{y}^s \leq \mathbf{h}(\boldsymbol{\xi}^\star). && \text{(5.10d)}
\end{aligned}$$

Note that problem (5.9) is solved only once in the algorithm, while problem (5.10) needs to be solved for each scenario. By solving this problem we obtain Attributes 8 and 9:

8. **Static objective** function value θ^s,
9. **Static second-stage** decisions $\mathbf{y}^s$

5.B. OMITTED PSEUDOCODES

The steps for obtaining the ML model used for node selection consists of two parts: (i) making training data and (ii) training the ML model. More details on these steps are given in Procedure 5.3.

Procedure 5.3: STRATEGYMODEL

Input : Train instances $\mathscr{P}_1^{train}(N^{train}),\ldots,\mathscr{P}_I^{train}(N^{train})$
number of partitions for training K^{train},
level for training L^{train},
quality threshold ϵ,
R for random dives per node

Output : Trained node selection strategy *model*.

`// Get training data for` I `instances`

for $i \in \{1,\ldots,I\}$ **do**
  $(D,\mathbf{p})_i \leftarrow$ *generate train data*$(\mathscr{P}_i^{train}(N^{train}), K^{train}, L^{train}, R)$ $\mathbf{q}_i \leftarrow$ *quality*$(\mathbf{p}_i,\epsilon)$
end

`// Train node selection strategy model`

Set *model* $\leftarrow$ *train ML model*$(\{(D,\mathbf{q})_1,\ldots,(D,\mathbf{q})_I\})$

return *model*

For scaling some of the features (see Section 5.3.2), information needs to be gathered by performing several initial dives in the tree. The steps of these dives are given in Procedure 5.4.

Procedure 5.4: INITIALRUN

Input : Test instance $\mathscr{P}^{test}(N^{test})$,
number of partitions for testing K^{test}

Output : Objective value θ, first-stage decisions $\mathbf{x}$, second-stage decisions $\mathbf{y} = \{\mathbf{y}_1,\ldots,\mathbf{y}_K\}$,
Subsets with scenarios Ξ'_k for all $k \in \{1,\ldots,K\}$

Initialization : Incumbent partition: $\tau^i := \{\Xi'_1,\ldots,\Xi'_K\}$, where $\Xi'_k = \emptyset$ for all $k \in \mathscr{K}$,

Set $\tau = \tau^i$
while *solution not robust* **do**
  $(\theta^\star,\mathbf{x}^\star,\mathbf{y}^\star) \leftarrow$ *main problem*(τ)
  $(\boldsymbol{\xi}^\star,\zeta^\star) \leftarrow$ *subproblem*$(\theta^\star,\mathbf{x}^\star,\mathbf{y}^\star)$
  if $\zeta^\star > 0$ **then**
    Solution not robust, random node selection.
    $k' \leftarrow$ *random uniform sample*$([1,K])$
    $\tau := \{\Xi'_1,\ldots,\Xi'_{k'} \cup \{\boldsymbol{\xi}^\star\},\ldots,\Xi'_K\}$
  else
    Robust solution found
    scaling info $\leftarrow (\theta^0,\zeta^0,\kappa^0)$
  end
end
return *scaling info*

5.C. PROBLEM FORMULATIONS

In the experiments, our method has been tested for several problems. The descriptions of these problems and their MILP formulations are given in this section.

5.C.1. CAPITAL BUDGETING WITH LOANS

We consider the capital budgeting with loans problem as defined in Subramanyam et al. [189], where a company wishes to invest in a subset of N projects. Each project i has an uncertain cost $c_i(\boldsymbol{\xi})$ and an uncertain profit $r_i(\boldsymbol{\xi})$, defined as

$$c_i(\boldsymbol{\xi}) = \big(1 + \boldsymbol{\Phi}_i^\intercal \boldsymbol{\xi}/2\big) c_i^0 \quad \text{and} \quad r_i(\boldsymbol{\xi}) = \big(1 + \boldsymbol{\Psi}_i^\intercal \boldsymbol{\xi}/2\big) r_i^0, \quad \forall i \in \{1, \dots, N\},$$

where c_i^0 and r_i^0 represent the nominal cost and the nominal profit of project i, respectively. $\boldsymbol{\Phi}_i^\intercal$ and $\boldsymbol{\Psi}_i^\intercal$ represent the i-th row vectors of the sensitivity matrices $\boldsymbol{\Phi}, \boldsymbol{\Psi} \in \mathbb{R}^{N \times N_z}$. The realizations of the uncertain vector $\boldsymbol{\xi}$ belong to the uncertainty set $\Xi = [-1, 1]^{N_z}$, where N_z is the dimension of the uncertainty set.

The company can invest in a project either before or after observing the risk factor $\boldsymbol{\xi}$. In the latter case, the company generates only a fraction η of the profit, which reflects a penalty of postponement. However, the cost remains the same as in the case of an early investment. The company has a given budget B, which the company can increase by loaning from the bank at a unit cost of $\lambda > 0$, before the risk factors $\boldsymbol{\xi}$ are observed. A loan after the observation occurs, has a unit cost of $\mu\lambda$, with $\mu > 1$. The objective of the capital budgeting problem is to maximize the total revenue subject to the budget. This problem can be formulated as an instance of the K-adaptability problem as follows:

$$\begin{aligned} \max_{(x_0,\mathbf{x})\in\mathscr{X},(y_0,\mathbf{y})\in\mathscr{Y}^K} \min_{\boldsymbol{\xi}\in\Xi} \max_{k\in\mathscr{K}} \quad & \theta \\ \text{s.t.} \quad & \boldsymbol{r}(\boldsymbol{\xi})^\intercal(\mathbf{x} + \eta\mathbf{y}_k) - \lambda(x_0 + \mu y_0^k) \geq \theta, \\ & \mathbf{x} + \mathbf{y}_k \leq \boldsymbol{e}, \\ & \boldsymbol{c}(\boldsymbol{\xi})^\intercal \mathbf{x} \leq B + x_0, \\ & \boldsymbol{c}(\boldsymbol{\xi})^\intercal(\mathbf{x} + \mathbf{y}_k) \leq B + x_0 + y_0^k, \end{aligned}$$

where $\mathscr{X} = \mathscr{Y} = \mathbb{R}_+ \times \{0,1\}^N$, $y_0 = \{y_0^1, \dots, y_0^K\}$, $\mathbf{y} = \{\mathbf{y}_1, \dots, \mathbf{y}_K\}$, x_0 and y_0 are the amounts of taken loan in the first and second stage, respectively. Moreover, x_i and y_i are the binary variables that indicate whether we invest in the i-th project in the first- and second-stage, respectively. The constraints $\boldsymbol{c}(\boldsymbol{\xi})^\intercal \mathbf{x} \leq B + x_0$ ensure that for the first stage, the expenditures are not more than the budget plus the loan taken before the realization of uncertainty.

Test case. Similarly as in Subramanyam et al. [189], the uncertainty set dimension N_z is set to $N_z = 4$. The nominal cost vector $\boldsymbol{c}^0$ is chosen uniformly at random from the set $[0, 10]^N$. Let $\boldsymbol{r}^0 = \boldsymbol{c}^0/5$, $B = \boldsymbol{e}^\intercal \boldsymbol{c}^0/2$, and $\eta = 0.8$. The rows of the sensitivity matrices $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ are sampled uniformly from the i-th row vector, which is sampled from $[0,1]^{N_z}$, such that $\boldsymbol{\Phi}_i^\intercal \boldsymbol{e} = \boldsymbol{\Psi}_i^\intercal \boldsymbol{e} = 1$ for all $i \in \{1, \dots, N\}$. This is also known as the unit simplex in $\mathbb{R}^{N_p}$. For determining the cost of the loans, we set $\lambda = 0.12$ and $\mu = 1.2$.

5.C.2. SHORTEST PATH

We consider the shortest path problem with uncertain arc weights as defined in Subramanyam et al. [189]. Let $G = (V, A)$ be a directed graph with nodes $V = \{1, ..., N\}$, arcs $A \subseteq V \times V$, and arc weights $d_{ij}(\boldsymbol{\xi}) = (1 + \xi_{ij}/2)d^0_{ij}$, $(i, j) \in A$. Where $d^0_{ij} \in \mathbb{R}_+$ represents the nominal weight of the arc $(i, j) \in A$ and ξ_{ij} denotes the uncertain deviation from the nominal weight. The uncertainty set is defined as

$$\Xi = \left\{ \boldsymbol{\xi} \in [0,1]^{|A|} : \sum_{(i,j)\in A} \xi_{ij} \leq \Gamma \right\}.$$

This uncertainty set imposes that at most Γ arc weights may maximally deviate from their nominal values. We need to find the shortest path from the source node s, to the sink node t before observing the realized arc weights. This shortest problem can be formulated as an instance of the K-adaptability problem

$$\begin{aligned} \min_{\mathbf{y}\in\mathscr{Y}^K} \max_{\boldsymbol{\xi}\in\Xi} \min_{k\in\mathscr{K}} \quad & \theta \\ \text{s.t.} \quad & \mathbf{d}(\boldsymbol{\xi})^\top \mathbf{y}_k \leq \theta, \\ & \sum_{(j,l)\in A} y^k_{jl} - \sum_{(i,j)\in A} y^k_{ij} \geq \mathbf{1}_{\{j=s\}} - \mathbf{1}_{\{j=t\}}, \qquad \forall j \in V, \\ & \mathscr{Y} \subseteq \{0,1\}^{|A|}. \end{aligned}$$

Note that this problem contains only binary second-stage decisions and uncertainty in the objective function. The K-B&B algorithm, will find K shortest paths from s to t. After $\boldsymbol{\xi}$ is observed, the path $\mathbf{y}_k$ will be chosen if $\boldsymbol{\xi} \in \Xi_k$.

Normal test case. The coordinates in $\mathbb{R}^2$ for each vertex $i \in V$ are uniformly chosen at random from the region $[0, 10]^2$. The nominal weight of the arc $(i, j) \in A$ is the Euclidean distance between node i and j. The source node s and the sink node t are defined to be the nodes with the maximum nominal distance between them. The $\lfloor 0.9(N^2 - N) \rfloor$ arcs with the highest nominal weight will be deleted to define the arc set A. The budget of the uncertainty set Γ is set to seven.

Sphere test case. The instances of this type have nodes that are spread over a sphere. This is done as follows. First, each node in the three-dimensional graph is sampled from the standard normal distribution and then normalized. The distance between node i and j is then derived by its spherical distance. To obtain this, first the Euclidean distance d_{ij} between node i and j is computed. Then, the arc sine of $d_{ij}/2$ is computed to get the spherical distance. The $\lfloor 0.7(N^2 - N) \rfloor$ arcs with the highest nominal weight will be deleted to define the arc set A. The budget of the uncertainty Γ is set to seven.

5.C.3. KNAPSACK

We consider the two-stage version of the knapsack problem where the profit per item is uncertain. This formulation is based on that of Buchheim and Kurtz [45]. Let N be the number of items, $p_i(\boldsymbol{\xi}) = (1 - \xi_i/2)p^0_i$ the profit for item $i \in \{1, \ldots, N\}$, where $p^0_i \in \mathbb{R}$ is the

nominal profit value, ξ_i is the deviation, $\mathbf{w} \in \mathbb{R}^N$ is the weight vector, and $C = c\sum_{i=1}^{N} w_i$ is the total capacity of the knapsack with $c \in (0,1)$. The uncertainty set is defined as

$$\Xi = \left\{ \boldsymbol{\xi} \in [0,1]^N : \sum_{i=1}^{N} \xi_i \leq \Gamma \right\},$$

where $\Gamma = \gamma N$ and $\gamma \in (0,1)$. This problem can be formulated as an instance of the K-adaptability problem

$$\begin{aligned} \max_{\mathbf{y} \in \mathscr{Y}^K} \min_{\boldsymbol{\xi} \in \Xi} \max_{k \in \mathscr{K}} \quad & \theta \\ \text{s.t.} \quad & \mathbf{p}(\boldsymbol{\xi})^\intercal \mathbf{y}_k \geq \theta, \\ & \mathbf{w}^\intercal \mathbf{y}_k \leq C, \\ & \mathscr{Y} \subseteq \{0,1\}^N, \end{aligned}$$

where y_i is the decision of putting item i in the knapsack.

Test case. The weight w_i of each item $i \in \{1,\dots,N\}$ is uniformly chosen at random from $[1,15]$ and the cost c_i from $[100,150]$. The values of c and γ are selected in Section 5.4.

5.D. PARAMETER TUNING

For both training the ML model (a random forest) and applying it to a problem, we have defined multiple parameters in STRATEGYMODEL and K-B&B-NODESELECTION. The tuning of these parameters is explained in this section.

5.D.1. CAPITAL BUDGETING WITH LOANS

For each $K \in \{2,\dots,6\}$ we have trained five random forests: each for $\epsilon \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. We have first decided on the values of ι for generating training data. Problems become more complex when K grows, which results in more time needed per dive. For tuning these parameters, we have generated training data by running the algorithm for two hours. Thus, we have fixed the parameter T to two. In Table 5.9, for each combination of K and $\iota \in \{2,5,10,15\}$ the following information is shown: number of data points, number of searched instances I, and the average of L^{train} reached per instance.

Table 5.9: Generated training data info for combinations of K and ι. (num. data points, I, average L^{train}).

K	ι (in minutes)			
	2	5	10	15
2	(17284, 60, 9)	(7192, 24, 10)	-	-
3	(37227, 60, 7)	(36510, 24, 8)	(34536, 12, 9)	(30510, 8, 9)
4	(21572, 60, 5)	(21860, 24, 6)	(24312, 12, 7)	(17008, 8, 7)
5	(22425, 60, 5)	(23830, 24, 5)	(22510, 12, 6)	(20040, 8, 6)
6	(17820, 60, 4)	(17544, 24, 5)	(18834, 12, 5)	(17382, 8, 5)

We have then selected per K a value of ι that has high values of the number of data points, I and L^{train}. Then, for each ϵ and K, the dataset related to this value of ι has been trained. The accuracy of these models are given in Table 5.10.

Table 5.10: Test accuracy of random forest for combinations of K (with best ι) and the threshold ϵ.

K (ι)	ϵ 0.05	0.1	0.2	0.3	0.4
2 (2)	0.971	0.988	0.971	0.988	0.983
3 (5)	0.929	0.959	0.959	0.967	0.981
4 (5)	0.922	0.950	0.977	0.982	0.995
5 (5)	0.958	0.937	0.967	0.975	0.992
6 (10)	0.937	0.952	0.968	0.974	0.984

The table above shows that ϵ does not influence the accuracy of the model. However, Figure 5.24 shows that the algorithm performs better when ϵ is very small. If we look at the density of success probabilities p in Figure 5.23, we notice that the vast majority of data points have $p_n \approx 0$. These two observations indicate that any value of p_n slightly higher than zero is special, and the corresponding node is considered as a good node to visit.

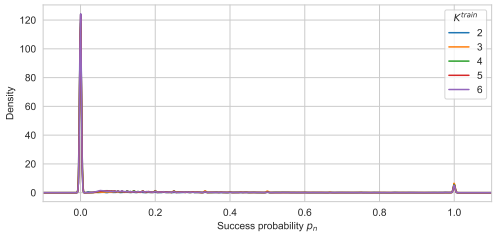


Figure 5.23: Density of the success probability p_n for each value of K^{train}.

In Section 5.4, we noticed that high values of L^{test} outperformed lower ones in the K-B&B-NODESELECTION algorithm. In Figure 5.24, we show the results for higher values of L^{train} than the ones shown in Figure 5.24 for fixed $\epsilon = 0.05$.

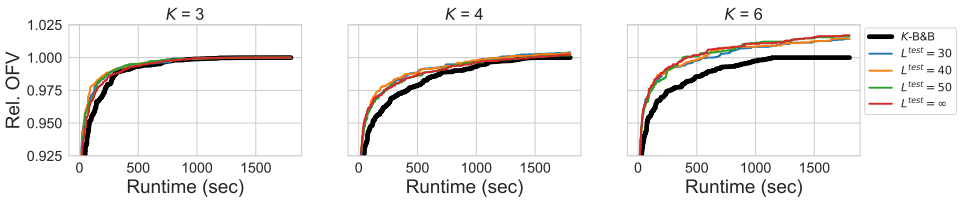


Figure 5.24: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K and bigger values of L^{test}.

5.D.2. SHORTEST PATH ON A SPHERE

For the shortest path problem we also want to decide on the parameter ι per $K \in \{2,3,4,5,6\}$. We noticed that the total number of scenarios needed until a robust solution is found, is larger for shortest path than for capital budgeting. Therefore, the duration per training instance should increase. The range of the number of minutes is $\iota \in \{5,10,15,20\}$. In Table 5.11, for each combination of K and ι the number of data points, instances, and training level L^{train} is given. For now, $T = 2$ is fixed.

Table 5.11: Generated training data info for combinations of K and ι. (num. data points, I, average L^{train}).

K	ι (in minutes)			
	5	10	15	20
2	(10802, 24, 8)	(8290, 12, 9)	(9160, 8, 10)	(8494, 6, 10)
3	(8370, 24, 6)	(6726, 12, 6)	(9858, 8, 6)	(7506, 6, 7)
4	(9884, 24, 5)	(8496, 12, 6)	(7916, 8, 6)	(15872, 6, 6)
5	(13795, 24, 4)	(7415, 12, 5)	(12490, 8, 5)	(21110, 6, 6)
6	(26346, 24, 5)	(10794, 12, 5)	(18948, 8, 5)	(25788, 6, 5)

We have then selected per K a value of ι that has high values of the number of data points, I and L^{train}. Then, for each ϵ and K, the dataset related to this value of ι has been trained. See Table 5.12 for the accuracy of these models.

Table 5.12: Number of data points used for training and test accuracy of random forest for combinations of K (with best ι) and hours spent for generating training data T, given the threshold $\epsilon = 0.05$. (num. data points, test accuracy).

$K(\iota)$	T			
	1	2	5	10
2 (15)	(6586, 0.955)	(9160, 0.946)	(16852, 0.923)	(35674, 0.947)
3 (15)	(4170, 0.952)	(9858, 0.939)	(28347, 0.919)	(47346, 0.937)
4 (20)	(2828, 0.931)	(15872, 0.969)	(37652, 0.966)	(69244, 0.932)
5 (20)	(4790, 0.875)	(21110, 0.972)	(47000, 0.93)	(77735, 0.91)
6 (15)	(13500, 0.948)	(18948, 0.916)	(54564, 0.945)	(91254, 0.955)

We noticed for the shortest path problem that more data points significantly increased the performance of the ML model on the algorithm. An overview of the chosen values of ι and T (hours spent for getting training data) per K are given in Table 5.13.

Table 5.13: Chosen parameter combination for each K. The values of ϵ and L^{test} are fixed to 0.05 and ∞, respectively.

K	ι	T
2	15	10
3	15	5
4	20	5
5	20	10
6	15	10

The density of the success probabilities for the data points of the shortest path prob-

lem is given in Figure 5.25. This is very similar to the density of capital budgeting (see Figure 5.23).

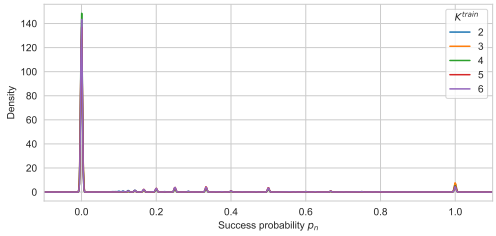


Figure 5.25: Density of the success probability p_n for each value of K^{train}.

5

5.E. FULL RESULTS

We have applied K-B&B-NODESELECTION to multiple problems, where the training and testing instance specifications also varied. In the main body, only a subset of the experiments are shown. In this section, all of them are given.

5.E.1. CAPITAL BUDGETING WITH LOANS

Incumbent objective results over runtime of $N^{test} = 10$ instances are shown in Figure 5.26, of $N^{test} = 20$ instances in Figure 5.27, and of $N^{test} = 30$ instances in Figure 5.28.

5.E.2. SHORTEST PATH ON A SPHERE

Incumbent objective results over runtime of $N^{test} = 20$ instances are shown in Figure 5.29, of $N^{test} = 40$ instances in Figure 5.30, and of $N^{test} = 60$ instances in Figure 5.31.

5.E.3. MIXED PROBLEMS

The incumbent objective results over runtime of the capital budgeting instances with $N^{test} = 10$, trained on shortest path data, is given in Figure 5.32. The results of the shortest path instances with $N^{test} = 20$, trained on capital budgeting data, is given in Figure 5.33.

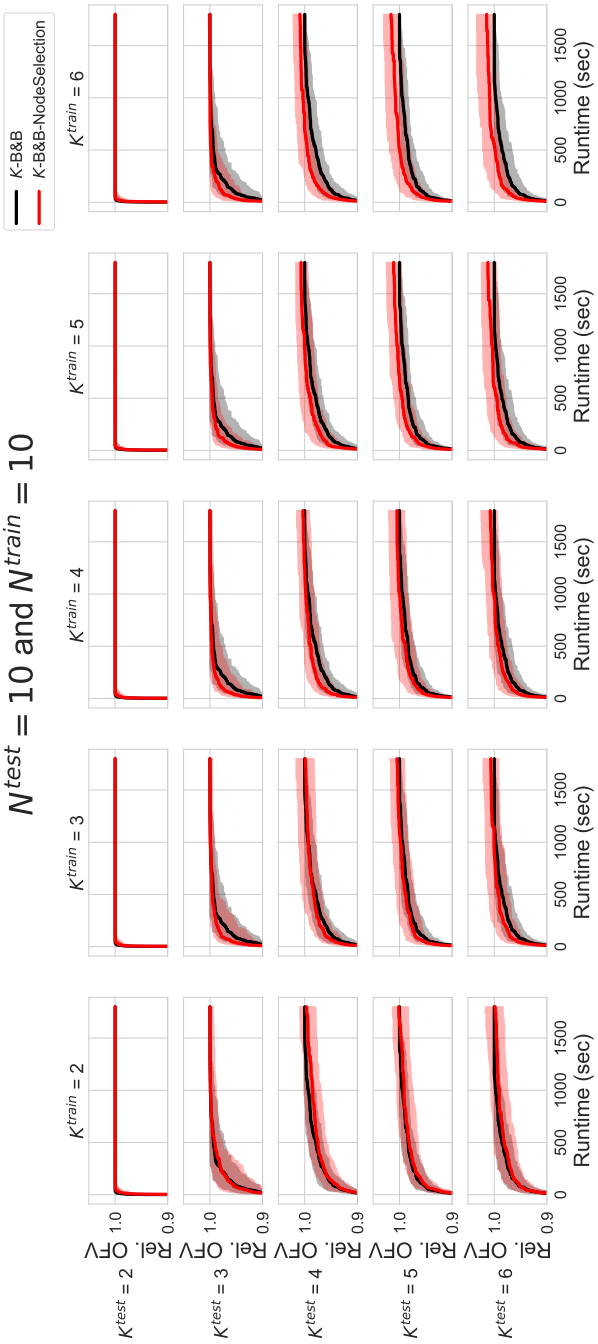


Figure 5.26: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the capital budgeting problem. The results of EXP1 and EXP2 are shown, where $N^{test} = N^{train} = 10$. The regions with shaded color around the curves denote its 75% CI.

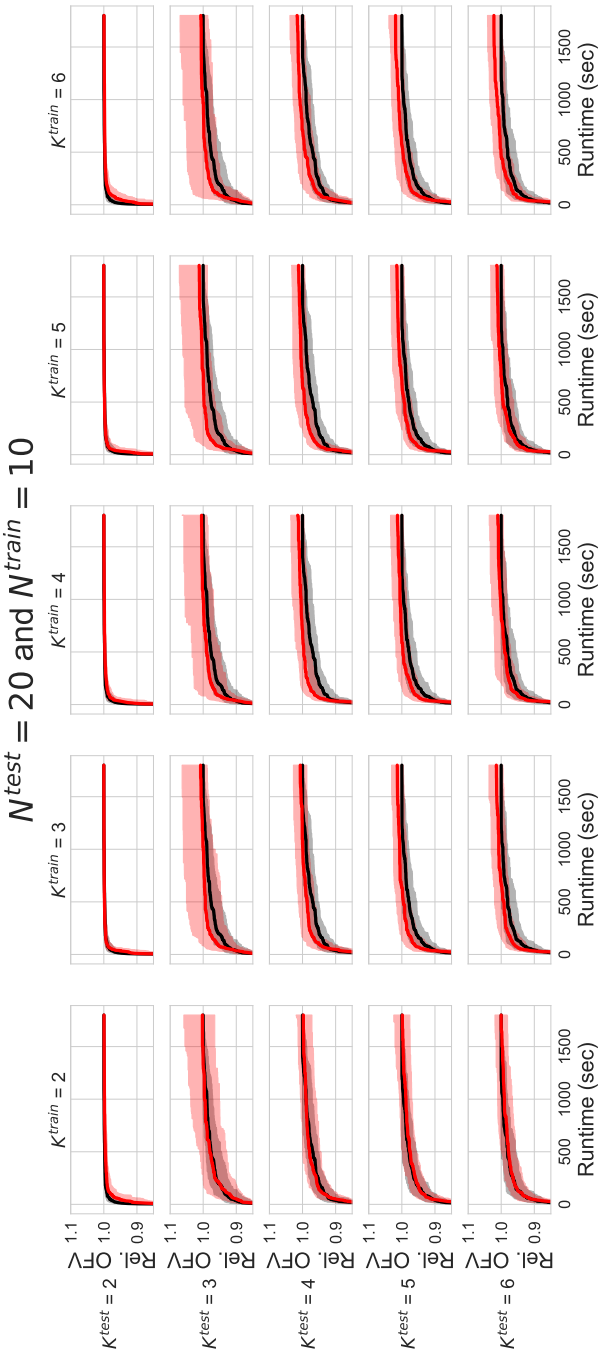


Figure 5.27: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the capital budgeting problem. The results of EXP3 and EXP4 are shown, where $N^{train} = 10, N^{test} = 20$. The regions with shaded color around the curves denote its 75% CI.

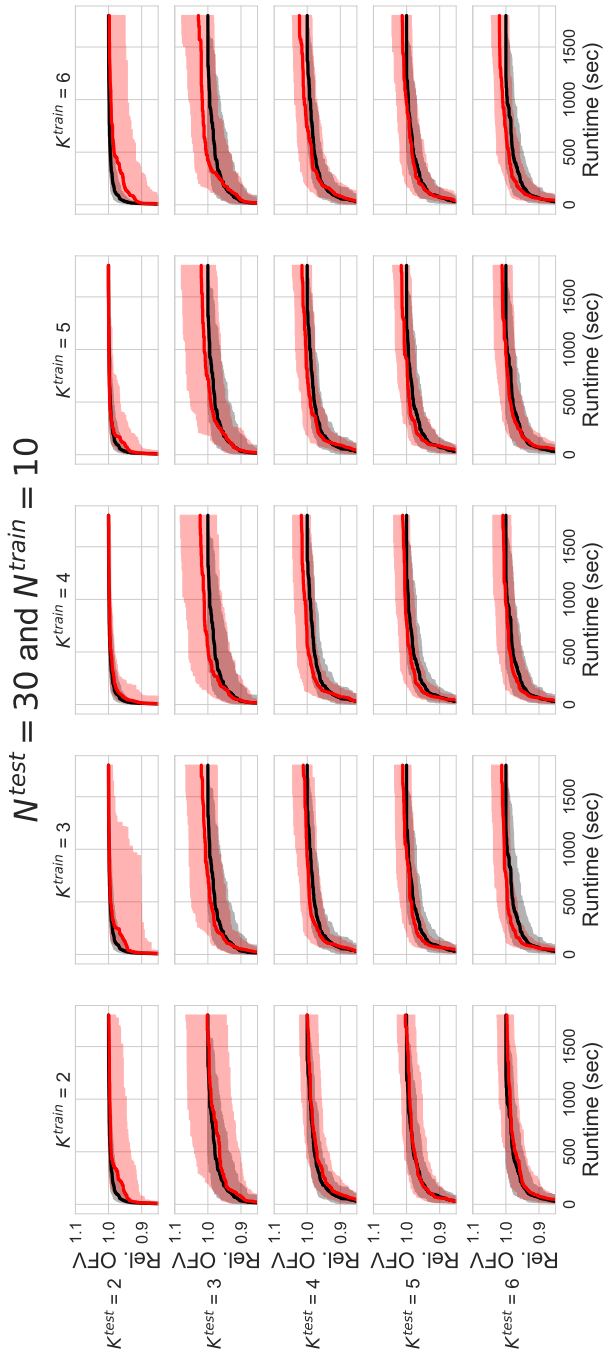


Figure 5.28: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the capital budgeting problem. The results of EXP3 and EXP4 are shown, where $N^{train} = 10, N^{test} = 30$. The regions with shaded color around the curves denote its 75% CI.

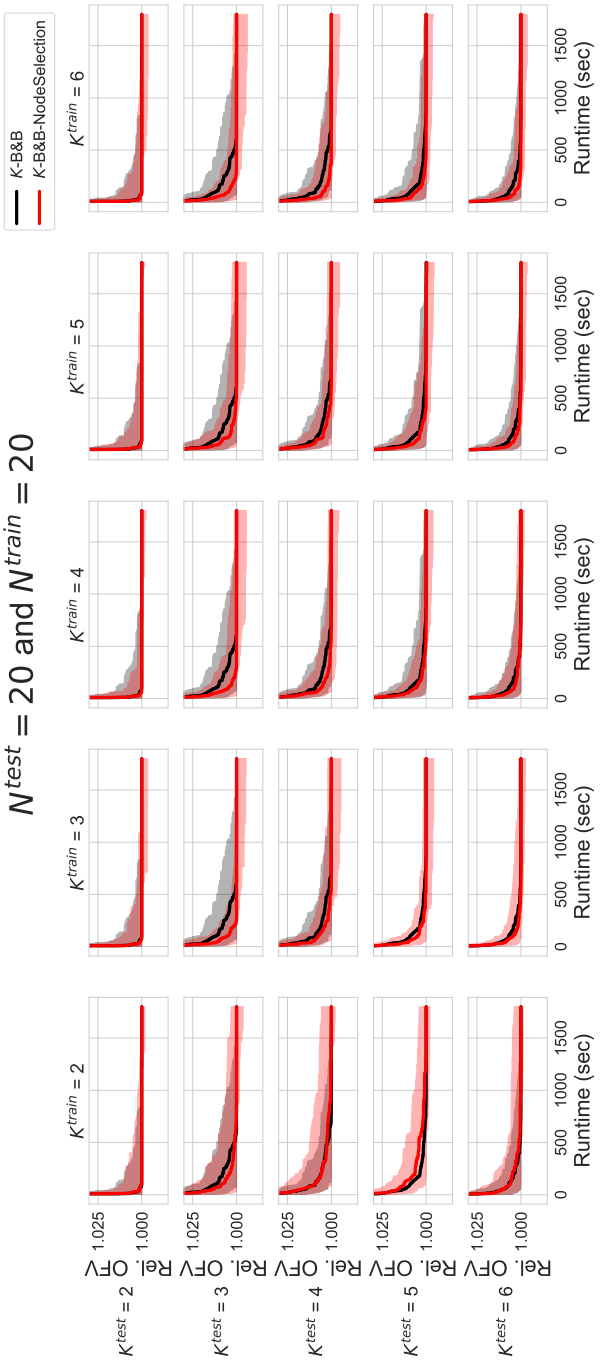


Figure 5.29: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the shortest path problem. The results of EXP1 and EXP2 are shown, where $N^{test} = N^{train} = 20$. The regions with shaded color around the curves denote its 75% CI.

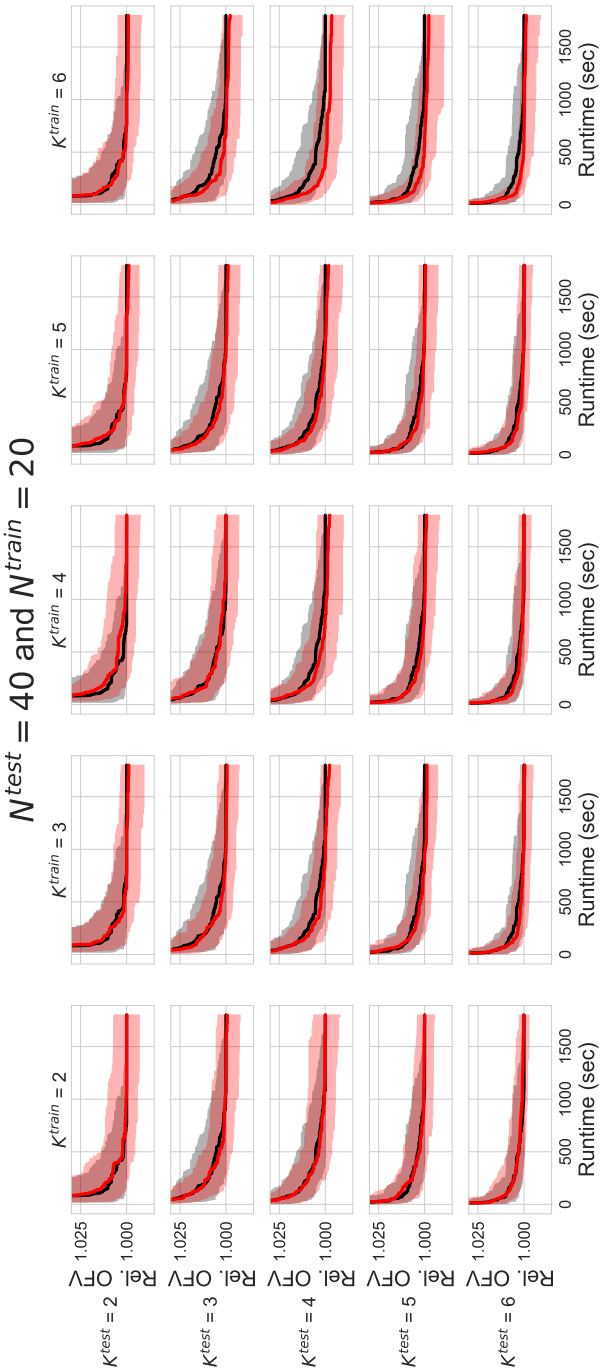


Figure 5.30: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the shortest path problem. The results of EXP3 and EXP4 are shown, where $N^{train} = 20, N^{test} = 40$. The regions with shaded color around the curves denote its 75% CI.

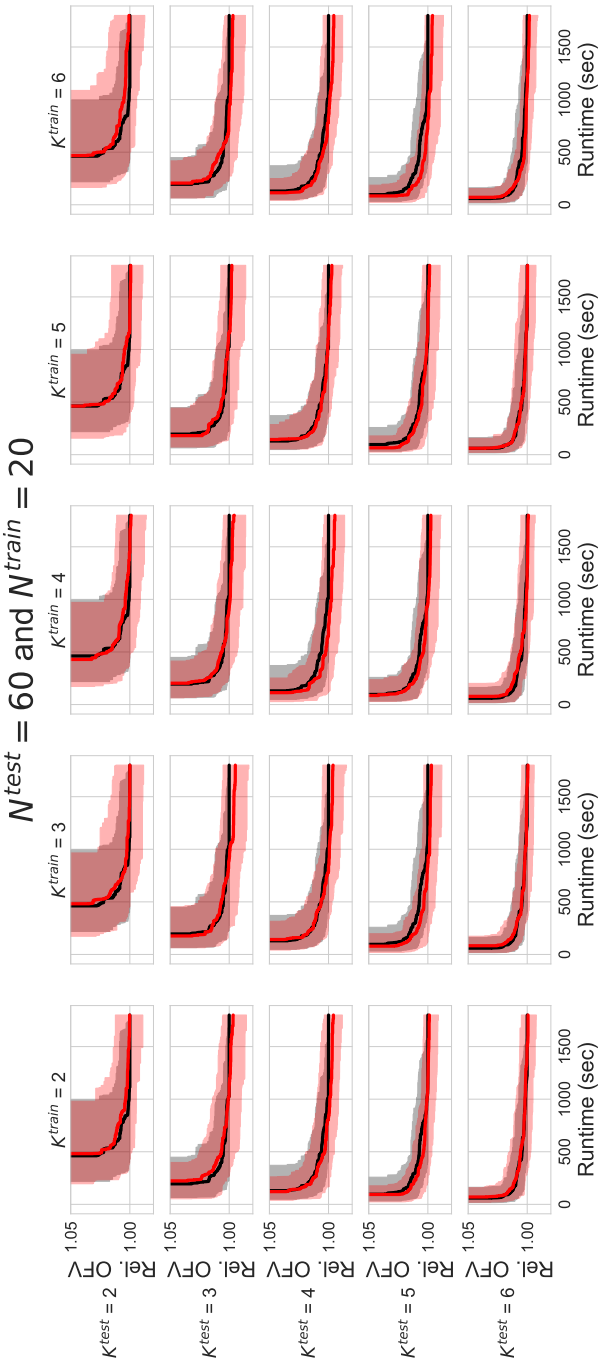


Figure 5.31: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the shortest path problem. The results of EXP3 and EXP4 are shown, where $N^{train} = 20, N^{test} = 60$. The regions with shaded color around the curves denote its 75% CI.

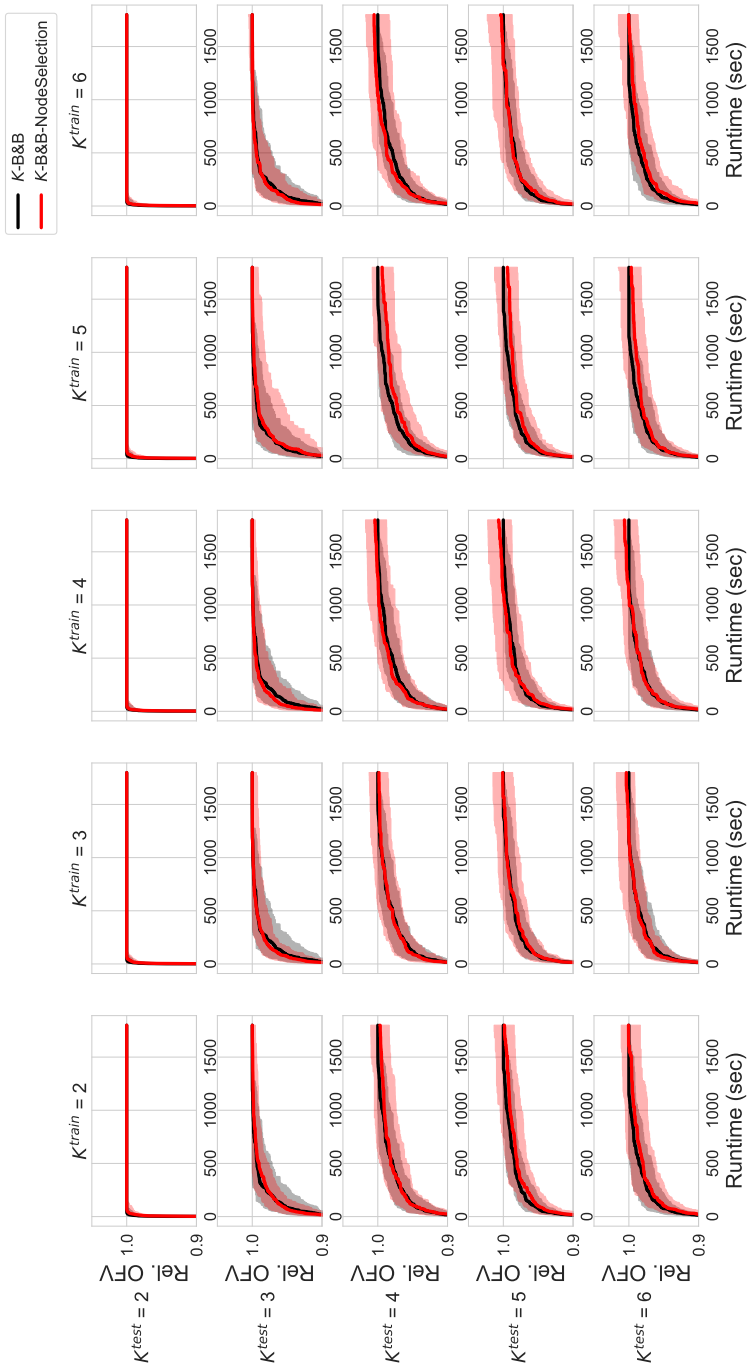


Figure 5.32: Results of the capital budgeting problem. The ML model that is used is trained on shortest path data.

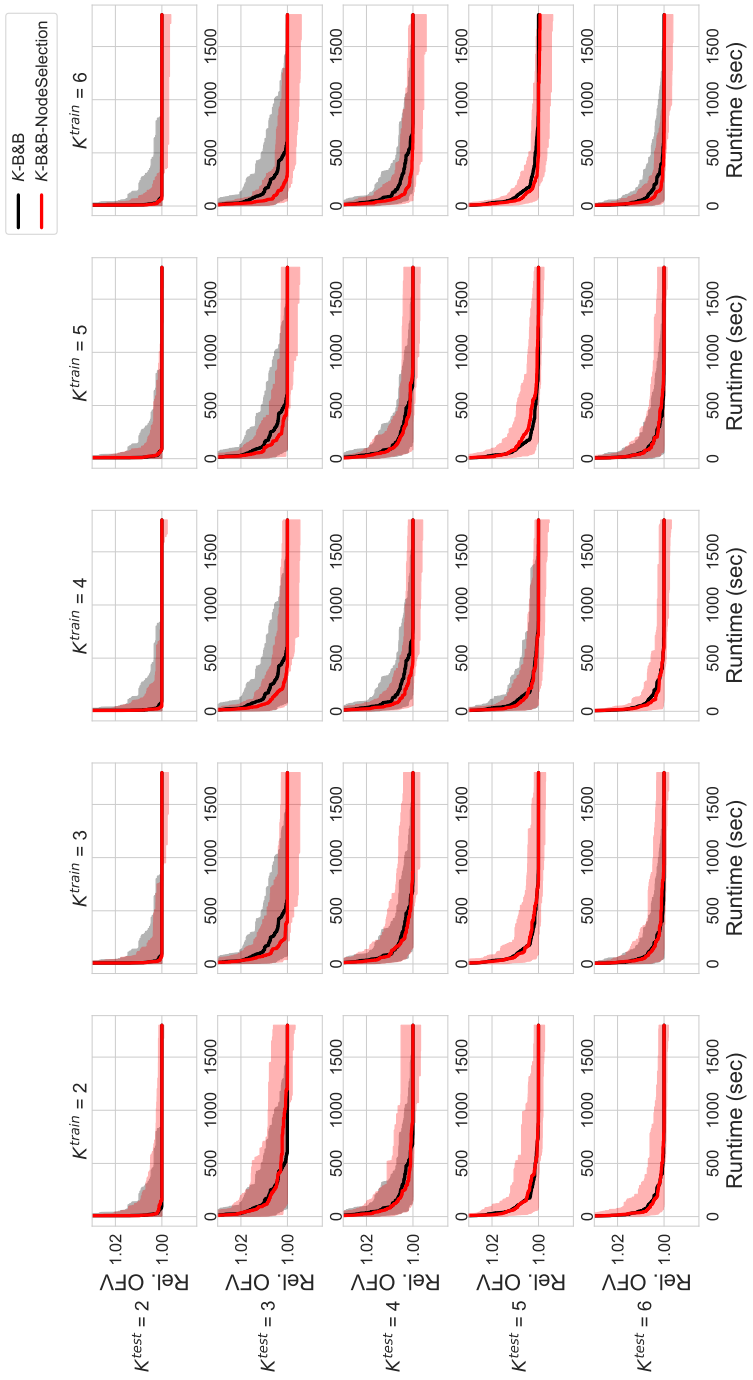


Figure 5.33: Results of the shortest path problem. The ML model that is used is trained on capital budgeting data.

II

PHYLOGENETICS

6

CHERRY-PICKING HEURISTIC FOR BINARY TREES

Combining a set of phylogenetic trees into a single phylogenetic network that explains all of them is a fundamental challenge in evolutionary studies. Existing methods are computationally expensive and can either handle only small numbers of phylogenetic trees or are limited to severely restricted classes of networks.

Proposed data-driven algorithm: *In this chapter, we apply the recently introduced theoretical framework of cherry picking to design a class of efficient heuristics that are guaranteed to produce a network containing each of the input trees, for practical-size datasets consisting of binary trees. Some of the heuristics in this framework are based on the design and training of a machine-learning model that captures essential information on the structure of the input trees and guides the algorithms towards better solutions. We also propose simple and fast randomized heuristics that prove to be very effective when run multiple times. Moreover, our machine-learned heuristics are one of the first applications of machine learning to phylogenetics and show its promise.*

Data generation scheme: *The machine-learning model is a supervised classification model where the data is obtained by sampling many phylogenetic networks (a solution) and then drawing a set of displayed trees (a problem instance). In this way, many data points can be labeled without needing to solve explicit problems.*

Experimental results: *Unlike the existing exact methods, our heuristics are applicable to datasets of practical size, and the experimental study we conducted on both simulated and real data shows that these solutions are qualitatively good, always within some small constant factor from the optimum.*

This chapter is based on Bernardini et al. [27] published in **International Workshop on Algorithms in Bioinformatics (WABI 2022)** and Bernardini et al. [25] published in **Algorithms for Molecular Biology**. In collaboration with Giulia Bernardini, Leo van Iersel, and Leen Stougie. The code is available at `https://github.com/estherjulien/learn2cherrypick`.

6.1. INTRODUCTION

Phylogenetic networks describe the evolutionary relationships between different objects: for example, genes, genomes, or species. One of the first and most natural approaches to constructing phylogenetic networks is to build a network from a set of gene trees. In the absence of incomplete lineage sorting, the constructed network is naturally required to "display", or embed, each of the gene trees. In addition, following the parsimony principle, a network assuming a minimum number of reticulate evolutionary events (like hybridization or lateral gene transfer) is often sought. Unfortunately, the associated computational problem, called HYBRIDIZATION, is NP-hard even for two binary input trees [39], and indeed existing solution methods do not scale well with problem size. For a long time, research on this topic was mostly restricted to inputs consisting of two trees. Proposed algorithms for multiple trees were either completely impractical or ran in reasonable time only for very small numbers of input trees. This situation changed drastically with the introduction of so-called cherry-picking sequences [134]. This theoretical setup opened the door to solving instances consisting of many input trees like most practical datasets have. Indeed, a recent paper showed that this technique can be used to solve instances with up to 100 input trees to optimality [201], although it was restricted to binary trees all having the same leaf set and to so-called "tree-child" networks. Moreover, its running time has a (strong) exponential dependence on the number of reticulate events.

In this chapter, we show significant progress toward a fully practical method by developing a heuristic framework based on cherry picking comprising very fast randomized heuristics and other slower but more accurate heuristics guided by machine learning. Admittedly, our methods are not yet widely applicable since they are still restricted to binary trees. However, our set-up is made in such a way that it may be extendable to general trees. Despite their limitations, we see our current methods already as an important contribution to the field as they are not restricted to tree-child networks and scale well with the number of trees, the number of taxa and the number of reticulations. In fact, we experimentally show that our heuristics can easily handle sets of 100 trees in a reasonable time: the slowest machine-learned method takes 4 minutes on average for sets consisting of 100 trees with 100 leaves each, while the faster, randomized heuristics already find feasible solutions in 2 seconds for the same instances. As the running time of the fastest heuristic depends at most quadratically on the number of input trees, linearly on the number of taxa, and linearly on the output number of reticulations, we expect it to be able to solve much larger instances still in a reasonable amount of time. In addition, in contrast with the existing algorithms, our methods can be applied to trees with different leaf sets, although they have not been specifically optimized for this kind of input. Indeed, we experimentally assessed that our methods give qualitatively good results only when the leaf sets of the input trees have small differences in percentage (up to 5-15%); when the differences are larger, they return feasible solutions that are far from the optimum. Some of the heuristics we present are among the first applications of machine learning in phylogenetics and show its promise. In particular, we show that crucial features of the networks generated in our simulation study can be identified with very high test accuracy (99.8%) purely based on the trees displayed by the networks. It is important to note at this point that no method is able to reconstruct any specific network

from displayed trees as networks are, in general, not uniquely determined by the trees they display [161]. In addition, in some applications, a phenomenon called "incomplete lineage sorting" can cause gene trees that are not displayed by the species network [225], and hence our methods, and other methods based on the HYBRIDIZATION problem, are not (directly) applicable to such data. We focus on *orchard* networks (also called *cherry-picking* networks), which are precisely those networks that can be drawn as a tree with additional horizontal arcs [200]. Such horizontal arcs can for example correspond to lateral gene transfer (LGT). Orchard networks are broadly applicable: in particular, the orchard network class is much bigger than the class of tree-child networks, to which the most efficient existing methods are limited [3].

Related work. Previous practical algorithms for HYBRIDIZATION include PIRN [220], PIRNs [149] and Hybroscale [3], exact methods that are only applicable to (very) small numbers of trees and/or to trees that can be combined into a network with a (very) small reticulation number. Other methods such as PHYLONET [213] and PHYLONETWORKS [186] also construct networks from trees but have different premises and use completely different models. The theoretical framework of cherry picking was introduced in Humphries et al. [101] (for the restricted class of temporal networks) and Linz and Semple [134] (for the class of tree-child networks) and was later turned into algorithms for reconstructing tree-child [201] and temporal [40] networks. These methods can handle instances containing many trees but do not scale well with the number of reticulations, due to an exponential dependence. The class of orchard networks, which is based on cherry picking, was introduced in Semple and Toft [176] and independently (as cherry-picking networks) in Janssen and Murakami [109], although their practical relevance as trees with added horizontal edges was only discovered later [200]. The applicability of machine-learning techniques to phylogenetic problems has not yet been fully explored, and to the best of our knowledge existing work is mainly limited to phylogenetic tree inference [9, 234] and to testing evolutionary hypotheses [126].

Our contributions. We introduce CHERRY-PICKING HEURISTICS (CPH), a class of heuristics to combine a set of binary phylogenetic trees into a single binary phylogenetic network based on cherry picking. We define and analyze several heuristics in the CPH class, all of which are guaranteed to produce feasible solutions to HYBRIDIZATION and all of which can handle instances of practical size (we run experiments on tree sets of up to 100 trees with up to 100 leaves which were processed in on average 4 minutes by our slowest heuristic). Two of the methods we propose are simple but effective randomized heuristics that proved to be extremely fast and to produce good solutions when run multiple times. The main contribution of this chapter consists in a machine-learning model that potentially captures essential information about the structure of the input set of trees. We trained the model on different extensive sets of synthetically generated data and applied it to guide our algorithms towards better solutions. Experimentally, we show that the two machine-learned heuristics we design yield good results when applied to both synthetically generated and real data. We also analyze our machine-learning model to identify the most relevant features and design a non-learned heuristic that is guided by those features only. Our experiments show that this heuristic leads to reason-

ably good results without the need to train a model. This result is interesting per se as it is an example of how machine learning can be used to guide the design of classical algorithms, which are not biased towards certain training data.

A preliminary version of this work appeared in Bernardini et al. [27]. Compared to the preliminary version, we have added the following material: (i), we defined a new non-learned heuristic based on important features and experimentally tested it (Section 6.5.3); (ii), we extended the experimental study to data generated from non-orchard networks (Section 6.5.2), data generated from a class of networks for which the optimum number of reticulations is known (Section 6.5.2) and to input trees with different leaf sets (Section 6.5.2); and (iii), we provided a formal analysis of the time complexity of all our methods (Section 6.4.1) and conducted experiments on their scalability (Section 6.5.2).

6.2. PRELIMINARIES

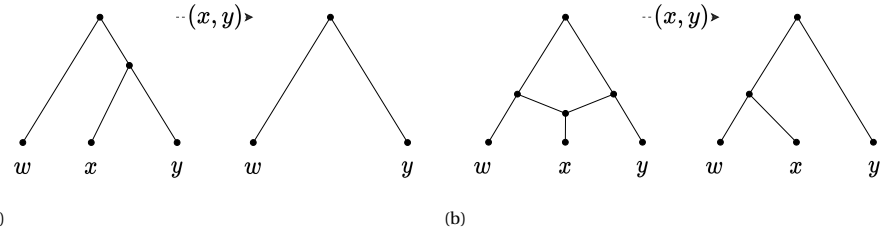
A *phylogenetic network* $N = (V, E, X)$ on a set of taxa X is a directed acyclic graph (V, E) with a single *root* with in-degree 0 and out-degree 1, and the other nodes with either (i) in-degree 1 and out-degree $k > 1$ (*tree nodes*); (ii) in-degree $k > 1$ and out-degree 1 (*reticulations*); or (iii) in-degree 1 and out-degree 0 (*leaves*). The leaves of N are biunivocally labeled by X. A surjective map $\ell : E \to \mathbb{R}^{\geq 0}$ may assign a nonnegative *branch length* to each edge of N. We will denote by $[1, n]$ the set of integers $\{1, 2, ..., n\}$. Throughout this chapter, we will only consider binary networks (with $k = 2$), and we will identify the leaves with their labels. We will also often drop the term "phylogenetic", as all the networks considered in this chapter are phylogenetic networks. The *reticulation number* $r(N)$ of a network N is $\sum_{v \in V} \max\left(0, d^-(v) - 1\right)$, where $d^-(v)$ is the in-degree of v. A network T with $r(T) = 0$ is a *phylogenetic tree*. It is easy to verify that binary networks with $r(N)$ reticulations have $|X| + r(N) - 1$ tree nodes.

CHERRY PICKING

We denote by $\mathcal{N}$ a set of networks and by $\mathcal{T}$ a set of trees. An *ordered* pair of leaves (x, y), $x \neq y$, is a *cherry* in a network if x and y have the same parent; (x, y) is a *reticulated cherry* if the parent $p(x)$ of x is a reticulation, and $p(y)$ is a tree node and a parent of $p(x)$ (see Figure 6.1). A pair is *reducible* if it is either a cherry or a reticulated cherry. Notice that trees have cherries but no reticulated cherries.

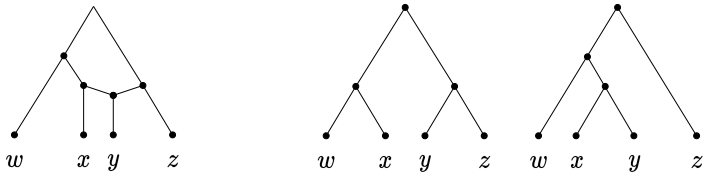
Reducing (or *picking*) a cherry (x, y) in a network N (or in a tree) is the action of deleting x and replacing the two edges $(p(p(x)), p(x))$ and $(p(x), y)$ with a single edge $(p(p(x)), y)$ (see Figure 6.1a). If N has branch lengths, the length of the new edge is $\ell(p(p(x)), y) = \ell(p(p(x)), p(x)) + \ell(p(x), y)$. A reticulated cherry (x, y) is reduced (or picked) by first deleting the edge $(p(y), p(x))$. Then, the edge $(z, p(x))$, incoming to $p(x)$, and the consecutive edge $(p(x), x)$ are replaced with a single edge (z, x). The length of the new edge is $\ell(z, x) = \ell(z, p(x)) + \ell(p(x), x)$ (if N has branch lengths). Another edge $(z', p(y))$, incoming to $p(y)$, and the edge $(p(y), y)$ are replaced by (z', y), with length $\ell(z', y) = \ell(z', p(y)) + \ell(p(y), y)$. Reducing a non-reducible pair has no effect on N. In all cases, the resulting network is denoted by $N_{(x,y)}$: we say that (x, y) affects N if $N \neq N_{(x,y)}$.

Any sequence $S = (x_1, y_1), \ldots, (x_n, y_n)$ of ordered leaf pairs, with $x_i \neq y_i$ for all i, is a *partial cherry-picking sequence*; S is a cherry-picking sequence (CPS) if, for each $i < n$, $y_i \in \{x_{i+1}, \ldots, x_n, y_n\}$. Given a network N and a (partial) CPS S, we denote by N_S the



(a) (b)

Figure 6.1: (x, y) is picked in two different networks. In **(a)** (x, y) is a cherry, and in **(b)** (x, y) is a reticulated cherry. After picking, degree-two nodes are replaced by a single edge.



(a) Network N (b) Input tree set $\mathcal{T}$

Figure 6.2: The two trees in **(b)** are displayed in the network **(a)**.

network obtained by reducing in N each element of S, in order. We denote $S \circ (x, y)$ the sequence obtained by appending pair (x, y) at the end of S. We say that S fully reduces N if N_S consists of the root with a single leaf. N is an *orchard network* if there exists a CPS that fully reduces it, and it is *tree-child* if every non-leaf node has at least one child that is a tree node or a leaf. A *normal* network is a tree-child network such that, in addition, the two parents of a reticulation are always incomparable, i.e., one is not a descendant of the other. If S fully reduces all $N \in \mathcal{N}$, we say that S fully reduces $\mathcal{N}$. In particular, in this chapter we will be interested in a CPS which fully reduces a set of trees $\mathcal{T}$ consisting of $|\mathcal{T}|$ trees of total size $||\mathcal{T}||$.

Adding a cherry (x, y) to a network N when x is not present in N is the action of adding the node z and edge (z, x) and replacing the incoming edge (w, y) of y with the edges (w, z) and (z, y). When x is already present in N, the nodes a and b are added together with the edge (a, b). The incoming edge (w, y) of y is replaced with (w, a) and (a, y). the incoming edge (z, x) of x is replaced with (z, b) and (b, x).

HYBRIDIZATION

The Hybridization problem can be thought of as the computational problem of combining a set of phylogenetic trees into a network with the smallest possible reticulation number, that is, to find a network that displays each of the input trees in the sense specified by Definition 6.1, below. See Figure 6.2 for an example. The definition describes not only what it means to display a tree but also to display another network, which will be useful later.

Definition 6.1. *Let $N = (V, E, X)$ and $N' = (V', E', X')$ be networks on the sets of taxa X and $X' \subseteq X$, respectively. The network N' is* displayed *in N if there is an* embedding *of*

N' in N: an injective map of the nodes of N' to the nodes of N, and of the edges of N' to edge-disjoint paths of N, such that the mapping of the edges respects the mapping of the nodes, and the mapping of the nodes respects the labeling of the leaves.

We call *exhaustive* a tree displayed in $N = (V, E, X)$ with the whole X as a leaf set. Note that Definition 6.1 only involves the topologies of the networks, disregarding possible branch lengths. In the following problem definition, the input trees may or may not have branch lengths, and the output is a network without branch lengths. We allow branch lengths for the input because they will be useful for the machine-learned heuristics of Section 6.4.

HYBRIDIZATION
Input: A set of phylogenetic trees $\mathscr{T}$ on a set of taxa X.
Output: A network displaying $\mathscr{T}$ with minimum possible reticulation number.

In this chapter, we focus on a tree set where each taxon set is X, i.e., on exhaustive trees. Only in Section 6.5.2 do we consider an input of non-exhaustive trees, where X is the union of the trees' taxon sets. In Chapter 7, the heuristic is adapted to better handle instances where not all trees share the same taxon set.

6.3. SOLVING THE HYBRIDIZATION PROBLEM VIA CHERRY-PICKING SEQUENCES

We will develop heuristics for the Hybridization problem using cherry-picking sequences that fully reduce the input trees, leveraging the following result by Janssen and Murakami.

Theorem 6.1 (Janssen and Murakami [109], Theorem 3)**.** *Let N be a binary orchard network, and N' a (not necessarily binary) orchard network on sets of taxa X and $X' \subseteq X$, respectively. If a minimum-length CPS S that fully reduces N also fully reduces N', then N' is displayed in N.*

Notice that HYBRIDIZATION remains NP-hard for binary orchard networks. For binary networks we have the following lemma, a special case of Janssen and Murakami [109, Lemma 1].

Lemma 6.1. *Let N be a binary network, and let (x, y) be a reducible pair of N. Then reducing (x, y) and then adding it back to $N_{(x,y)}$ results in N.*

Note that Lemma 6.1 only holds for binary networks: in fact, there are different ways to add a pair to a non-binary network, thus the lemma does not hold unless a specific rule for adding pairs is specified (inspect Janssen and Murakami [109] for details). Theorem 6.1 and Lemma 6.1 provide the following approach for finding a feasible solution to HYBRIDIZATION: find a CPS S that fully reduces all the input trees, and then uniquely reconstruct the binary orchard network N for which S is a minimum-length CPS, by processing S in the reverse order. N can be reconstructed from S using one of the methods underlying Lemma 6.1 proposed in the literature, e.g., in Janssen and Murakami [109] (illustrated in Figure 6.3) or in van Iersel et al. [201]. The following lemma relates the length of a CPS S and the number of reticulations of the network constructed from S.

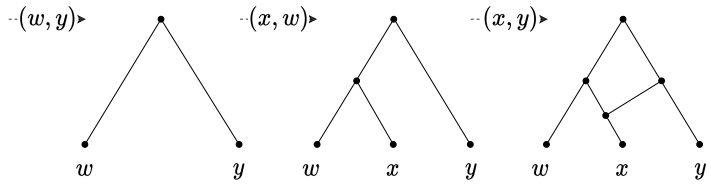


Figure 6.3: The network reconstructed from the sequence $S = (x,y),(x,w),(w,y)$. The pairs are added to the network in reverse order: if the first element of a pair is not yet in the network, it is added as a cherry with the second element (see the pair (x,w)). Otherwise, a reticulation is added above the first element with an incoming edge from a new parent of the second element (see the pair (x,y)).

Lemma 6.2 (van Iersel et al. [202])**.** *Let S be a CPS on a set of taxa X. The number of reticulations of the network N reconstructed from S is $r(N) = |S| - |X| + 1$.*

In the next section we focus on the first part of the heuristic: producing a CPS that fully reduces a given set of phylogenetic trees.

6.3.1. RANDOMIZED HEURISTICS

We define a class of randomized heuristics that construct a CPS by picking one reducible pair of the input set $\mathcal{T}$ at a time and by appending this pair to a growing partial sequence, as described in Algorithm 6.1 (the two subroutines PickNext and CompleteSeq will be later described in detail). We call this class CPH (for Cherry-Picking Heuristics). Recall that $\mathcal{T}_S$ denotes the set of trees $\mathcal{T}$ after reducing all trees with a (partial) CPS S.

Algorithm 6.1: CPH

```
   Input  : A set 𝒯 of phylogenetic trees
   Output : A CPS reducing 𝒯.
1  S ← ∅
2  while there is a reducible pair in 𝒯_S do
3  |  (x, y) ← PickNext(𝒯_S)
4  |  S ← S ∘ (x, y)
5  |  Reduce (x, y) in all trees of 𝒯_S
6  end
7  S ← CompleteSeq(S)
8  return S
```

The while loop at lines 2-5 produces, in general, a partial CPS S, as shown in Example 6.1. To make it into a CPS, the subroutine CompleteSeq at line 7 appends at the end of S a sequence S' of pairs such that each second element in a pair of $S \circ S'$ is a first element in a later pair (except for the last one), as required by the definition of CPS. These additional pairs do not affect the trees in $\mathcal{T}$, which are already fully reduced by S. Algorithm 6.2 describes a procedure CompleteSeq that runs in time linear in the length of S.

Example 6.1. *Let $\mathcal{T}$ consist of the 2-leaf trees (x,y) and (w,z). A partial CPS at the end of the while loop in Algorithm 6.1 could be, e.g., $S = (x,y),(w,z)$. The trees are both reduced*

to one leaf, so there are no more reducible pairs, but S is not a CPS. To make it into a CPS either pair (y,z) or pair (z,y) can be appended: e.g., $S \circ (y,z) = (x,y),(w,z),(y,z)$ is a CPS, and it still fully reduces the two input trees.

Algorithm 6.2: CompleteSeq

Input : A partial CPS $S = (x_1,y_1),\ldots,(x_n,y_n)$ that reduces $\mathcal{T}$
Output : A CPS S' for $\mathcal{T}$.

```
1  C ← ∅; P ← ∅
2  for i = n,...,1 do
3  |  if y_i ∉ C then
4  |  |  P ← P ∪ {y_i}
5  |  end
6  |  C ← C ∪ {x_i, y_i}
7  end
8  S' ← S
9  while |P| > 1 do
10 |  Let r_1 and r_2 be two arbitrary elements of P
11 |  S' ← S' ∘ (r_1, r_2)
12 |  P ← P \ {r_1}
13 end
14 return S'
```

The class of heuristics given by Algorithm 6.1 is concretized in different heuristics depending on the function PickNext at line 3 used to choose a reducible pair at each iteration. To formulate them we need to introduce the following notions of height pair and trivial pair. Let N be a network with branch lengths and let (x,y) be a reducible pair in N. The *height pair* of (x,y) in N is a pair $(h_x^N, h_y^N) \in \mathbb{R}^2_{\geq 0}$, where $h_x^N = \ell(p(x),x)$ and $h_y^N = \ell(p(y),y)$ if (x,y) is a cherry (indeed, in this case, $p(x) = p(y)$); $h_x^N = \ell(p(y),p(x)) + \ell(p(x),x)$ and $h_y^N = \ell(p(y),y)$ if (x,y) is a reticulated cherry. The *height* $h_{(x,y)}^N$ of (x,y) is the average $(h_x^N + h_y^N)/2$ of h_x^N and h_y^N. Let $\mathcal{T}$ be a set of trees whose leaf sets are subsets of a set of taxa X. An ordered leaf pair (x,y) is a *trivial pair* of $\mathcal{T}$ if it is reducible in all $T \in \mathcal{T}$ that contain both x and y, and there is at least one tree in which it is reducible.

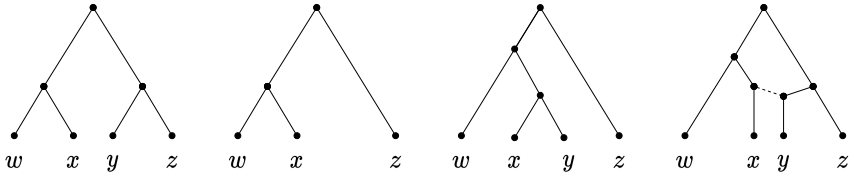
We define the following three heuristics in the CPH class, resulting from as many possible implementations of PickNext.

Rand. Function PickNext picks uniformly at random a reducible pair of $\mathcal{T}_S$.

LowPair. Function PickNext picks a reducible pair (x,y) with the lowest average of values $h_{(x,y)}^T$ over all $T \in \mathcal{T}_S$ in which (x,y) is reducible (ties are broken randomly).

TrivialRand. Function PickNext picks a trivial pair if there exists one and otherwise picks a reducible pair of $\mathcal{T}_S$ uniformly at random.

Theorem 6.2. *Algorithm 6.1 computes a CPS that fully reduces $\mathcal{T}$, for any function PickNext that picks, in each iteration, a reducible pair of $\mathcal{T}_S$.*



(a) Input tree T (b) $T^{(1)} = T_{(y,z)}$ (c) Expand $T^{(1)}$ with (x,y) (d) Network N_T

Figure 6.4: Tree expansion of T **(a)** with the trivial cherry (x,y) of $\mathcal{T}_{(y,z)}$. **(b)** After picking cherry (y,z), leaf y is missing in $T^{(1)}$. **(c)** Leaf x is replaced by the cherry (x,y). After completion of the heuristic, we have $S_T = (y,z),(x,y),(y,w),(w,z)$. **(d)** The network N_T reconstructed from $S^1 \cdot (x,y)$. Note that the input tree T is displayed in N_T (solid edges).

Proof. The sequence S is initiated as an empty sequence. Then, each iteration of the while loop (lines 2-5) of Algorithm 6.1 appends one pair to S that is reducible in at least one of the trees in $\mathcal{T}$, and reduces it in all trees. Hence, in each iteration, the total size of $\mathcal{T}_S$ is reduced, so the algorithm finishes in finite time. Moreover, at the end of the while loop, each tree in $\mathcal{T}_S$ is reduced, thus the partial CPS S reduces $\mathcal{T}_S$. As CompleteSeq only appends pairs at the end of S, the result of this subroutine still reduces all trees in $\mathcal{T}_S$. □

In Section 6.5 we experimentally show that TrivialRand produces the best results among the proposed randomized heuristics. In the next section, we introduce a further heuristic step for TrivialRand which improves the output quality.

6.3.2. IMPROVING HEURISTIC TRIVIALRAND VIA TREE EXPANSION

Let $\mathcal{T}$ be a set of trees whose leaf sets are subsets of a set of taxa X, let S be a partial CPS for $\mathcal{T}$ and let $\mathcal{T}_S$ be the tree set obtained by reducing in order the pairs of S in $\mathcal{T}$. With respect to a trivial pair (x,y), each tree $T \in \mathcal{T}_S$ is of one of the following types: (i) (x,y) is reducible in T; or (ii) neither x nor y are leaves of T; or (iii) y is a leaf of T but x is not; or (iv) x is a leaf of T but y is not.

Suppose that at some iteration of TrivialRand, the subroutine PickNext returns the trivial pair (x,y). Then, before reducing (x,y) in all trees, we do the following extra step: for each tree of type (iv), replace leaf x with cherry (x,y). We call this operation the *tree expansion*: see Figure 6.4(c). The effect of this step is that, after reducing (x,y), leaf x disappears from the set of trees, which would have not necessarily been the case before, because of trees of type (iv). Tree expansion followed by the reduction of (x,y) can, alternatively, be seen as relabeling leaf x in any tree of type (iv) by y. The choice of describing this relabeling as tree expansion is just for the purpose of proving Lemma 6.3.

To guarantee that a CPS S produced with tree expansion implies a feasible solution for HYBRIDIZATION, we must show that the network N reconstructed from S displays all the trees in the input set $\mathcal{T}$. We prove that indeed this is the case with the following steps: (1), we consider the networks N_T obtained by "reverting" a partial CPS S obtained right after applying tree expansion to a tree T_S: in other words, to obtain N_T we add to the partially reduced tree T_S the trivial pair (x,y) and then all the pairs previously reduced by S in the sense of Lemma 6.1. We show that N_T always displays T, the original tree; (2),

we prove that this holds for an arbitrary sequence of tree expansion operations; and (3), since the CPS obtained using tree expansions fully reduces the networks of point (2), and since these networks display the trees in the original set $\mathcal{T}$, we have the desired property by Theorem 6.1. We prove this more formally with the following lemma.

Lemma 6.3. *Let S be the CPS produced by* TrivialRand *using tree expansion with input $\mathcal{T}$. Then the network reconstructed from S displays all the trees in $\mathcal{T}$.*

Proof. Let us start with the case where only 1 tree expansion occurs. Let $S^{(i-1)}$ be the partial CPS constructed in the first $i-1$ steps of TrivialRand, and let i be the step in which we pick a trivial pair (x,y). For each $T \in \mathcal{T}_{S^{(i-1)}}$ that is reduced by $S^{(i-1)}$ to a tree $T^{(i-1)}$ of type (iv) for (x,y), let $S_T^{(i-1)}$ be the subsequence of $S^{(i-1)}$ consisting only of the pairs that subsequently affect T. We use the partial CPS $S_T^i = S_T^{(i-1)} \circ (x,y)$ to reconstruct a network N_T with a method underlying Lemma 6.1, starting from $T^{(i-1)}$: see Figure 6.4(d).

For trees of type (i)-(iii), $N_T = T$. We call the set $\mathcal{N}_{\mathcal{T}}$, consisting of the networks N_T for all $T \in \mathcal{T}$, the *expanded reconstruction* of $\mathcal{T}$. Note that, by construction and Lemma 6.1, all the elements of $\mathcal{N}_{\mathcal{T}}$ after reducing, in order, the pairs of $S^{(i-1)} \circ (x,y)$, are trees: in particular, they are equal to the trees of $\mathcal{T}_{S^{(i-1)} \circ (x,y)}$ in which all the labels y have been replaced by x. We denote this set of trees $(\mathcal{N}_{\mathcal{T}})_{S^{(i-1)} \circ (x,y)}$.

We can generalize this notion to multiple trivial pairs: we denote by $\mathcal{N}_{\mathcal{T}}^{(j)}$ the expanded reconstruction of $\mathcal{T}$ with the first j trivial pairs, and suppose we added the j-th pair (w,z) to the partial CPS S at the k-th step. Consider a tree $T' \in (\mathcal{N}_{\mathcal{T}}^{(j-1)})_{S^{(k-1)}}$ of type (iv) for (w,z), and let $N_T^{(j-1)} \in \mathcal{N}_{\mathcal{T}}^{(j-1)}$ be the network it originated from. Let $S_T^{(k-1)}$ be the subsequence of $S^{(k-1)}$ consisting only of the pairs that subsequently affected $N_T^{(j-1)}$. Then $N_T^{(j)}$ is the network reconstructed from $S_T^{(k-1)} \circ (w,z)$, starting from T'. For trees of $(\mathcal{N}_{\mathcal{T}}^{(j-1)})_{S^{(k-1)}}$ that are of type (i)-(iii) for (w,z), we have $N_T^{(j)} = N_T^{(j-1)}$. The elements of $\mathcal{N}_{\mathcal{T}}^{(j)}$ are all networks $N_T^{(j)}$. For completeness, we define $\mathcal{N}_{\mathcal{T}}^{(0)} = \mathcal{T}$ and $\mathcal{N}_{\mathcal{T}}^{(1)} = \mathcal{N}_{\mathcal{T}}$.

By construction, S fully reduces all the networks in $\mathcal{N}_{\mathcal{T}}^{(j)}$, thus the network N reconstructed from S displays all of them by Theorem 6.1. We prove that $N_T^{(j)}$ displays T for all $T \in \mathcal{T}$, and thus N displays the original tree set $\mathcal{T}$ too, by induction on j.

In the base case, we pick $j = 0$ trivial pairs, so the statement is true by Theorem 6.1. Now let $j > 0$. The induction hypothesis is that each network $N_T^{(j-1)} \in \mathcal{N}_{\mathcal{T}}^{(j-1)}$ displays the tree $T \in \mathcal{T}$ it originated from. Let (w,z) be the j-th trivial pair, added to the sequence at position k. Let $T' \in (\mathcal{N}_{\mathcal{T}}^{(j-1)})_{S^{(k-1)}}$ be a tree of type (iv) for (w,z), and let $N_T^{(j-1)}$ be the network it originates from. Then there are two possibilities: either z is a leaf of $N_T^{(j-1)}$ or it is not. In case it is not, then adding (w,z) to $N_T^{(j-1)}$ does not create any new reticulation, and clearly $N_T^{(j)}$ keeps displaying T. If z does appear in $N_T^{(j-1)}$, then it must have been reduced by a pair (z,v) of $S^{(k-1)}$ (otherwise T' would not be of type (iv)). Then the network $N_T^{(j)}$ has an extra reticulation, created with the insertion of (z,v) at some point after (w,z) during the backward reconstruction. In both cases, by Janssen and Murakami [109, Lemma 10] $N_T^{(j-1)}$ is displayed in $N_T^{(j)}$, and thus by the induction hypothesis T is displayed too. □

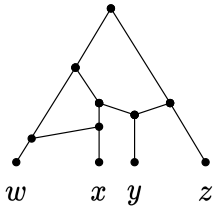


Figure 6.5: Network N' of Example 6.2.

6.3.3. GOOD CHERRIES IN THEORY

By Lemma 6.1 the binary network N reconstructed from a CPS S is such that S is of minimum length for N, that is, there exists no shorter CPS that fully reduces N. By Theorem 6.1 if S, in turn, fully reduces $\mathcal{T}$, then N displays all the trees in $\mathcal{T}$. Depending on S, though, N is not necessarily a network with minimum reticulation number among the ones displaying $\mathcal{T}$ as we aim to find the shortest CPS (see Lemma 6.2). For an illustration, see the following example.

Example 6.2. *Consider the set $\mathcal{T}$ of Figure 6.2b: $S = (y,x),(y,z),(w,x),(x,z)$ is a CPS that fully reduces $\mathcal{T}$ and consists only of pairs successively reducible in the network N of Fig. 6.2a, thus it reconstructs it by Lemma 6.1. Now consider (w,x), which is reducible in $\mathcal{T}$ but not in N, and pick it as first pair, to obtain e.g. $S' = (w,x),(y,z),(y,x),(w,x),(x,z)$. The network N' reconstructed from S', depicted in Figure 6.5, has $r(N') = 2$, whereas $r(N) = 1$.*

For the class of tree-child networks, we have as characterization that a minimum-sized tree-child sequence (TCS) (see Definition 6.2) gives an *optimal* solution to the TREE-CHILD HYBRIDIZATION problem [134, Theorem 2.1]. This problem takes the description of HYBRIDIZATION (in Section 6.2), but is restricted to generate tree-child networks. From Janssen and Murakami [109, Theorem 5], we have for two tree-child networks N and N', that N' is contained in N if and only if any TCS of N also reduces N'.

Definition 6.2. *A tree-child sequence (TCS) is a CPS in which every leaf appearing as the first coordinate of a pair does not appear as a second coordinate of a pair in any of the subsequent cherries in the sequence.*

In our case, we focus on reconstructing an orchard phylogenetic network N (which is not necessarily tree-child) on $\mathcal{T}$. Each tree is tree child, and by construction displayed in N. By the former given theorem of [109, Theorem 5], a TCS exists that reduces the network and all trees in $\mathcal{T}$. In our heuristic setting, we derive a CPS to construct orchard networks. We lose the tree-child condition as orchard networks are more general than tree-child networks. Moreover, the CPH of Algorithm 6.1 cannot be easily extended to TCSs; the restrictions can cause complications after some iterations, as it might be the case that none of the reducible pairs left in $\mathcal{T}$ can be picked (see Example 6.3). However, we still use the idea that the trees in $\mathcal{T}$ can be reduced by a (regular) CPS of a network N that displays $\mathcal{T}$, for determining which cherries would be good to pick.

Example 6.3. *Consider the set $\mathcal{T}$ of Figure 6.2b. Suppose that we first pick (y,z), then (x,w), and then (w,z), comprising the partial CPS $S=(y,z)(x,w)(w,z)$. Note that thus far we have complied with the property of a TCS. The only remaining reducible pairs in $\mathcal{T}_S$ are (x,y) or (y,x). Picking either of them violates the TCS property.*

Let $\mathsf{OPT}(\mathcal{T})$ denote the set of networks that display $\mathcal{T}$ with the minimum possible number of reticulations (in general, this set contains more than one network). Ideally, we would like to produce a CPS fully reducing $\mathcal{T}$ that is also a minimum-length CPS fully reducing some network of $\mathsf{OPT}(\mathcal{T})$. In other words, we aim to find a CPS $\tilde{S}=(x_1,y_1),\ldots,(x_n,y_n)$ such that, for any $i\in[1,n]$, (x_i,y_i) is a reducible pair of $\tilde{N}_{\tilde{S}^{(i-1)}}$, where $\tilde{S}^{(0)}=\emptyset$, $\tilde{S}^{(k)}=(x_1,y_1),\ldots,(x_k,y_k)$ for all $k\in[1,n]$, and $\tilde{N}\in\mathsf{OPT}(\mathcal{T})$. Let $S=(x_1,y_1),\ldots,(x_n,y_n)$ be a CPS fully reducing $\mathcal{T}$ and let $\mathsf{OPT}^{(k)}(\mathcal{T})$ consist of all networks $N\in\mathsf{OPT}(\mathcal{T})$ such that each pair (x_i,y_i), $i\in[1,k]$, is reducible in $N_{S^{(i-1)}}$.

Suppose we are incrementally constructing a CPS $S=(x_1,y_1),\ldots,(x_n,y_n)$ for $\mathcal{T}$ with some heuristic in the CPH class. If we had an oracle that at each iteration i told us if a reducible pair (x,y) of $\mathcal{T}^{(i-1)}$ were a reducible pair in some $N\in\mathsf{OPT}^{(i-1)}(\mathcal{T})$, then picking such a cherry would likely lead to a short CPS, which consequently leads to a network with few reticulations.

6.4. PREDICTING GOOD CHERRIES VIA MACHINE LEARNING

In this section, we present a supervised machine-learning classifier that (imperfectly) simulates the ideal oracle described at the end of Section 6.3.3. The goal is to predict, based on $\mathcal{T}$, whether a given cherry of $\mathcal{T}$ is a cherry or a reticulated cherry in a network N displaying $\mathcal{T}$ with a close-to-optimal number of reticulations, without knowing N. We then exploit the output of the classifier to define new functions PickNext, that in turn define new machine-learned heuristics in the class of CPH (Algorithm 6.1).

Specifically, we train a random forest classifier on data that encapsulates information on the cherries in the tree set. Given a partial CPS, each reducible pair in $\mathcal{T}_S$ is represented by one data point. Each data point is a pair $(\mathbf{F},\mathbf{c})$, where $\mathbf{F}$ is an array containing the features of a cherry (x,y) and $\mathbf{c}$ is an array containing the probability that the cherry belongs to each of the possible classes described below. Recall that cherries are ordered pairs, so (x,y) and (y,x) give rise to two distinct data points. The classification model learns the association between $\mathbf{F}$ and $\mathbf{c}$.

The true class of a cherry (x,y) of $\mathcal{T}$ depends on whether, for the (unknown) network N that we aim to reconstruct: (class 1) (x,y) is a cherry of N; (class 2) (x,y) is a reticulated cherry of N; (class 3) (x,y) is not reducible in N, but (y,x) is a reticulated cherry; or (class 4) neither (x,y) nor (y,x) are reducible in N. Thus, for the data point of a cherry (x,y), $\mathbf{c}[i]$ contains the probability that (x,y) is in class i, and $\mathbf{c}[1]+\mathbf{c}[2]$ gives the predicted probability that (x,y) is reducible in N. We define the following two heuristics in the CPH framework.

ML. Given a threshold $\tau\in[0,1)$, function PickNext picks the cherry with the highest predicted probability of being reducible in N if this probability is at least τ; or a random cherry if none has a probability of being reducible above τ.

TrivialML. Function PickNext picks a random trivial pair, if there exists one; otherwise it uses the same rules as ML.

In both cases, whenever a trivial pair is picked, we do tree expansion, as described in Section 6.3.2. Note that if $\tau = 0$, since the predicted probabilities are never exactly 0, ML is fully deterministic. In Section 6.5.2 we show how the performance of ML is impacted by the choice of different thresholds.

Table 6.1: Features of a cherry (x, y). Features 6-12 can be computed for both branch lengths and unweighted branches. We refer to these two options as *distance* and *topological distance*, respectively.

Num	Feature name	Description
1	Cherry in tree	Ratio of trees that contain cherry (x, y)
2	New cherries	Number of new cherries of $\mathcal{T}$ after picking cherry (x, y)
3	Before/after	Ratio of the number of cherries of $\mathcal{T}$ before/after picking cherry (x, y)
4	Trivial	Ratio of trees with both leaves x and y that contain cherry (x, y)
5	Leaves in tree	Ratio of trees that contain both leaves x and y
Features measured by distance (d) and topology (t)		
$6_{d,t}$	Tree depth	Avg over trees with (x, y) of ratios "depth of the tree/max depth over all trees"
$7_{d,t}$	Cherry depth	Avg over trees with (x, y) of ratios "depth of (x, y) in the tree/depth of the tree"
$8_{d,t}$	Leaf distance	Avg over trees with x and y of ratios "x-y leaf distance/depth of the tree"
$9_{d,t}$	Leaf depth x	Avg over trees with x and y of ratios "root-x distance/depth of the tree"
$10_{d,t}$	Leaf depth y	Avg over trees with x and y of ratios "root-y distance/depth of the tree"
$11_{d,t}$	LCA distance	Avg over trees with x and y of ratios "x-LCA(x, y) distance/y-LCA(x, y) distance"
$12_{d,t}$	Depth x/y	Avg over trees with x and y of ratios "root-x distance/root-y distance"

To assign a class to each cherry, we define 19 features, summarized in Table 6.1, that may capture essential information about the structure of the set of trees, and that can be efficiently computed and updated at every iteration of the heuristics.

The *depth* (resp. *topological* depth) of a node u in a tree T is the total branch length (resp. the total number of edges) on the root-to-u path; the depth of a cherry (x, y) is the depth of the common parent of x and y; the depth of T is the maximum depth of any cherry of T.The (topological) leaf distance between x and y is the total branch length of the path from the parent of x to the lowest common ancestor of x and y, denoted by LCA(x, y), plus the total length of the path from the parent of y to LCA(x, y) (resp. the total number of edges on both paths). In particular, the leaf distance between the leaves of a cherry is zero.

6.4.1. TIME COMPLEXITY

Designing algorithms with the best possible time complexity was not the main objective of this work. However, for completeness, we provide worst-case upper bounds on the running time of our heuristics. We start by stating a general upper bound for the whole CPH framework in the function of the time required by the PickNext routine.

Lemma 6.4. *The running time of the heuristics in the CPH framework is* $\mathcal{O}(|\mathcal{T}|^2|X| + cost(\text{PickNext}))$, *where* $cost(\text{PickNext})$ *is the total time required to choose reducible pairs over all iterations. In particular, Rand takes* $\mathcal{O}(|\mathcal{T}|^2|X|)$ *time.*

Proof. An upper bound for the sequence length is $(|X| - 1)|\mathcal{T}|$ as each tree can individually be fully reduced using at most $|X| - 1$ pairs. Hence, the while loop of Algorithm 6.1 is executed at most $(|X| - 1)|\mathcal{T}|$ times. Moreover, reducing the pair and updating the set

of reducible pairs after one iteration takes $O(1)$ time per tree. Combining this with the fact that CompleteSeq takes $\mathcal{O}(|S|) = \mathcal{O}(|X||\mathcal{T}|)$ time, we obtain the stated time complexity. Since choosing a random reducible pair takes $\mathcal{O}(1)$ time at each iteration, Rand takes trivially $\mathcal{O}(|\mathcal{T}|^2|X|)$ time. □

Note that by Lemma 6.2 the number of reticulations $r(N)$ of the network reconstructed from the output CPS is bounded by $(|X|-1)|\mathcal{T}|-|X|+1 = \mathcal{O}(|\mathcal{T}|\cdot|X|)$, and thus the time complexity of Rand is also $\mathcal{O}(r(N)|\mathcal{T}|)$.

Let us now focus on the time complexity of the machine-learned heuristics ML and TrivialML. At any moment during the execution of the heuristics, we maintain a data structure that stores all the current cherries in $\mathcal{T}$ and allows constant-time insertions, deletions, and access to the cherries and their features. A possible implementation of this data structure consists of a hashtable cherryfeatures paired with a list cherrylist of the pairs currently stored in cherryfeatures. We will use cherrylist to iterate over the current cherries of $\mathcal{T}$, and cherryfeatures to check whether a certain pair is currently a cherry of $\mathcal{T}$ and to access its features.

The total number of cherries inserted in cherryfeatures over all the iterations is bounded by the total size of the trees $||\mathcal{T}||$ because up to two cherries can be created for each internal node over the whole execution. We will assume that we have constant-time access to the leaves of each tree: specifically, given $T \in \mathcal{T}$ and $x \in X$, we can check in constant time whether x is currently a leaf of T[1].

INITIALISATION

The cherries of $\mathcal{T}$ can be identified and features 1-3 can be initially computed in $\mathcal{O}(||\mathcal{T}||)$ time by traversing all trees bottom-up. Features 4-5 can be computed in $\mathcal{O}(\min\{|\mathcal{T}|\cdot||\mathcal{T}||, |\mathcal{T}|\cdot|X|^2\})$ time by checking, for each $T \in \mathcal{T}$ and each cherry (x,y) of $\mathcal{T}$, whether both x and y appear in T. Features $6_{d,t}$ to $12_{d,t}$ can also be initially computed with a traversal of $\mathcal{T}$ made efficient by preprocessing each tree in linear time to allow constant-time LCA queries [96] and by storing the depth (both topological and with the branch lengths) of each node. We also store the topological and branch length depth of each tree and their maximum value over $\mathcal{T}$. Altogether this gives the following lemma.

Lemma 6.5. *Initialising all features for a tree set $\mathcal{T}$ of total size $||\mathcal{T}||$ over a set of taxa X requires $\mathcal{O}(\min\{|\mathcal{T}|\cdot||\mathcal{T}||, |\mathcal{T}|\cdot|X|^2\})$ time and $\mathcal{O}(||\mathcal{T}||)$ space.*

The next lemma provides an upper bound on the time complexity of updating the distance-independent features. The proof is given in Appendix 6.A.

Lemma 6.6. *Updating features 1-5 for a set $\mathcal{T}$ of $|\mathcal{T}|$ trees of total size $||\mathcal{T}||$ over a set of taxa X requires $\mathcal{O}(|\mathcal{T}|(||\mathcal{T}||+|X|^2))$ total time and $\mathcal{O}(||\mathcal{T}||)$ space.*

Since searching for trivial cherries at each iteration of the randomized heuristic TrivialRand can be done with the same procedure we use for updating feature 4 in the machine-learned heuristics, which in particular requires $\mathcal{O}(|\mathcal{T}|\cdot||\mathcal{T}||)$ time, we have the following corollary.

[1] This can be obtained maintaining a list of leaves of each tree and a hashtable with the leaves as keys: the value of a key x is a pointer to the position of x in the list.

Corollary 6.1. *The time complexity of* TrivialRand *is* $\mathcal{O}(|\mathcal{T}| \cdot ||\mathcal{T}||) = \mathcal{O}(|\mathcal{T}|^2 \cdot |X|)$.

The total time required for updating the distance-dependent features raises the time complexity of ML and TrivialML to quadratic in the input size. However, the extensive analysis reported in Appendix 6.A shows that this is only due to the single feature 6_d, and without such a feature, the machine-learned heuristics would be asymptotically as fast as the randomized ones. Since Table 6.4 in Appendix 6.C shows that this feature is not particularly important, in future work it could be worth investigating whether disregarding it leads to equally good results in shorter time.

Lemma 6.7. *The time complexity of* ML and TrivialML *is* $\mathcal{O}(||\mathcal{T}||^2)$.

6.4.2. OBTAINING TRAINING DATA

The high-level idea to obtain training data is to first generate a phylogenetic network N; then to extract the set $\mathcal{T}$ of all the exhaustive trees displayed in N; and finally, to iteratively choose a random reducible pair (x, y) of N, to reduce it in $\mathcal{T}$ as well as in N, and to label the remaining cherries of $\mathcal{T}$ with one of the four classes defined in Section 6.4 until the network is fully reduced. Here, we assume that each reducible pair (x, y) of N can also be reduced in $\mathcal{T}$, as is the case for tree-child networks and restricted CPSs. As Janssen and Murakami [109, Figure 13] noticed, this is not necessarily the case for orchard networks and a regular CPS. Whenever we reach an iteration in which the reducible pairs of N are not reducible in $\mathcal{T}$, we terminate, but keep all previously obtained data.

We generate two different kinds of binary orchard networks, normal and not normal, with branch lengths and up to 9 reticulations using the LGT (lateral gene transfer) network generator of Pons et al. [169], imposing normality constraints when generating the normal networks. For each such network N, we then generate the set $\mathcal{T}$ consisting of all the exhaustive trees displayed in N. If N is normal, N is an optimal network for $\mathcal{T}$ Willson [218, Theorem 3.1]. This is not necessarily true for any LGT-generated network, but even in this case, we expect N to be reasonably close to optimal, because we remove redundant reticulations (i.e., reticulations that do not create any new displayed trees) when we generate it and because the trees in $\mathcal{T}$ cover all the edges of N. In particular, for LGT networks $r(N)$ provides an upper bound estimate on the minimum possible number of reticulations of any network displaying $\mathcal{T}$, and we will use it as a reference value for assessing the quality of our results on synthetic LGT-generated data.

6.5. EXPERIMENTS

The code of all our heuristics and for generating data is written in Python and is available at `https://github.com/estherjulien/learn2cherrypick`. All experiments ran on an Intel Xeon Gold 6130 CPU @ 2.1 GHz with 96 GB RAM. We conducted experiments on both synthetic and real data, comparing the performance of Rand, TrivialRand, ML and TrivialML, using threshold $\tau = 0$. Similar to the training data, we generated two synthetic datasets by first growing a binary orchard network N using Pons et al. [169], and then extracting $\mathcal{T}$ as a subset of the exhaustive trees displayed in N. We provide details on each dataset in Section 6.5.2.

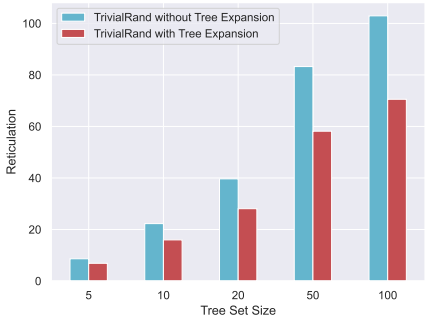


Figure 6.6: Number of reticulations output by TrivialRand with and without using tree expansion. The height of the bars is the average reticulation number over each group, obtained by selecting the best of 200 runs for each instance.

We start by analysing the usefulness of tree expansion, the heuristic rule described in Section 6.3.2. We synthetically generated 112 instances for each tree set size $|\mathscr{T}| \in \{5, 10, 20, 50, 100\}$ (560 in total), all consisting of trees with 20 leaves each, and grouped them by $|\mathscr{T}|$; we then ran TrivialRand 200 times (both with and without tree expansion) on each instance, selected the best output for each of them, and finally took the average of these results over each group of instances. The results are in Figure 6.6, showing that the use of tree expansion brought the output reticulation number down by at least 16% (for small instances) and up to 40% for the larger instances. We consistently chose to use this rule in all the heuristics that detect trivial cherries, namely, TrivialRand, TrivialML, ML (although ML does not explicitly favour trivial cherries, it does check whether a selected cherry is trivial using feature number 2), and the non-learned heuristic that will be introduced in Section 6.5.3.

6.5.1. PREDICTION MODEL

The random forest is implemented with Python's `scikit-learn` [166] package using default settings. We evaluated the performance of our trained random forest models on different datasets in a holdout procedure: namely, we removed 10% of the data from each training dataset, trained the models on the remaining 90% and used the holdout 10% for testing. The accuracy was assessed by assigning to each test data point the class with the highest predicted probability and comparing it with the true class. Before training the models, we balanced each dataset so that each class had the same number of representatives. Each training dataset differed in terms of the number M of networks used for generating it and the number of leaves of the networks. For each dataset, the number L of leaves of each generated network was uniformly sampled from $[2, \max L]$, where $\max L$ is the maximum number of leaves per network. We constructed LGT networks using the LGT generator of Pons et al. [169]. This generator has three parameters: n for the number of steps, α for the probability of lateral gene transfer events, and β for regulating the size of the biconnected components of the network (called *blobs*). The combination of these parameters determines the level (maximum number of reticulations per blob), the

number of reticulations, and the number of leaves of the output network. In our experiments, α was uniformly sampled from $[0.1, 0.5]$ and $\beta = 1$ (see Pons et al. [169] for more details).

To generate normal networks we used the same generator with the same parameters, but before adding a reticulation we check if it respects the normality constraints and only add it if it does. Each generated network gave rise to a number of data points: the total number of data points per dataset is shown in Table 6.3 in Appendix 6.B. Each row of Table 6.3 corresponds to a dataset on which the random forest can be trained, obtaining as many ML models. We tested all the models on all the synthetically generated instances: we show these results in Figures 6.18, 6.19 and 6.20 in Appendix 6.C. In Section 6.5.2 we will report the results obtained for the best-performing model for each type of instance. Among the advantages of using a random forest as a prediction model, there is the ability of computing feature importance, shown in Table 6.4 in Appendix 6.B. Some of the most useful features for a cherry (x, y) appear to be 'Trivial' (the ratio of the trees containing both leaves x and y in which (x, y) is a cherry) and 'Cherry in tree' (the ratio of trees that contain (x, y)). This was not unexpected, as these features are well-suited to identify trivial cherries.

'Leaf distance' (t,d), 'LCA distance' (t) and 'Depth x/y' (t) are also important features. The rationale behind these features was to try to identify reticulated cherries. This was also the idea for the feature 'Before/after', but this has, surprisingly, a very low importance score. In future work, we plan to conduct a thorough analysis of whether some of the seemingly least important features can be removed without affecting the quality of the results.

6.5.2. EXPERIMENTAL RESULTS

We assessed the performance of our heuristics on instances of four types: normal, LGT, ZODS (binary non-orchard networks), and real data. Normal, LGT and ZODS data are synthetically generated. We generated the normal instances as we did for the training data: we first grew a normal network using the LGT generator and then extracted all the exhaustive trees displayed in the network. We generated normal data for different combinations of the following parameters: $L \in \{20, 50, 100\}$ (number of leaves per tree) and $R \in \{5, 6, 7\}$ (reticulation number of the original network). Note that, for normal instances, $|\mathscr{T}| = 2^R$. For every combination of the parameters L and R we generated 48 instances: by *instance group* we indicate the set of instances generated for one specific parameter pair.

For the LGT instances, we grew the networks using the LGT generator, but unlike for the normal instances we then extracted only a subset of the exhaustive trees from each of them, up to a certain amount $|\mathscr{T}| \in \{20, 50, 100\}$. The other parameters for LGT instances are the number of leaves $L \in \{20, 50, 100\}$ and the number of reticulations $R \in \{10, 20, 30\}$. For a fixed pair $(L, |\mathscr{T}|)$, we generated 16 instances for each possible value of R, and analogously, for a fixed pair (L, R) we generated 16 instances for each value of $|\mathscr{T}|$. The 48 instances generated for a fixed pair of values constitute a LGT instance group.

We generated non-orchard binary networks using the ZODS generator [228]. This generator has two user-defined parameters: λ, which regulates the speciation rate, and ν, which regulates the hybridization rate. Following Janssen and Liu [108] we set $\lambda = 1$

Table 6.2: Number of real data instances for each group (combination of parameters L and $|\mathcal{T}|$).

L	$\|\mathcal{T}\|$				Tot. Trees
	10	20	50	100	
20	50	50	50	50	1684
50	20	20	20	20	290
100	5	5	1	0	53

and we sampled $\nu \in [0.0001, 0.4]$ uniformly at random. Like for the LGT instances, we generated an instance group of size 48 for each pair of values $(L, |\mathcal{T}|)$ and (L, R), with $L \in \{20, 50, 100\}$, $|\mathcal{T}| \in \{20, 50, 100\}$, $R \in \{10, 20, 30\}$.

Finally, the real-world dataset consists of gene trees on homologous gene sets found in bacterial and archaeal genomes, was originally constructed in Beiko [20] and made binary in van Iersel et al. [201]. We extracted a subset of instances (Table 6.2) from the binary dataset, for every combination of parameters $L \in \{20, 50, 100\}$ and $|\mathcal{T}| \in \{10, 20, 50, 100\}$.

For the synthetically generated datasets, we evaluated the performance of each heuristic in terms of the output number of reticulations, comparing it with the number of reticulations of the network N from which we extracted $\mathcal{T}$. For the normal instances, N is the optimal network [218, Theorem 3.1]; this is not true, in general, for the LGT and ZODS datasets, but even in these cases, $r(N)$ clearly provides an estimate (from above) of the optimal value, and thus we used it as a reference value for our experimental evaluation.

For real data, in the absence of the natural estimate on the optimal number of reticulations provided by the starting network, we evaluated the performance of the heuristics comparing our results with the ones given by the exact algorithms from van Iersel et al. [201] (TreeChild) and from Albrecht [3] (Hybroscale), using the same datasets that were used to test the two methods in van Iersel et al. [201]. These datasets consist of rather small instances ($|\mathcal{T}| \leq 8$); for larger instances, we run TrivialRand 1000 times for each instance group, selected the best result for each group, and used it as a reference value (Figure 6.10).

We now describe in detail the results we obtained for each type of data and each of the algorithms we tested.

EXPERIMENTS ON NORMAL DATA

For the experiments in this section we used he ML model trained on 1000 normal networks with at most 100 leaves per network (see Figure 6.18 in Appendix 6.C). We ran the machine-learned heuristics once for each instance and then averaged the results within each instance group (recall that one instance group consists of the sets of all the exhaustive trees of 48 normal networks having the same fixed number of leaves and reticulations). The randomized heuristics Rand and TrivialRand were run $\min\{x(I), 1000\}$ times for each instance I, where $x(I)$ is the number of runs that can be executed in the same time as one run of ML on the same instance. We omitted the results for LowPair because they were at least 44% worse on average than the worst-performing heuristic we report.

In Figure 6.7 we summarize the results. Solid bars represent the ratio between the

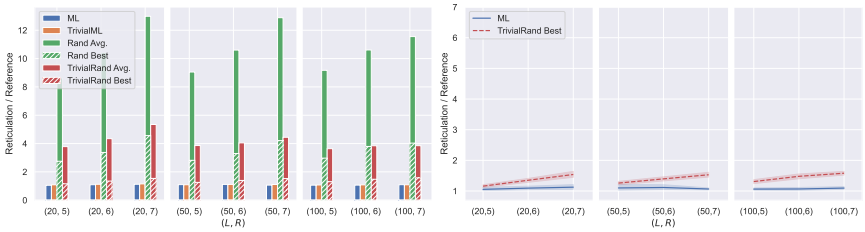


Figure 6.7: Experimental results for normal data. Each point on the horizontal axis corresponds to one instance group. In the left graph, the height of each bar gives the average of the results over all instances of the group, scaled by the optimum value for the group. The right graph compares the average output of ML within each instance group and the average of the best output given by TrivialRand for each instance of a group. The shaded areas represent 95% confidence intervals.

average reported reticulation number and the optimal value, for each instance group and for each of the four heuristics. Dashed bars represent the ratio between the average (over the instances within each group) of the best result among the $\min\{x(I), 1000\}$ runs for each instance I and the optimum. The machine-learned heuristics ML and TrivialML seem to perform very similarly, both leading to solutions close to optimum. The average performance of TrivialRand is around 4 times worse than the machine-learned heuristics; in contrast, if we only consider the best solution among the multiple runs for each instance, they are quite good, having only up to 49% more reticulations than the optimal solution, but they are still at least 4% worse (29% worse on average) than the machine-learned heuristics' solutions: see the right graph of Figure 6.7. The left graph of Figure 6.7 shows that the performance of the randomized heuristics seems to be negatively impacted by the number of reticulations of the optimal solution, while we do not observe a clear trend for the machine-learned heuristics, whose performance is very close to optimum for all the considered instance groups. Indeed, the number of existing phylogenetic networks with a certain number of leaves grows exponentially in the number of reticulations, thus making it less probable to reconstruct a "good" network with random choices. This is consistent with the existing exact methods being FPT in the number of reticulations [201, 216]. The fully randomized heuristic Rand always performed much worse than all the others, indicating that identifying the trivial cherries has a great impact on the effectiveness of the algorithms (recall that ML implicitly identifies trivial cherries).

EXPERIMENTS ON LGT DATA

For the experiments on LGT data we used the ML model trained on 1000 LGT networks with at most 100 leaves per network (see Figure 6.19 in Appendix 6.C). The setting of the experiments is the same as for the normal data (we run the randomized heuristics multiple times and the machine-learned heuristics only once for each instance), with two important differences. First, for LGT data we only take proper subsets of the exhaustive trees displayed by the generating networks, and thus we have two kinds of instance groups: one where in each group the number of trees extracted from a network and the number of leaves of the networks are fixed, but the trees come from networks with different numbers of reticulations; and one where the number of reticulations of the gen-

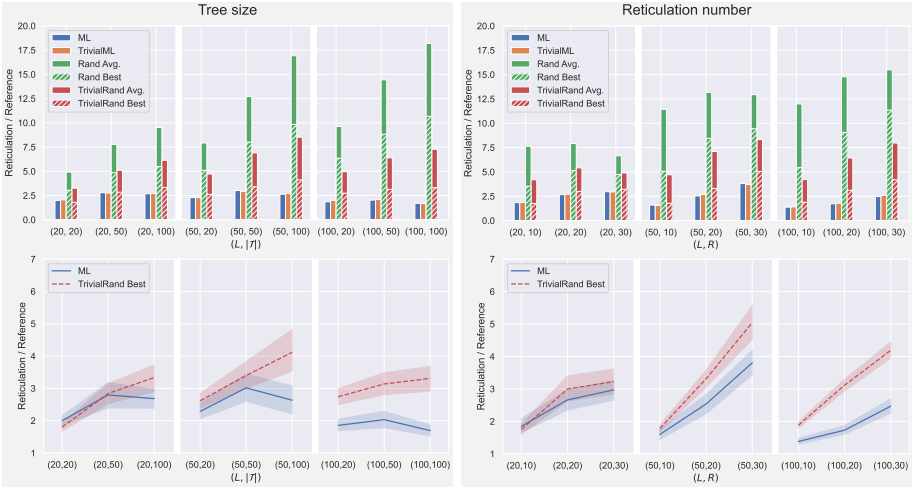


Figure 6.8: Experimental results for LGT data. Each point on the horizontal axis corresponds to one instance group. For the graphs on the left, there is one group for each fixed pair $(L, |\mathcal{T}|)$ consisting of 16 instances coming from LGT networks for each value of $R \in \{10, 20, 30\}$. For the graphs on the right, there is one group for each fixed pair (L, R) consisting of 16 instances coming from LGT networks for each value of $|\mathcal{T}| \in \{20, 50, 100\}$. In the top graphs, the height of each bar gives the average of the results over all instances of the group, each scaled by the number of reticulations of the generating network. The bottom graphs compare the average output of ML within each instance group and the average of the best output given by TrivialRand for each instance group. The shaded areas represent 95% confidence intervals.

erating networks and their number of leaves are fixed, but the number of trees extracted from a network varies. The second important difference is that the reference value we use for LGT networks is not necessarily the optimum, but it is just an upper bound given by the number of reticulations of the generating networks which we expect to be reasonably close to the optimum (see Section 6.4.2). The results for the LGT datasets are shown in Figure 6.8. Comparing these results with those of Figure 6.7, it is evident that the LGT instances were more difficult than the normal ones for all the tested heuristics: this could be because the normal instances consisted of all the exhaustive trees of the generating networks, while the LGT instances only have a subset of them and thus carry less information.

The machine-learned heuristics performed substantially better (up to 80% on average) than the best randomized heuristic TrivialRand in all instance groups but the ones with the smallest values for parameters R, $|\mathcal{T}|$ and L, for which the performances are essentially overlapping. On the contrary, the advantage of the machine-learned methods is more pronounced when the parameters are set to the highest values. This is because the larger the parameters, the more the possible different networks that embed $\mathcal{T}$, thus the less likely for the randomized methods to find a good solution. From the graphs on the right of Figure 6.8, it seems that the number of reticulations has a negative impact on both machine-learned and randomized heuristics, the effect being more pronounced for the randomized ones. The effect of the number of trees $|\mathcal{T}|$ on the quality of the solutions is not as clear (Figure 6.8, left). However, we can still see that the trend of ML and TrivialRand is the same: the “difficult" instance groups are so for both heuristics, even

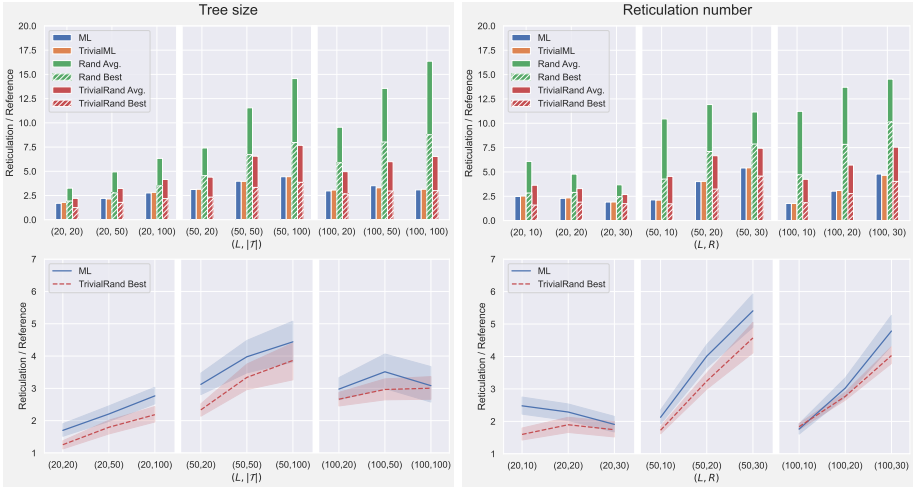


Figure 6.9: Experimental results for ZODS data. Each point on the horizontal axis corresponds to one instance group. For the graphs on the left, there is one group for each fixed pair $(L, |\mathcal{T}|)$ consisting of 16 instances coming from ZODS networks for each value of $R \in \{10, 20, 30\}$. For the graphs on the right, there is one group for each fixed pair (L, R) consisting of 16 instances coming from ZODS networks for each value of $|\mathcal{T}| \in \{20, 50, 100\}$. In the top graphs, the height of each bar gives the average of the results it represents over all instances of the group, each scaled by the number of reticulations of the network the instance originated from. The bottom graphs compare the average output of ML within each instance group and the average of the best output given by TrivialRand for each group instance. The shaded areas represent 95% confidence intervals.

if the degradation in the quality of the solutions for such instance groups is less marked for ML than for TrivialRand.

EXPERIMENTS ON ZODS DATA

For the experiments on ZODS data we used the ML model trained on 1000 LGT networks with at most 100 leaves per network (see Figure 6.20 in Appendix 6.C). The setting of the experiments is the same as for the LGT data, and the results are shown in Figure 6.9. At first glance, the performance of the randomized heuristics seems to be better for ZODS data than for LGT data (compare figures 6.8 and 6.9), which sounds counterintuitive. Recall, however, that all the graphs show the ratio between the number of reticulations returned by our methods and a reference value, i.e., the number of reticulations of the generating network: while we expect this reference to be reasonably close to the optimum for LGT networks, this is not the case for ZODS networks. In fact, a closer look to ZODS networks shows that they have a large number of redundant reticulations which could be removed without changing the set of trees they display, and thus their reticulation number is in general quite larger than the optimum. This is an inherent effect of the ZODS generator not having any constraints on the reticulations that can be introduced, and it is more marked on networks with a small number of leaves. Having a reference value significantly larger than the optimum makes the ratios shown in Figure 6.9 small (close to 1, especially for TrivialRand on small instances) without implying that the results for the ZODS data are better than the ones for the LGT data. The graphs of Figures 6.8 and 6.9 are thus not directly comparable. The reference value for the ex-

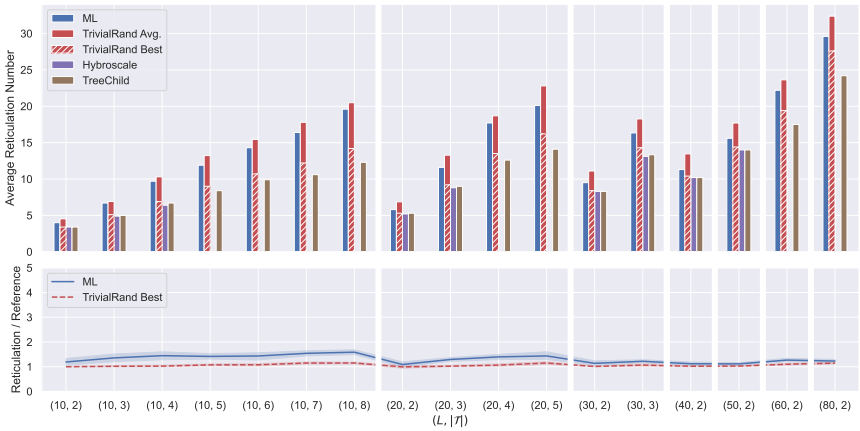


Figure 6.10: Comparison of ML, TrivialRand, Hybroscale, and TreeChild on real data. Each point on the horizontal axis corresponds to one instance group, consisting of 10 instances for a fixed pair $(L, |\mathcal{T}|)$. In the top graph, the height of each bar gives the average, over all instances of the group, of the number of reticulations returned by the method. The bottom graphs compare the average output of ML within each instance group and the average of the best output given by TrivialRand within the group. The shaded areas represent 95% confidence intervals.

6

periments on ZODS data not being realistically close to the optimum, however, does not invalidate their significance. Indeed, the scope of such experiments was just to compare the performance of the machine-learned heuristics on data entirely different from those they were trained on with the performance of the randomized heuristics, which should not depend on the type of network that was used to generate the input. As expected and in contrast with normal and LGT data, the results show that the machine-learned heuristics perform worse than the randomized ones on ZODS data, consistent with the ML methods being trained on a completely different class of networks.

EXPERIMENTS ON REAL DATA

We conducted two sets of experiments on real data, using the ML model trained on the dataset trained on 1000 LGT networks with at most 100 leaves each. For sufficiently small instances, we compared the results of our heuristics with the results of two existing tools for reconstructing networks from binary trees: TreeChild [201] and Hybroscale [3]. Hybroscale is an exact method performing an exhaustive search on the networks displaying the input trees, therefore it can only handle reasonably small instances in terms of the number of input trees. TreeChild is a fixed-parameter (in the number of reticulations of the output) exact algorithm that reconstructs the best *tree-child* network, a restricted class of phylogenetic networks, and due to its fast-growing computation time cannot handle large instances either.

We tested ML and TrivialRand against Hybroscale and TreeChild using the same dataset used in van Iersel et al. [201], in turn taken from Beiko [20]. The dataset consists of ten instances for each possible combination of the parameters $|\mathcal{T}| \in [2,8]$ and $L \in \{10,20,30,40,50,60,80,100,150\}$. In Figure 6.10 we show results only for the instance groups for which Hybroscale or TreeChild could output a solution within 1 hour, con-

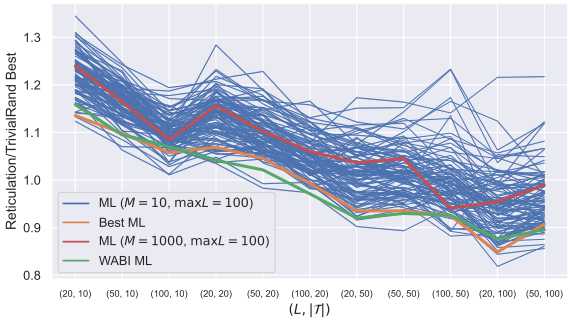
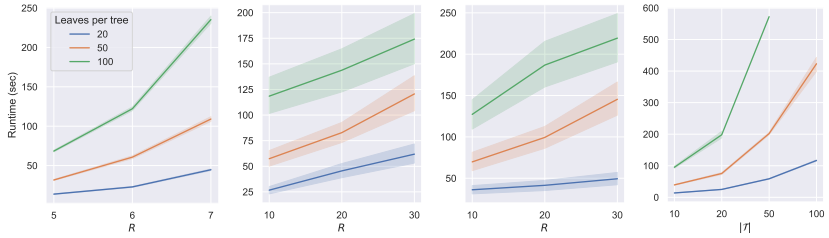


Figure 6.11: Ratio between the performance of ML and the best value output by TrivialRand for different instance groups and different training sets. TrivialRand is executed $\min\{x(I), 1000\}$ times for each instance I, $x(I)$ being the number of runs that could be completed in the same time as one run of ML on I. The results are then averaged within each group. Each blue line represents the results obtained training the model with a different set of 10 randomly generated LGT networks with at most 100 leaves each. The green line corresponds to the training set used in Bernardini et al. [27]; the orange line represents one of the best-performing sets; the red line corresponds to the training set we used for the experiments on LGT and ZODS data in this chapter, consisting of 1000 randomly generated LGT networks.

sistent with the experiments in van Iersel et al. [201]. As a consequence of Hybroscale and TreeChild being exact methods (TreeChild only for a restricted class of networks), they performed better than both ML and TrivialRand on all instances they could solve, although the best results of TrivialRand are often close (no worse than 15%) and sometimes match the optimal value. The main advantage of our heuristics is that they can handle much larger instances than the exact methods. In the conference version of this work [27] we showed the results of our heuristics on large real instances, using a ML model trained on 10 networks with at most 100 leaves each. These results demonstrated that consistently with the simulated data, the machine-learned heuristics gave significantly better results than the randomized ones for the largest instances. When we first repeated the experiments with the new models trained on 1000 networks with max$L$ = 100, however, we did not obtain similar results: instead, the results of the randomized heuristics were better or only marginally worse than the machine-learned ones on almost all the instance groups, including the largest. Puzzled by these results, we conducted an experiment on the impact of the training set on real data. The results are reported in Figure 6.11, and show that the choice of the networks on which we train our model has a big impact on the quality of the results for the real datasets. This is in contrast with what we observed for the synthetic datasets, for which only the class of the training networks was important, not the specific instances of the networks themselves. According to what was noted in van Iersel et al. [201], this is most likely due to the fact that the real phylogenetic data have substantially more structure than random synthetic datasets, and the randomly generated training networks do not always reflect this structure. By chance, the networks we used for training the model we used in Bernardini et al. [27] were similar to real phylogenetic networks, unlike the 1000 networks in the training set of this chapter.



(a) Normal (b) LGT (c) ZODS (d) Real

Figure 6.12: The running time (in seconds) of ML for the instance groups described in Section 6.5.2. The solid lines represent the average of the running times for the instances within each instance group. The shaded areas represent 95% confidence intervals.

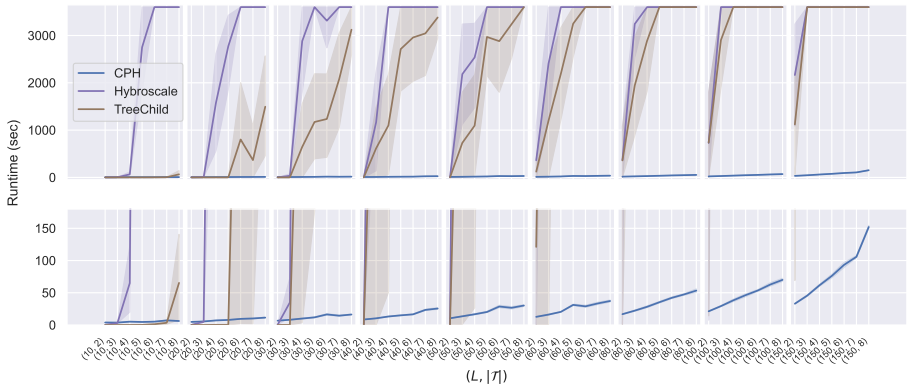


Figure 6.13: The running time of ML on the real dataset described in Section 6.5.2 compared with the running time of the exact methods Hybroscale and TreeChild on the same dataset. The solid lines represent the average running times within each instance group. The shaded areas represent 95% confidence intervals.

EXPERIMENTS ON SCALABILITY

We conducted experiments to study how the running time of our heuristics scales with increasing instance size for all datasets. In Figure 6.12 we report the average of the running times of ML for the instances within each instance group with a 95% confidence interval, for an increasing number of reticulations (synthetic datasets) or number of trees (real dataset). The datasets and the instance groups are those described in the previous sections. Note that we did not report the running times of the randomized heuristics because they are meant to be executed multiple times on each instance, and in all the experiments we bounded the number of executions precisely using the time required for one run of ML.

We also compared the running time of our heuristics with the running times of the exact methods TreeChild and Hybroscale. The results are shown in Figure 6.13 and are consistent with the execution times of the exact methods growing exponentially, while the running time of our heuristics grows polynomially. Note that networks with more reticulations are reduced by longer CPS and thus the running time increases with the number of reticulations.

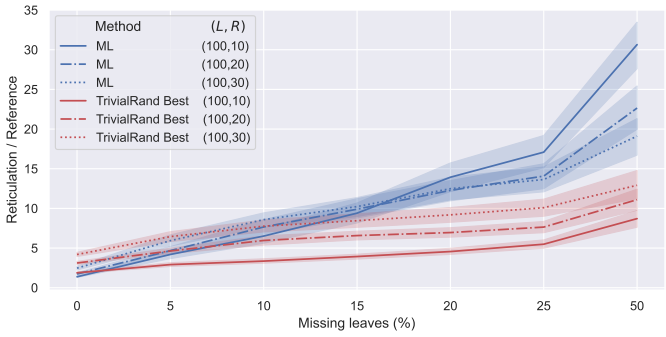


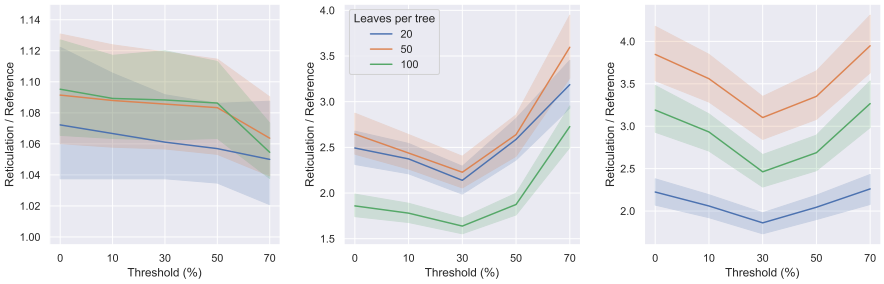
Figure 6.14: Ratio between the number of reticulations outputted by ML and TrivialRand Best and the reference value for an increasing percentage of removed leaves on LGT data. Each point on the horizontal axis corresponds to a certain percentage of leaves removed from each tree; each line represents the average, within the instances of a group (L, R) with a certain percentage of removed leaves, of the output reticulation number divided by the reference value. The shaded areas represent 95% confidence intervals.

EXPERIMENTS ON NON-EXHAUSTIVE INPUT TREES

The instances on which we tested our methods so far all consisted of a set of exhaustive trees, that is, each input tree had the same set of leaves which coincided with the set of leaves of the network. However, this is not a requirement of our heuristics, which are able to produce feasible solutions also when the leaf sets of the input trees are different, that is when their leaves are proper subsets of the leaves of the optimal networks that display them. To test their performance on this kind of data, we generated 18 LGT instance groups starting from the instances we used in Section 6.5.2 and removing a certain percentage p of leaves from each tree in each instance uniformly at random. Specifically, we generated an instance group for each value of $p \in \{5, 10, 15, 20, 25, 50\}$ starting from the LGT instance groups with $L = 100$ leaves and $R \in \{10, 20, 30\}$ reticulations. Since the performances of the two machine-learned heuristics were essentially overlapping for all of the other experiments, and since TrivialRand performed consistently better than the other randomized heuristics, we limited this test to ML and TrivialRand. The results are shown in Figure 6.14. In accordance with intuition, the performance of both methods decreases with an increasing percentage of removed leaves, as the trees become progressively less informative. However, the degradation in the quality of the solutions is faster for ML than for TrivialRand, consistent with the fact that ML was trained on exhaustive trees only: when the difference between the training data and the input data becomes too large, the behavior of the machine-learned heuristic becomes unpredictable. We demand the design of algorithms better suited for trees with missing leaves for future work.

EFFECT OF THE THRESHOLD ON ML

We tested the effectiveness of adding a threshold $\tau > 0$ to ML on the same datasets of Sections 6.5.2, 6.5.2 and 6.5.2 (normal, LGT and ZODS). Recall that each instance group consists of 48 instances. We ran ML ten times for each threshold $\tau \in \{0, 0.1, 0.3, 0.5, 0.7\}$ on each instance, took the lowest output reticulation number and averaged these results within each instance group.



(a) Normal (b) LGT (c) ZODS

Figure 6.15: The reticulation number when running ML with different thresholds on the instance groups of Sections 6.5.2, 6.5.2 and 6.5.2. Each instance was run 10 times, and the lowest reticulation value of these runs was selected. The shaded areas represent 95% confidence intervals.

The results are shown in Figure 6.15. For all types of data, a threshold $\tau \leq 0.3$ is beneficial, intuitively indicating that when the probability of a pair being reducible is small it gives no meaningful indication, and thus random choices among these pairs are more suited. The seemingly best value for the threshold, though, is different for different types of instances. The normal instances seem to benefit from quite high values of τ, the best among the tested values being $\tau = 0.7$. While the optimal τ value for normal instances could be even higher, we know from Figure 6.7 that it must be $\tau < 1$, as the random strategies are less effective than the one based on machine learning for normal data. For the LGT and the ZODS instances, the best threshold seems to be around $\tau = 0.3$, while very high values ($\tau = 0.7$) are counterproductive. This is especially true for the LGT instances, consistent with the randomized heuristics being less effective for them than for the other types of data (see Figure 6.8).

These experiments should be seen as an indication that introducing some randomness may improve the performance of the ML heuristics, at the price of running them multiple times. We defer a more thorough analysis to future work.

6.5.3. A NON-LEARNED HEURISTIC BASED ON IMPORTANT FEATURES

In this section we propose FeatImp, yet another heuristic in the CPH framework. Although FeatImp does not rely on a machine-learning model, we defined the rules to choose a cherry on the basis of the features that were found to be the most relevant according to the model we used for ML and TrivialML.

To identify the most suitable rules, we trained a classification tree using the same features and training data as the ones used for the ML heuristic (see figure 6.17 in Appendix 6.B). We then selected the most relevant features used in such tree and used them to define the function PickNext listed by Algorithm 6.3: namely, the features 4, 8_t, 11_d and 12_t of Table 6.1 (the ratio of trees having both leaves x and y in which (x, y) is reducible, the average of the topological leaf distance between x and y scaled by the depth of the trees, the average of the ratios $d(x, \mathsf{LCA}(x, y))/d(y, \mathsf{LCA}(x, y))$ and the average of the topological distance from x to the root over the topological distance from y to the root, respectively).

To compute and update these quantities we proceed as described in Section 6.4.1. The general idea of the function PickNext used in FeatImp is to mimic the first splits of the classification tree by progressively discarding the candidate reducible pairs that are not among the top α% scoring for each of the considered features, for some input parameter α.

Algorithm 6.3: Function PickNext used in FeatImp

Input : A set $\mathscr{T}$ of phylogenetic trees and a parameter $\alpha \in (0,100)$
Output : Next cherry to pick (x,y).

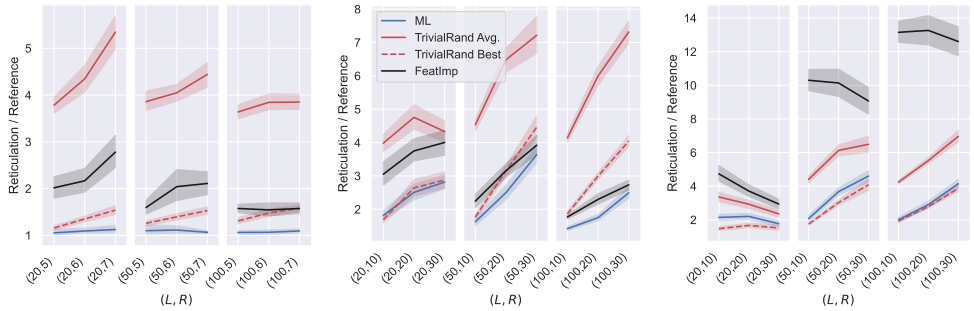
1 **if** *there exists a trivial cherry* **then**
2 Select a trivial cherry (x,y) uniformly at random
3 **else**
4 $C \leftarrow$ all reducible pairs of $\mathscr{T}$
5 $C \leftarrow$ the α% cherries of C with the highest value for feature 4
6 $C \leftarrow$ the α% cherries of C with the highest value for feature 8_t
7 $C \leftarrow$ the α% cherries of C with the highest value for feature 11_d
8 $(x,y) \leftarrow$ the pair of C with the highest value for feature 12_t
9 **end**
10 **return** (x,y)

We implemented FeatImp and test it on the same instances as sections 6.5.2, 6.5.2 and 6.5.2 with $\alpha = 20$. The results are shown in Figure 6.16. As expected, FeatImp works consistently worse than ML on all the tested datasets, and it also performs worse than TrivialRand on most instance groups. However, it is on average 12% better than TrivialRand on the LGT instance group having 50 leaves and 30 reticulations and on all the LGT instance groups with 100 leaves, which are the most difficult for the randomized heuristics, as already noticed in Section 6.5.2. The results it provides for such difficult instances are only on average 20% worse than those of ML, with the advantage of not having to train a model to apply the heuristic.

These experiments are not intended to be exhaustive, but should rather be seen as an indication that machine learning can be used as a guide to design smarter non-learned heuristics. Possible improvements of FeatImp include using different values of α for different features, introducing some randomness in Line 8, that is, instead of choosing the single top scoring pair to choose one among the top α% at random, or to use fewer/more features.

6.6. CONCLUSIONS

Our contributions are twofold: first, we presented the first methods that allow reconstructing a phylogenetic network from a large set of large binary phylogenetic trees. Second, we show the promise and the limitation of the use of machine learning in this context. Our experimental studies indicate that machine-learned strategies, consistent with intuition, are very effective when the training data have a structure similar enough to the test data. In this case, the results we obtained with machine learning were the best among all the tested methods, and the advantage is particularly evident in the most difficult instances. Furthermore, preliminary experiments indicate that the performance of the machine-learned methods can even be improved by introducing appropriate thresh-



(a) Normal (b) LGT (c) ZODS

Figure 6.16: Comparison of the results of FeatImp, ML and TrivialRand on the instance groups described in Sections 6.5.2, 6.5.2 and 6.5.2. Each point on the horizontal axis corresponds to an instance group; each line represents the average, within the instance group, of the output reticulation number divided by the reference value. The shaded areas represent 95% confidence intervals.

olds, in fact mediating between random choices and predictions. However, when the training data do not sufficiently reflect the structure of the test data, repeated runs of the fast randomized heuristics lead to better results. The non-learned cherry-picking heuristic we designed based on the most relevant features of the input (identified using machine learning) shows yet another interesting direction.

Our results suggest many interesting directions for future work. First of all, we have seen that machine learning is an extremely promising tool for this problem since it can identify cherries and reticulated cherries of a network, from displayed trees, with very high accuracy. It would be interesting to prove a relationship between the machine-learned models' accuracy and the produced networks' quality. In addition, do there exist algorithms that exploit the high accuracy of the machine-learned models even better? Could other machine-learning methods than random forests, or more training data, lead to even better results? Our methods are applicable to trees with missing leaves but perform well only if the percentage of missing leaves is small. Can modified sets of features be defined that are more suitable for input trees with many missing leaves? Moreover, we have seen that combining randomness with machine learning can lead to better results than either individual approach. However, we considered only one strategy to achieve this. What are the best strategies for combining randomness with machine learning for this, and other, problems? From a practical point of view, it is important to investigate whether our methods can be extended to deal with nonbinary input trees and to develop efficient implementations: in fact, we point out that our current implementations are in Python and not optimized for speed. Faster implementations could make machine-learned heuristics with nonzero thresholds even more effective. Finally, can the machine-learning-based approach be adapted to other problems in the phylogenetic networks research field?

APPENDIX OF CHAPTER 6

6.A. TIME COMPLEXITY

▷ **Lemma 6.6.** *Updating features 1-5 for a set $\mathscr{T}$ of $|\mathscr{T}|$ trees of total size $||\mathscr{T}||$ over a set of taxa X requires $\mathcal{O}(|\mathscr{T}|(||\mathscr{T}|| + |X|^2))$ total time and $\mathcal{O}(||\mathscr{T}||)$ space.*

Proof. Let $F^i_{(x,y)}$ denote the current value of the i-th feature for a cherry (x,y). When reducing a cherry (x,y) in a tree T (thus deleting x and $p(x) = p(y)$ and then adding a direct edge from $p(p(y))$ to y), we check whether the other child of $p(p(y))$ is a leaf z or not. If not, no new cherry is created in T, thus the features 1-4 remain unaffected for all the cherries of $\mathscr{T}$. Otherwise, (z,y) and (y,z) are new cherries of T and we can distinguish two cases.

1. (z,y) and (y,z) are already cherries of $\mathscr{T}$. Then, $F^1_{(y,z)}$ and $F^1_{(z,y)}$ are increased by $\frac{1}{|\mathscr{T}|}$; $F^4_{(y,z)}$ and $F^4_{(z,y)}$ are increased by $\frac{1}{|\mathscr{T}^{y,z}|}$, where $|\mathscr{T}^{y,z}|$ is the number of trees that contain both y and z and is equal to $|\mathscr{T}|F^5_{(y,z)}$. To update features 2 and 3 we use two auxiliary data structures $\mathsf{new\_cherries}_{(y,z)}$ and $\mathsf{new\_cherries}_{(z,y)}$ to collect the distinct cherries that would originate after picking (y,z) and (z,y) in each tree, respectively. These structures must allow efficient insertions, membership queries, and iteration over the elements[2], and can be deleted before picking the next cherry in $\mathscr{T}$. If the other child of $p(p(z))$ is a leaf w, we add (z,w) and (w,z) to $\mathsf{new\_cherries}_{(y,z)}$ and (y,w) and (w,y) to $\mathsf{new\_cherries}_{(z,y)}$ (unless they are already present).

2. (z,y) and (y,z) are new cherries of $\mathscr{T}$. Then we insert them into $\mathsf{cherryfeatures}$. We initially set $F^1_{(y,z)} = F^1_{(z,y)} = \frac{1}{|\mathscr{T}|}$, and for features 2-3 we create the same data structures as the previous case. To compute $F^5_{(y,z)} = F^5_{(z,y)}$ we first compute $|\mathscr{T}^{y,z}|$ by checking whether y and z are both leaves of T for each $T \in \mathscr{T}$. Then we set $F^5_{(y,z)} = F^5_{(z,y)} = \frac{|\mathscr{T}^{y,z}|}{|\mathscr{T}|}$ and $F^4_{(y,z)} = F^4_{(z,y)} = \frac{1}{|\mathscr{T}^{y,z}|}$.

Once we have reduced (x,y) in all trees, we count the elements of each of the auxiliary data structures $\mathsf{new\_cherries}$ and update features 2-3 of the corresponding cherries accordingly. Since picking a cherry can create up to two new cherries in each tree, and for each new cherry we add up to two elements to an auxiliary data structure, this step requires $\mathcal{O}(|\mathscr{T}|)$ time for each iteration.

Feature 5 must be updated for all the cherries corresponding to the unordered pairs $\{x,w\}$ with $w \neq y$. To do so, when we reduce (x,y) in a tree T we go over its leaves: for each leaf $w \neq y$ we decrease $F^5_{(x,w)}$ and $F^5_{(w,x)}$ by $\frac{1}{|\mathscr{T}|}$ (if (x,w) and (w,x) are currently cherries of $\mathscr{T}$). This requires $\mathcal{O}(|X|^2)$ total time per tree over all the iterations, because

[2] For example, hashtables paired with lists.

we scan the leaves of a tree only when we reduce a cherry in that tree. Computing feature 5 when new cherries of $\mathcal{T}$ are created (case 2) requires constant time per tree per cherry. The total number of cherries created in $\mathcal{T}$ over all the iterations cannot exceed $2||\mathcal{T}||$, thus the total time required to update feature 5 is $\mathcal{O}(|\mathcal{T}|(||\mathcal{T}|| + |X|^2))$. We arrived at the following result. □

▷ **Lemma 6.7.** *The time complexity of* ML *and* TrivialML *is* $\mathcal{O}(||\mathcal{T}||^2)$.

Proof. Recall that during the initialization phase, we store the depth of each node, both topological and with respect to the branch lengths, and we preprocess each tree to allow constant-time LCA queries. Note that reducing cherries in the trees does not affect the height of the nodes nor their ancestry relations, thus it suffices to preprocess the tree set only once at the beginning of the algorithm.

When we reduce a cherry (x, y) in a tree T, this may affect the depth of T as a consequence of the internal node $p(x)$ being deleted. We thus visit T to update its depth (both topological and with the branch lengths), and after updating the depth of all trees, we update the maximum value over the whole set $\mathcal{T}$ accordingly. In order to describe how to update the features $6_{d,t} - 12_{d,t}$ we denote by $\mathsf{old\_depth}^t(T)$ the topological depth of T before reducing (x, y), $\mathsf{new\_depth}^t(T)$ its depth after reducing (x, y), and use analogous notation for the distances $\mathsf{old\_dist}^t$ and $\mathsf{new\_dist}^t$ between two nodes of a tree and for the depth, the max depth, and distances with the branch lengths.

Whenever the value of the maximum topological depth changes, we update the value of feature 6_t for all the current cherries (z, w) as $F^{6_t}_{(z,w)} = \frac{F^{6_t}_{(z,w)} \cdot \mathsf{old\_max\_depth}^t}{\mathsf{new\_max\_depth}^t}$. Since the maximum topological depth can change $\mathcal{O}(|X|)$ times over all the iterations, and the total number of cherries at any moment is $\mathcal{O}(|\mathcal{T}||X|)$, these updates require $\mathcal{O}(|\mathcal{T}||X|^2)$ total time. We do the same for feature 6_d, but since the maximum branch-length depth can change once per iteration in the worst case, this requires $\mathcal{O}(||\mathcal{T}||^2)$ time overall.

Features $8_{d,t} - 12_{d,t}$ must be then updated to remove the contribution of T for the cherries (x, w) and (w, x) for each leaf $w \neq x \neq y$ of T, because x and w will no longer appear together in T. These updates require $\mathcal{O}(1)$ time per leaf and can be done as follows. We set

$$F^{8_t}_{(x,w)} = \frac{F^{8_t}_{(x,w)} \cdot |\mathcal{T}^{x,w}| - \frac{\mathsf{old\_dist}^t(x,w)}{\mathsf{old\_depth}^t(T)}}{|\mathcal{T}^{x,w}| - 1} \tag{6.1}$$

and use analogous formulas to update $F^{8_d}_{(x,w)}$ and features $9_{d,t} - 12_{d,t}$ for (x, w) and (w, x).

We finally need to further update all the features $6_{d,t} - 12_{d,t}$ for all the cherries of a tree T in which (x, y) has been reduced and whose depth has changed, including the newly created ones. This can be done in $\mathcal{O}(1)$ time per cherry per tree with opportune formulas of the form of (6.1). We have obtained the stated bound. □

6.B. RANDOM FOREST MODELS

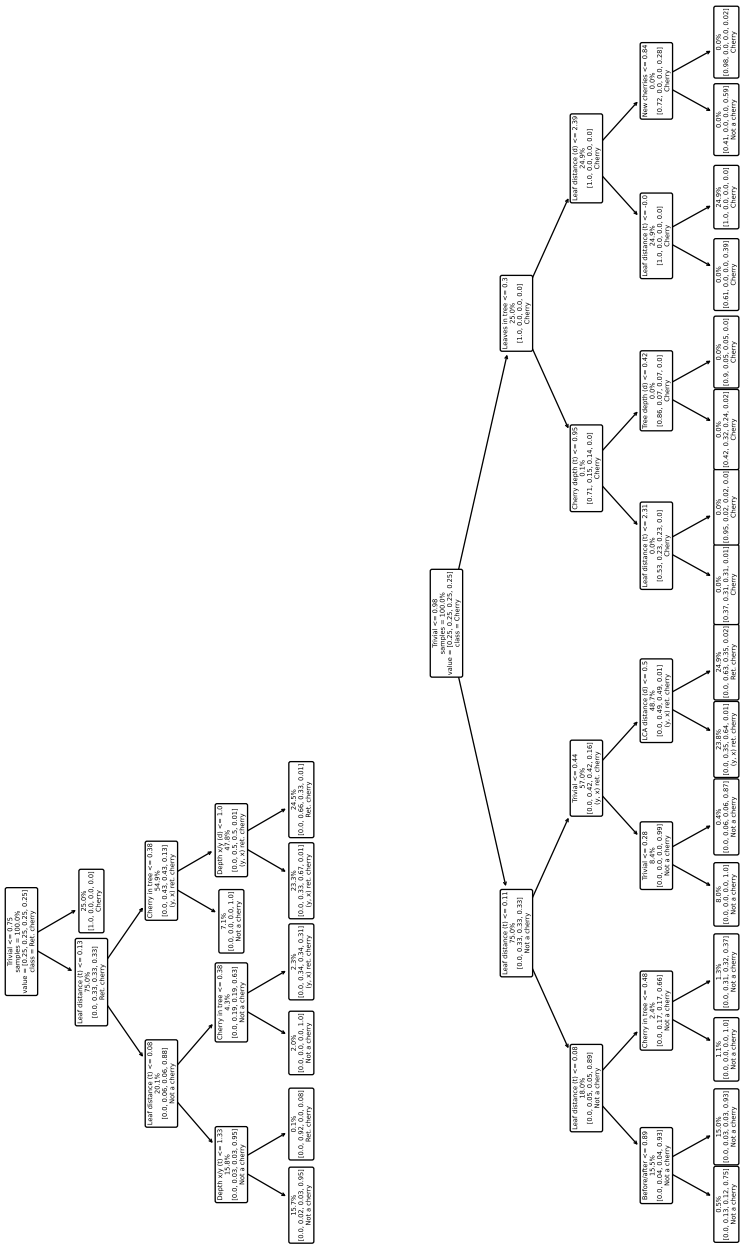
Table 6.3: Trained random forest models on different datasets for different combinations of max L (maximum number of leaves per network) and M (number of networks). Each row in the table represents one model. For each model, the testing accuracy is given under "Accuracy", and the total number of data points retrieved from all M networks is given under "Num. data". Each dataset is split for training and testing (90% – 10%). The training duration for the random forest is given in column "Training" and the time needed to generate the training data is given in column "Data gen.", in hours per core (we used 16 cores in total).

(a) Normal

max L	M	Accuracy	Num. data	Training (min)	Data gen. (hour/core)
20	5	1.0	840	00:00	00:00:12
	10	0.994	1,804	00:00	00:00:22
	100	0.998	17,388	00:03	00:04:19
	500	0.994	73,168	00:16	00:15:18
	1000	0.993	151,308	00:42	00:29:49
50	5	0.994	3,580	00:00	00:01:21
	10	0.997	7,860	00:01	00:02:22
	100	0.996	53,988	00:11	00:18:07
	500	0.997	268,552	01:04	01:31:18
	1000	0.998	535,624	04:01	02:56:21
100	5	1.0	4,944	00:00	00:01:13
	10	0.999	12,444	00:01	00:04:05
	100	0.999	128,824	00:25	00:41:54
	500	0.999	676,768	04:21	04:15:49
	1000	0.999	1,362,220	12:10	08:08:58

(b) LGT

max L	M	Accuracy	Num. data	Training (min)	Data gen. (hour/core)
20	5	0.974	768	00:01	00:00:19
	10	0.994	1,548	00:02	00:00:41
	100	0.976	12,244	00:09	00:04:20
	500	0.975	58,900	00:24	00:19:13
	1000	0.975	118,104	00:27	00:35:38
50	5	0.997	2,952	00:01	00:00:43
	10	0.995	3,796	00:03	00:01:01
	100	0.995	44,116	00:23	00:14:01
	500	0.994	219,472	01:39	01:06:45
	1000	0.994	421,204	02:45	02:10:45
100	5	0.996	5,080	00:06	00:01:23
	10	0.996	7,540	00:05	00:01:58
	100	0.998	114,900	00:31	00:34:25
	500	0.998	605,652	04:44	02:54:15
	1000	0.998	1,175,628	10:23	05:31:13



(a) Normal ML. Test accuracy = 0.815

(b) LGT ML. Test accuracy = 0.802

Figure 6.17: Classification tree with depth 4 of (a) the normal data set and (b) the LGT data set. For each node in the trees, except for the terminal ones, the first line is the feature condition. If this condition is met by a data point, it traverses to the left child node, otherwise to the right one. In the terminal nodes this line is omitted as there is no condition given. In each node, as also indicated with labels in the root node, the second line 'samples' is the proportional number of samples that follow the YES/NO conditions from the root to the parent of that node during the training process. The 'value' list gives the proportion of data points in each class, compared to the sample of that node. The last line indicates the most dominant class of that node. If a data point reaches a terminal node, the observation will be classified as the indicated class.

Table 6.4: Feature importances of random forest trained on the biggest dataset ($M = 1000$ and $\max L = 100$) based on normal (a) and LGT (b) network data. Higher importance indicates that a feature has more effect on the trained model. The values sum up to one. The descriptions of the features are given in Table 6.1.

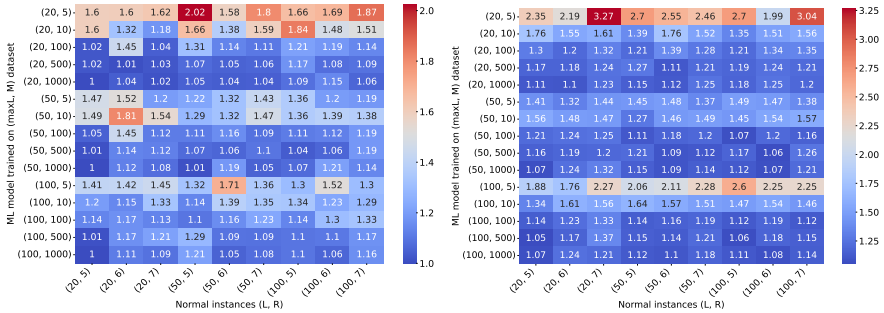
(a) Normal

Features		Importance
Leaf distance	(t)	0.190
Trivial		0.155
Cherry in tree		0.143
Leaf distance	(d)	0.122
LCA distance	(t)	0.068
Depth x/y	(t)	0.050
Cherry depth	(t)	0.047
Depth x/y	(d)	0.043
LCA distance	(d)	0.028
Leaf depth x	(t)	0.023
Leaf depth y	(t)	0.023
Cherry depth	(d)	0.020
Leaf depth x	(d)	0.020
Leaf depth y	(d)	0.020
Before/after		0.015
Tree depth	(d)	0.012
Tree depth	(t)	0.011
New cherries		0.006
Leaves in tree		0.004

(b) LGT

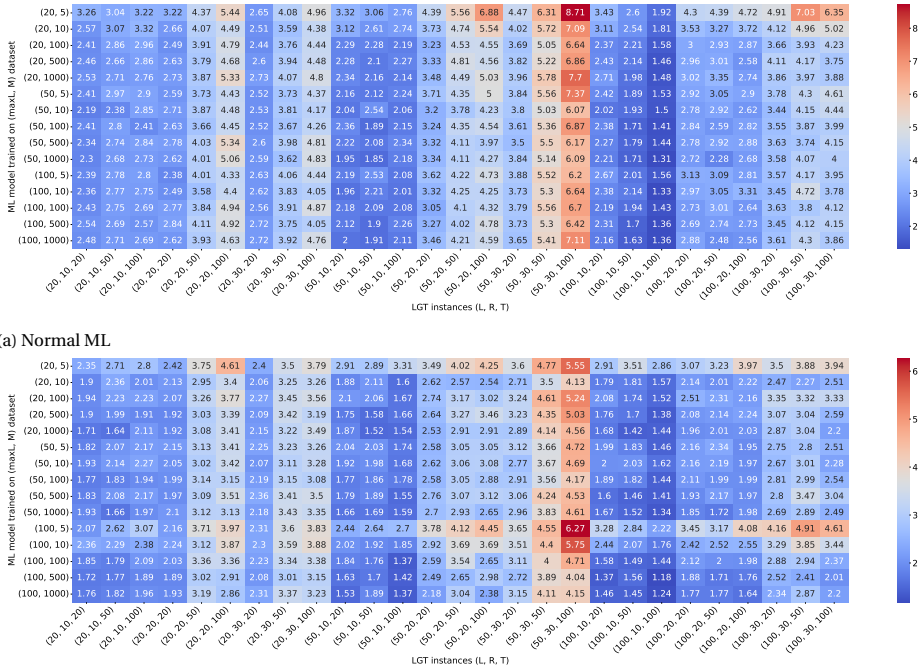
Features		Importance
Trivial		0.184
Leaf distance	(t)	0.162
Cherry in tree		0.146
Leaf distance	(d)	0.114
Depth x/y	(t)	0.058
LCA distance	(t)	0.056
Cherry depth	(t)	0.045
Depth x/y	(d)	0.038
LCA distance	(d)	0.032
Leaf depth y	(t)	0.024
Leaf depth x	(t)	0.023
Cherry depth	(d)	0.023
Leaf depth y	(d)	0.022
Leaf depth x	(d)	0.022
Before/after		0.016
Tree depth	(d)	0.013
Tree depth	(t)	0.011
New cherries		0.006
Leaves in tree		0.003

6.C. HEURISTIC PERFORMANCE OF ML MODELS



(a) Normal ML (b) LGT ML

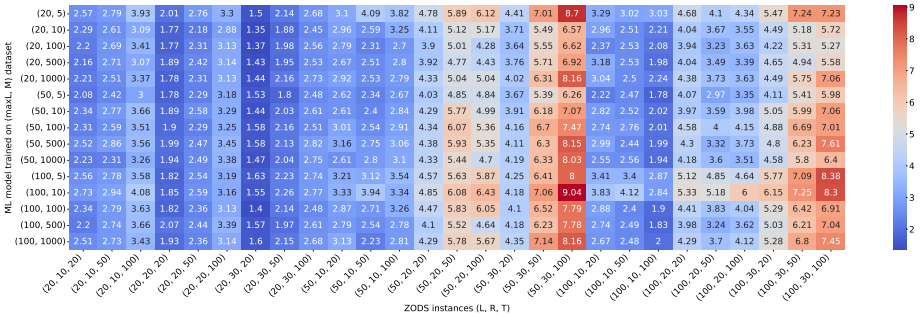
Figure 6.18: Results for ML on normal instances with the random forest model trained on each of the datasets given in Table 6.3, where **(a)** gives the results when the ML model is trained on normal data, and **(b)** gives the results when the model is trained on LGT data. For each training dataset, identified by the parameter pair $(\max L, M)$, the value shown in the heatmap is the average, within each instance group, of the reticulation number found by ML divided by the reference value. We used a group of 16 instances for each combination of parameters $L \in \{20, 50, 100\}$ and $R \in \{5, 6, 7\}$.



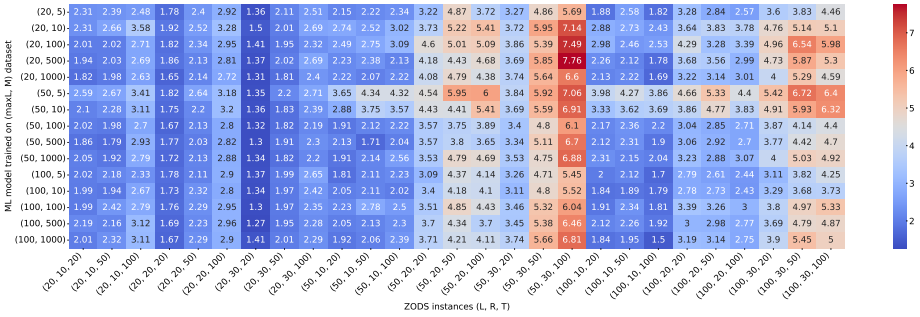
(a) Normal ML

(b) LGT ML

Figure 6.19: Results for ML on LGT instances for different training datasets, similar to Fig. 6.18, with $L \in \{20, 50, 100\}$, $R \in \{10, 20, 30\}$ and $|\mathcal{T}| \in \{20, 50, 100\}$.



(a) Normal ML



(b) LGT ML

Figure 6.20: Results for ML on ZODS instances for different training datasets, similar to Fig. 6.18, with $L \in \{20,50,100\}$, $R \in \{10,20,30\}$ and $|\mathcal{T}| \in \{20,50,100\}$.

7

CHERRY-PICKING HEURISTIC FOR NON-BINARY TREES

The Hybridization problem asks to combine a set of conflicting phylogenetic trees into a single phylogenetic network with the smallest possible number of reticulation nodes. This problem is computationally hard and previous solutions are restricted to small and/or severely restricted data sets, for example, a set of binary trees with the same taxon set or only two non-binary trees with non-equal taxon sets.

Proposed data-driven algorithm: *Building on our previous work on binary trees presented in Chapter 6, we present FHyNCH, the first algorithmic framework to heuristically solve the Hybridization problem for large sets of non-binary (or multifurcating) trees whose sets of taxa may differ. Our heuristics combine the cherry-picking technique, recently proposed to solve the same problem for binary trees, with two carefully designed machine-learning models. To better deal with the possible non-binarity of trees, we introduce another machine-learning model that drastically reduces the solution space in each iteration.*

Data generation scheme: *The data generation scheme is very similar to the one proposed in Chapter 6, with the necessary alterations to also handle multifurcating tree sets with mostly overlapping, but not necessarily equal, leaf sets.*

Experimental results: *We demonstrate that our methods are practical and produce qualitatively good solutions through experiments on both synthetic and real data sets.*

This chapter is based on Bernardini et al. [26] published in **Molecular Phylogenetics and Evolution**. In collaboration with Giulia Bernardini, Leo van Iersel, and Leen Stougie. The code is available at `https://github.com/estherjulien/FHyNCH`.

7.1. INTRODUCTION

Until recently, the evolutionary history of a set of species was normally modeled as a rooted phylogenetic tree. However, the greater availability of molecular data is encouraging a paradigm shift to multilocus approaches for phylogenetic inference, which often leads to discovering relationships among the species that deviate from the simple model of a tree [13, 104, 155]. Indeed, the phylogenetic trees inferred from different loci of the genomes often have conflicting branching patterns, due to evolutionary events like recombination, hybrid speciation, introgression or lateral gene transfer [41, 133, 143, 144]. In the presence of such events, evolution is more accurately represented by a rooted phylogenetic network, which extends the tree model and allows representing multi-parental inheritance of genetic material as *reticulation* nodes [132, 144].

A crucial problem is then to infer a single phylogenetic network from a set of conflicting trees built from different loci of the genomes in a data set. A commonly used criterion to estimate such a network, which is reasonable when discordance between trees is believed to be caused by multi-parent inheritance, is *parsimony* [104]: the goal is then to construct a network that simultaneously explains all ancestral relationships encoded by the trees with the fewest number of reticulation nodes. This problem is known in the literature by the name of HYBRIDIZATION and has been extensively studied.

HYBRIDIZATION has been shown to be NP-hard even for two binary input trees [39]. Most of the solutions proposed in the literature are limited to inputs consisting of only two binary trees with identical leaf sets. A few methods exist that waive some of these assumptions: some admit inputs consisting of several binary trees with identical [201] or largely overlapping [25, 27] leaf sets; others are able to process a pair of multifurcating (i.e., nonbinary) trees with overlapping, but not identical, leaf sets [102] or several multifurcating trees with identical leaf sets [149, 220].

However, to the best of our knowledge, there currently exist no solutions to HYBRIDIZATION for several multifurcating trees with different leaf sets, although realistic phylogenetic trees in biological studies are usually multifurcating and hardly contain exactly the same taxa. This work aims to fill this gap: we propose FHyNCH[1] (Finding Hybridization Networks via Cherry-picking Heuristics), a heuristic framework to find feasible (and qualitatively good) solutions to HYBRIDIZATION for a large number of multifurcating phylogenetic trees with overlapping, but not identical, leaf sets. Our methods combine the technique of *cherry picking*, first introduced in Linz and Semple [134], with machine learning to guide the search in the solution space.

The high-level scheme and the theoretical foundations of the methods we propose are the same as in Bernardini et al. [27]: however, the approach of Bernardini et al. [27] is restricted to binary trees, while most practical data sets consist of multifurcating trees. A straightforward adaptation to multifurcating trees would lead to a time-consuming algorithm that would be impractical for large instances (see Section 7.2.3). In contrast with previous methods, our new heuristics employ two machine-learning classifiers that are used sequentially at every iteration. The main novelty resides in the design and use of the first classifier, whose crucial role is to reduce the solution space at every iteration. Furthermore, making the new machine-learned heuristics applicable to multifurcating

[1] Pronounced as 'finch'. Finches are birds that love cherries and are notoriously known for picking cherries from trees in orchards.

trees with missing leaves required new, nontrivial techniques to generate training data: see Section 7.2.3.

Two things are important to notice at this point. First, since networks are not uniquely determined by the trees they contain [161], there may exist a large number of different good solutions, and our algorithm does not attempt to enumerate them all: in fact, how to summarize all equally good networks is still an open practical problem [99, 105]. In particular, no method to solve HYBRIDIZATION (whose goal is to minimize the number of reticulations only) can guarantee to reconstruct a specific network: all networks that display the input trees with the minimum possible number of reticulation nodes are optimal solutions. The network outputted by any algorithm that solves HYBRIDIZATION, including ours, should thus be interpreted as a possible (parsimonious) evolutionary history that is consistent with all the input trees.

Second, our heuristics output networks from the broad *orchard* class, which contains all and only the networks that can be obtained from a tree by adding horizontal arcs [200]. Such horizontal arcs can model lateral gene transfer (LGT) events, but also many networks with reticulation nodes modeling (for example) hybridization events are in the class of orchard networks. On the other hand, our methods are not suitable to be applied in the presence of incomplete lineage sorting.

The rest of the chapter is organized as follows. In Section 7.1.1 we discuss related work; in Section 7.2.1 we introduce notation and basic notions; in Section 7.2.2 we summarize the cherry-picking framework for HYBRIDIZATION, which lies at the heart of our solutions; in Section 7.2.3 we describe FHyNCH-MultiML, our main algorithmic scheme based on machine learning; in Section 7.3 we present our experimental results; finally, in Section 7.4 we give conclusions and future directions.

7.1.1. RELATED WORK

Several methods have been proposed in the literature to solve HYBRIDIZATION for two binary trees with equal leaf sets, both exactly [4, 38] and heuristically [162, 163]. The first practical methods to solve HYBRIDIZATION to optimality for more than two binary trees with equal leaf sets were PIRN$_C$ [220] and Hybroscale [3], which were able to process a small number of input trees (up to 5) that could be combined into a network with a relatively small number of reticulations. More recently, heuristic methods have been proposed to process larger sets of binary trees with identical taxa [149, 231].

The introduction of the so-called cherry-picking sequences [101, 134] was a game changer in the area: this theoretical framework allowed the design of the first methods capable of processing instances of up to 100 binary trees with identical leaf sets to optimality [40, 201], albeit with restrictions on the class of the output network and its number of reticulations.

To the best of our knowledge, only two methods have been proposed to solve HYBRIDIZATION for multifurcating trees, both limited to inputs consisting of only two trees: a simple FPT algorithm for trees with identical leaf set [168] and the Autumn algorithm, which allows differences between the leaf sets [102].

The potential of machine learning in phylogenetic studies has not been extensively explored yet. A few methods have been proposed for phylogenetic tree inference [1, 9, 10, 125, 185, 234], testing evolutionary hypotheses [126], and distance imputation [36];

finally, in previous work by the authors of this work, machine-learning techniques have been combined with cherry picking to solve HYBRIDIZATION for multiple binary trees with largely overlapping leaf sets [25, 27] or see Chapter 6.

7.2. METHODS

7.2.1. DEFINITIONS AND NOTATION

A *rooted phylogenetic network* N on a set of taxa X is a rooted directed acyclic graph such that the nodes other than the root are either (i) *tree nodes*, with in-degree 1 and out-degree greater than 1, or (ii) *reticulations*, with in-degree greater than 1 and out-degree 1, or (iii) *leaves*, with in-degree 1 and out-degree 0. The leaves of N are bi-univocally labeled by X, and we identify the leaves with their labels. The edges of N may be assigned a nonnegative *branch length*. We denote by $[1,n]$ the set of integers $\{1,2,...,n\}$. Throughout this chapter, we will often drop the terms "rooted" and "phylogenetic", as all the networks we consider are rooted phylogenetic networks.

We denote the *reticulation number* of a network N by $r(N)$, which can be obtained using the following formula: $r(N) = \sum_{v \in V} \max\left(0, d^-(v) - 1\right)$, where V is the set of nodes of N and $d^-(v)$ is the in-degree of a node v. A network T with $r(T) = 0$ is a *phylogenetic tree*.

We denote by $\mathcal{N}$ a set of networks and by $\mathcal{T}$ a set of trees. An ordered pair of leaves (x,y), $x \neq y$, is a *cherry* in a network if x and y share the same parent. Note that cherries (x,y) and (y,x) correspond to the same nodes and edges of the tree; the reason why they are considered two distinct cherries is motivated by the definition of the cherry-picking operation given below. An ordered pair (x,y) is a *reticulated cherry* if the parent of x, denoted by $p(x)$, is a reticulation, and the parent of y is a tree node that is one of the parents of $p(x)$ (see Figure 7.1b). Note that, in contrast with cherries, if (x,y) is a reticulated cherry then (y,x) is not, because the reticulation is constrained to be the parent of the first element of the pair. A pair of leaves is *reducible* if it is either a cherry or a reticulated cherry. Note that trees may have cherries but no reticulated cherries.

Suppressing a node v with a single parent $p(v)$ and a single child $c(v)$ is defined as replacing the arcs $(p(v),v)$ and $(v,c(v))$ by a single arc $(p(v),c(v))$ and deleting v. If the network has branch lengths, the length of the new edge is $\ell(p(v),c(v)) = \ell(p(v),v) + \ell(v,c(v))$. *Reducing* (or *picking*) a cherry (x,y) in a network N (or in a tree) is the action of deleting x and suppressing any resulting indegree-1 outdegree-1 nodes. A reticulated cherry (x,y) is *reduced (picked)* by deleting the edge $(p(y),p(x))$ and suppressing any indegree-1 outdegree-1 nodes. See Figure 7.1. Reducing a non-reducible pair does not affect N. In all cases, the resulting network is denoted by $N_{(x,y)}$: we say that (x,y) affects N if (x,y) is reducible in N, i.e., $N \neq N_{(x,y)}$.

Any sequence $S = (x_1,y_1),\ldots,(x_n,y_n)$ of ordered leaf pairs, with $x_i \neq y_i$ for all i, is a *partial cherry-picking sequence*; S is a *cherry-picking sequence* (CPS) if in addition, for each $i < n$, $y_i \in \{x_{i+1},\ldots,x_n,y_n\}$. Given a network N and a (partial) CPS S, we denote by N_S the network obtained by reducing in N each element of S, in order. We let $S \circ (x,y)$ denote the sequence obtained by appending pair (x,y) at the end of S. We say that a CPS S *fully reduces* a network N if N_S is just a root with a single leaf; S is of minimum length for N if all pairs of S affect the network.

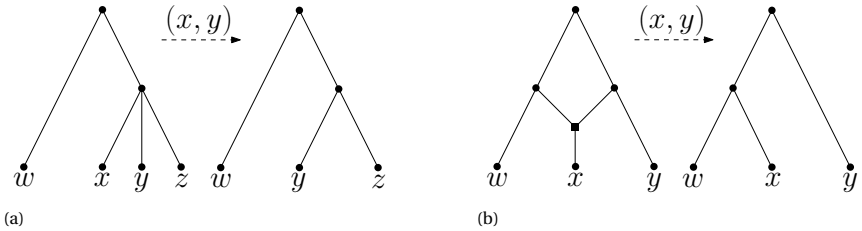


Figure 7.1: The leaf pair (x, y) is picked in two different networks. In **(a)** (x, y) is a cherry, and in **(b)** (x, y) is a reticulated cherry, as well as (x, w). Note that in **(b)** the parent of x and the parent of y are suppressed after picking (x, y).

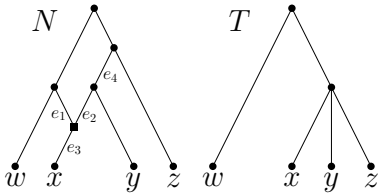


Figure 7.2: Example of a multifurcating tree T that is displayed in the binary network N via the following operations: edge e_1 is deleted, then the parents of x and w are suppressed, and finally edge e_4 is contracted.

N is an *orchard network* if there exists a CPS that fully reduces it. If a CPS fully reduces all networks in a set $\mathcal{N}$, we say that it *fully reduces* $\mathcal{N}$. In this chapter, we will consider CPSs which fully reduce a set of trees $\mathcal{T}$; $|\mathcal{T}|$ denotes the number of trees in $\mathcal{T}$.

7

THE HYBRIDIZATION PROBLEM

The main problem considered in this chapter is the following: given a set of phylogenetic trees on overlapping (but not necessarily equal) sets of taxa, infer a single network with the fewest number of reticulations that summarizes all the input trees. Definition 7.1 formalizes the concept of *summarizing* a set of trees: we seek a network where each of the input trees is displayed (see Figure 7.2 for an example).

Definition 7.1. *Let N be a network on a set of taxa X and let T be a tree on a set of taxa $X' \subseteq X$. Then, T is* displayed *in N if T can be obtained from N by applying a sequence of the following operations in any order:*

(a) Contract an edge (u, v) to a single node w: all parents of u and v except u become parents of w and all children of u and v except v become children of w.

(b) Delete an edge: if the head of the edge is a leaf, delete the leaf node as well.

(c) Suppress a node with in- and out-degree 1.

We now formally define the key problem of this work, called HYBRIDIZATION [16].

Input: A set $\mathcal{T} = \{T_1, T_2, \ldots, T_t\}$ of phylogenetic trees on sets of taxa X_1, $X_2, \ldots, X_t$, respectively.

Output: A binary phylogenetic network N on the set of taxa $X = \bigcup_{i=1}^{t} X_i$ which displays all the trees in $\mathcal{T}$ with the smallest possible number of reticulations.

Note that the input trees are not required to be binary nor to have identical leaf sets: a tree $T_i \in \mathcal{T}$ on a set of taxa X_i is said to have *missing leaves* if $X_i \subsetneq X$. The input trees may or may not have branch lengths. Branch lengths do not play any role in HYBRIDIZATION, as the requirements to be satisfied by a solution only affect its topological structure (and for this reason, output networks do not have branch lengths); however when branch lengths are part of the input, our methods use them as features to train and guide the decisions of the underlying machine-learning model.

7.2.2. SOLVING HYBRIDIZATION VIA CHERRY PICKING

Our methods fall in the Cherry-Picking Heuristic (CPH) framework, first introduced in Bernardini et al. [27] to find feasible solutions to HYBRIDIZATION for *binary* input trees. In this section, we recall the main characteristics of the CPH framework; we refer the reader to Bernardini et al. [27, Section 3] or Section 6.3 of this thesis for a complete discussion.

The CPH framework relies on the following results given in Janssen and Murakami [109]: (i) if a minimum-length CPS that fully reduces a binary orchard network N on a set of taxa X also fully reduces a tree T (or another network, not necessarily binary) on a set of taxa $X' \subseteq X$, then T is displayed in N; and (ii), any CPS S can be processed in reverse order to reconstruct a unique binary orchard network for which S is a minimum-length CPS.

The main idea underlying CPH is thus to construct a CPS that fully reduces the input set of trees $\mathcal{T}$ and then to process this sequence in reverse order to obtain a network N which is guaranteed to be a feasible solution to HYBRIDIZATION by means of result (i). Any algorithm in the CPH framework constructs a CPS S in an incremental way (starting from an empty sequence) by repeating the following steps until all the input trees are fully reduced:

1. Choose a pair of leaves (x, y) that is reducible in at least one tree (i.e., a cherry of the tree set).
2. Reduce (x, y) in all trees.
3. Append (x, y) to S.

Once the input trees are fully reduced, the obtained sequence S is processed in reverse order to construct the output network N (after a last technical step to make sure S is a CPS and not just a partial sequence, see Bernardini et al. [27, Section 3.1] or Algorithm 6.2 of this thesis for details) using the dedicated method from Janssen and Murakami [109].

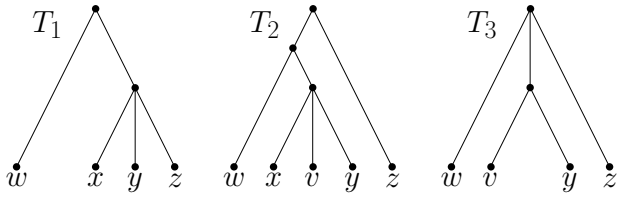


Figure 7.3: Example of a tree set with a *trivial cherry* (x, y): in trees T_1 and T_2, x and y form a cherry, and x is not in T_3. In contrast, (x, z) is not a trivial cherry: it is a cherry in T_1, but both x and z are in T_2 without forming a cherry.

Since the latter method outputs binary networks, so do all algorithms in the CPH framework. Note that this is not a significant restriction because whenever there exists a multifurcating network displaying $\mathscr{T}$, there also exists a binary network displaying $\mathscr{T}$ with the same reticulation number. The following lemma links the number of reticulations of N with the length of the CPS it is reconstructed from.

Lemma 7.1 (van Iersel et al. [202])**.** *Let S be a CPS on a set of taxa X. The number of reticulations of the network N reconstructed from S is $r(N) = |S| - |X| + 1$.*

The formula of Lemma 7.1 implies that the shorter the cherry-picking sequence constructed by the algorithm, the fewer the reticulations of the output network. The algorithms in the CPH framework differ from one to another for the criterion with which a reducible pair is chosen at each iteration: the goal of this study is to find a criterion that produces as short as possible sequences for input multifurcating trees with missing leaves.

Before discussing our new methods, we recall a simple, but rather effective algorithm in the CPH framework that can be easily modified to be applied to multifurcating input trees with missing leaves. In the rest of this chapter, we will call FHyNCH-TrivialRand the adaptation of this strategy to multifurcating trees with missing leaves. We need the following definition (see Figure 7.3 for an example).

Definition 7.2. *An ordered leaf pair (x, y) is a* trivial cherry *(or trivial pair) of $\mathscr{T}$ if it is reducible in all $T \in \mathscr{T}$ that contain both x and y, and there is at least one tree in which it is reducible.*

It has been empirically shown in the previous chapter that picking trivial cherries (when they exist) produces good results in terms of the number of reticulations of the output network. The criterion used by FHyNCH-TrivialRand to pick a pair at each iteration is thus to choose a trivial cherry if there is any; and to choose a pair uniformly at random among the cherries of the current tree set if no trivial cherry exists. This randomized algorithm is so simple and fast that several runs on the same input can be computed in a reasonable time so as to select the best output as a final result: in our experiments, we will compare our new methods against this strategy.

7.2.3. A MACHINE-LEARNED ALGORITHM FOR HYBRIDIZATION

A machine-learning model in the CPH framework for solving HYBRIDIZATION on binary input trees was first proposed in Bernardini et al. [27]: although in theory this method

is applicable in the presence of missing leaves (i.e., to input trees with different sets of taxa), the authors experimentally showed that the quality of the results rapidly degrades for increasing percentage of missing leaves. In principle, the machine-learning model of CPH could be straightforwardly adapted to work on multifurcating trees; however, its time complexity would get much worse, resulting in a slow algorithm that does not handle well differences among the sets of taxa.

In this section, we propose a new, different machine-learning model specifically designed for multifurcating input trees with missing leaves.

THEORETICAL BACKGROUND

The foundations on which our new methods rely are the same as for the model on binary trees. We report here a high-level description of this background and refer the reader to Section 6.3.3 for details.

The main idea is the following. Let $\mathsf{OPT}(\mathcal{T})$ denote the set of networks that display the input trees $\mathcal{T}$ with the minimum possible reticulation number (note that, in general, $\mathsf{OPT}(\mathcal{T})$ contains more than one network [161]). Ideally, we aim at finding a CPS fully reducing $\mathcal{T}$ that is also a minimum-length CPS that fully reduces some network of $\mathsf{OPT}(\mathcal{T})$. This is because any method in the CPH framework outputs a network for which the produced CPS is a minimum-length sequence. Our goal is to design a machine-learned oracle to predict, at each iteration of the method, which pairs of $\mathcal{T}$ are reducible in some optimal network. Using this prediction, at every iteration the algorithm chooses a pair that most probably leads to an optimal solution.

MACHINE-LEARNING MODELS

To predict whether a given cherry of the tree set is a reducible pair in some optimal network N for $\mathcal{T}$, we train two random-forest classifiers: one using features that carry information on the leaves of the trees, another using features about their cherries. The main novelty of this approach compared to those proposed in the previous chapter is in the design and use of the first classifier, whose crucial role is to reduce the solution space at every iteration: without its introduction, the method would be infeasible for non-binary input data sets of practical size. This is because it may require computing features for a quadratic number of cherries at every iteration, in contrast with the binary case, in which the number of cherries is always linear in the number of taxa. The accuracy of the simple random forest models for our problem was so good that we did not find any advantage in applying deep learning instead.

In the first classifier, a data point is a pair $(\mathbf{F}_1, c_1)$, where $\mathbf{F}_1$ is an array containing 8 features, listed in Table 7.1, of a leaf x, and c_1 is a binary label modeling whether or not x belongs to a reducible pair (either a cherry or a reticulated cherry) of the unknown target optimal network N. The second classifier is similar to the one proposed in Bernardini et al. [27]: here, a data point is a pair $(\mathbf{F}_2, c_2)$, where $\mathbf{F}_2$ is an array containing 21 features, listed in Table 7.2, of a cherry (y, z), and c_2 is a binary label modeling whether or not (y, z) is a reducible pair of N. The two classifiers receive in input the arrays of features, learn the association between $\mathbf{F}_1$ and c_1 and between $\mathbf{F}_2$ and c_2, respectively, and output a label for each data point together with a confidence score modeling the probability that the predicted label is correct.

The general scheme of our strategy is as follows. First, the algorithm computes the array $\mathbf{F}_1$ of features for each $x \in X$, thus creating a data point for the first classifier for each leaf of the initial tree set, and initializes an empty cherry-picking sequence S. It then repeats the following steps until all the trees are fully reduced.

1. Select a subset C of k leaves from the current tree set, based on the predictions of the first classifier.
2. Compute $\mathbf{F}_2$ for each cherry in the current tree set *that contains one leaf from* C, thus creating data points for the second classifier only for this subset of cherries.
3. Choose a cherry (x, y) based on the predictions of the second classifier, append it to S, and reduce (x, y) in all trees.
4. Update $\mathbf{F}_1$ for all the data points for the first classifier.

We name this algorithmic scheme FHyNCH-MultiML. The constant $k \geq 1$ determining the size of the subset of leaves selected in Step 1 at each iteration is a parameter of the algorithm: in Section 7.3.1 we report experiments about the impact of k on the running time and the quality of the results of our algorithm. The simplest way to implement Step 1 is to select the k leaves that are predicted by the first classifier to be part of a reducible pair of an optimal network with the highest probability; other possible strategies are supported by our method, e.g., to fix a threshold $\lambda \in (0,1)$ and to select the k leaves uniformly at random among the ones whose probability to be part of a reducible pair of an optimal network is at least λ. A similar argument can be made for the choice of the cherry in Step 3. The number of data points for the first classifier is always bounded by $|X|$, the number of taxa. The array $\mathbf{F}_1$ of features is efficiently updated in Step 4 at each iteration for each data point, that is, for each leaf of the current tree set. In contrast, arrays $\mathbf{F}_2$ are computed from scratch in Step 2 at every iteration because the subset of cherries for which a data point is created changes across different iterations (it depends on the leaves chosen at Step 1).

The main role of the first classifier is in fact to reduce the number of cherries for which $\mathbf{F}_2$ must be computed: this is needed because the total number of cherries in the tree set could be superlinear (up to quadratic in the number $|X|$ of taxa), which could make it impractical to compute $\mathbf{F}_2$ for every cherry at every iteration. Using the first classifier beforehand guarantees that $\mathbf{F}_2$ must be computed only for a linear number $\mathcal{O}(k|X|)$ of cherries at each iteration, resulting in a much faster and more practical algorithm.

Features 5-6 for the first classifier and 6-13 for the second classifier can be computed for both branch lengths and unweighted branches. We refer to these two options as *branch distance* and *topological distance*, respectively. The *branch depth* (resp. *topological* depth) of a node u in a tree T is the total branch length (resp. the total number of edges) on the path from the root to u; the *leaf-depth* of T is the maximum depth of any leaf of T; the depth of a cherry (x, y) is the depth of the common parent of x and y; and the *cherry-depth* of T is the maximum depth of any cherry of T. The leaf distance between x and y is the total length of the path from the parent of x to the lowest common ancestor of x and y, denoted by $\mathrm{LCA}(x, y)$, plus the total length of the path from the parent of y to $\mathrm{LCA}(x, y)$. In particular, the leaf distance between the leaves of a cherry

Table 7.1: Features of a leaf x for the first classifier.

Num.	Feature name	Description
1	Leaf pickable	Ratio of trees in which x is part of a cherry
2	Leaf in tree	Ratio of trees that contain leaf x
3	Siblings avg.	Avg over trees with x of ratios "num. of siblings of x/number of leaves in tree"
4	Siblings std.	Standard deviation of "num. of siblings of x/number of leaves in tree"
Features measured by distance (d) and topology (t)		
$5_{b,t}$	Leaf depth x avg.	Avg over the trees that contain x of ratios "depth of x/leaf-depth of the tree"
$6_{b,t}$	Leaf depth x std.	Standard deviation of "depth of x/leaf-depth of the tree"

Table 7.2: Features of a cherry (x, y) for the second classifier. These features are the same as those for the classifier used in Bernardini et al. [27]; however, the latter classified cherries into four classes instead of only two.

Num	Feature name	Description
1	Cherry in tree	Ratio of trees that contain cherry (x, y)
2	New cherries	Number of new cherries of $\mathcal{T}$ after picking cherry (x, y)
3	Before/after	Ratio of the number of cherries of $\mathcal{T}$ before/after picking cherry (x, y)
4	Trivial	Ratio of trees with both leaves x and y that contain cherry (x, y)
5	Leaves in tree	Ratio of trees that contain both leaves x and y
Features measured by distance (d) and topology (t)		
$6_{b,t}$	Tree depth	Avg over trees with (x, y) of ratios "cherry-depth of the tree/max cherry-depth over all trees"
$7_{b,t}$	Cherry depth	Avg over trees with (x, y) of ratios "depth of (x, y) /cherry-depth of the tree"
$8_{b,t}$	Leaf distance	Avg over trees with x and y of ratios "x-y leaf distance/cherry-depth of the tree"
$9_{b,t}$	Leaf depth x	Avg over trees with x and y of ratios "depth of x/cherry-depth of the tree"
$10_{b,t}$	Leaf depth y	Avg over trees with x and y of ratios "depth of y/cherry-depth of the tree"
$11_{b,t}$	LCA distance	Avg over trees with x and y of ratios "x-LCA(x, y) distance/y-LCA(x, y) distance"
$12_{b,t}$	Depth x/y	Avg over trees with x and y of ratios "depth of x/depth of y"
$13_{b,t}$	LCA depth	Avg over trees with (x, y) of ratios "depth of LCA(x, y)/cherry-depth of the tree"

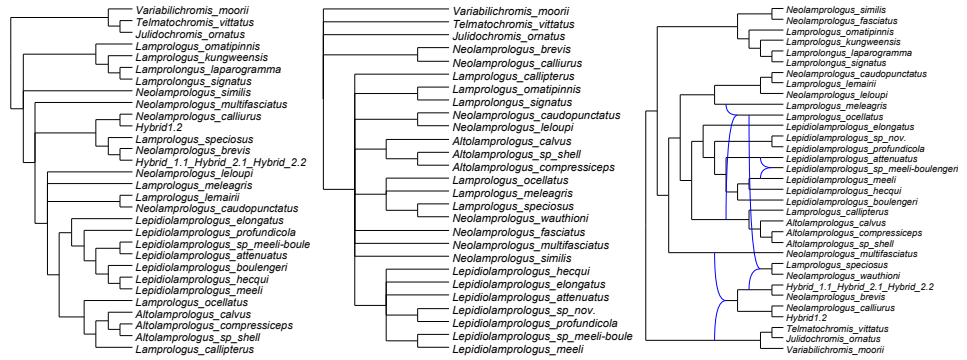
is zero as their LCA is their common parent. All of the above quantities can be defined both using branch distance and topological distance.

TREE EXPANSION

We now briefly describe a heuristic improvement to our methods, called *tree expansion*, that was already introduced for binary trees in Chapter 6, and can be applied as-is to multifurcating trees. Tree expansion is applied whenever a trivial cherry (x, y) is chosen to be reduced at some iteration. By Definition 7.2, each tree T in the current tree set belongs to one of the following classes with respect to the trivial cherry (x, y): (i), (x, y) is a cherry of T; (ii), neither x nor y are leaves of T; (iii), T has leaf y but not x; and (iv), T has leaf x but not y.

Without tree expansion, after reducing (x, y) in the tree set, leaf x is removed from the trees in class (i), but not from the trees in class (iv), thus it still may be present in the tree set. The goal of tree expansion is to make x disappear from the whole tree set, as this empirically reduces the length of the produced sequence and thus the number of reticulations in the output network. Tree expansion consists of the following operation:

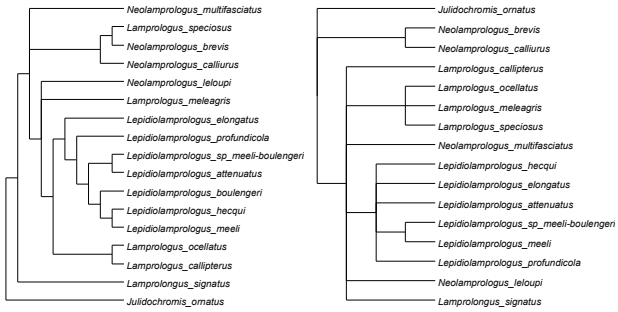
Rule 1 (Tree expansion)**.** *Before reducing a trivial cherry* (x, y) *in the tree set, add leaf* y *to form a cherry with* x *in all the trees in class* (iv).



(a) Mitochondrial tree (b) Nuclear tree (c) Output network

Figure 7.4: Mitochondrial (a) and nuclear (b) phylogenies for the Lamprologini tribe, preprocessed as in Huson and Linz [102]. The network outputted by FHyNCH-MultiML (c) has the optimal number of 4 reticulations.

After tree expansion, picking (x, y) will make x disappear from the set. Another way of viewing this operation is as a *relabeling* of x by y in all the trees in class (iv). It was proved in Bernardini et al. [27, Lemma 6] that this move does not affect the feasibility of the output: in other words, the network produced using tree expansion still displays the input set of trees. The same proof applies to the case of multifurcating trees.



(a) Mitochondrial tree (b) Nuclear tree

Figure 7.5: The trees of Figure 7.4 after reducing all their trivial cherries.

Example 7.1. *To illustrate the workings of FHyNCH-MultiML, we applied it to two phylogenetic trees for the Lamprologini tribe (one representing the mitochondrial phylogeny, the other the nuclear phylogeny), studied in Koblmüller et al. [120]. The same data were later used to test the Autumn algorithm [102]. In Huson and Linz [102], the phylogenetic trees were preprocessed to contract edges that had a bootstrap support of* 50% *or less. We applied FHyNCH-MultiML to these preprocessed trees, after deleting a few species that were misspelt in one of the two trees of Huson and Linz [102] and were mistakenly considered two different species in the two phylogenies*[2]*. Note that removing these species does not affect the number of reticulations needed because each of the variants was in only one*

[2] E.g., *Neolamprologus wauthioni* is mistakenly spelt as *Naolamprologus wauthioni* in the mitochondrial tree

of the trees in Huson and Linz [102]. The trees are multifurcating and have different taxa sets.

The input trees and the network outputted by FHyNCH-MultiML are shown in Figure 7.4. Notably, although FHyNCH-MultiML is a heuristic specifically designed for multiple trees, in this case, it returns a network with the same number of reticulations as in the output of the exact Autumn algorithm Huson and Linz [102], thus an optimal result. We also observe that the Autumn algorithm has several practical advantages: it returns multiple optimal networks and it returns nonbinary networks. In comparison, the networks produced by FHyNCH-MultiML could be more resolved than necessary to display the input trees.

Let us now have a closer look at the first iterations of FHyNCH-MultiML. The two input trees contain several trivial cherries: e.g., (Hybrid1.2,Neolamprologus_calliurus) is a cherry in the mitochondrial tree, and the label 'Hybrid1.2' does not appear in the nuclear tree, thus the cherry is trivial as per Definition 7.2; another trivial pair is (Telmatochromis_vittatus,Julidochromis_ornatus), which is a cherry both in the mitochondrial and in the nuclear tree; and many more (16 in total). The first iterations are devoted to picking all such trivial cherries, which are also cherries of the output network of Figure 7.4 (c). After picking all the initial trivial cherries, the two input trees were reduced to the two trees shown in Figure 7.5.

At this point, the first classifier computed the features of Table 7.1 for all the 15 leaves remained in the trees of Figure 7.5 and returned 'Lepidiolamprologus_sp_meeli-boulengeri' (abbreviated as 'Lep_sp_meeli-boule' in the rest of the example) as the top-scoring leaf, with a score of 0.98*. This leaf formed a cherry with 'Lepidiolamprologus_attenuatus' in the Mitochondrial tree (abbreviated as 'Lep_att') and with 'Lepidiolamprologus_meeli' ('Lep_meeli') in the nuclear tree. The second classifier thus computed the features of Table 7.2 for the four cherries (Lep_sp_meeli-boule,Lep_att), (Lep_att,Lep_sp_meeli-boule), (Lep_sp_meeli-boule,Lep_meeli), (Lep_meeli,Lep_sp_meeli-boule), and returned (Lep_sp_meeli-boule,Lep_att) as the top-scoring. This cherry was thus picked from the mitochondrial tree. After this iteration, the cherry (Lep_sp_meeli-boule,Lep_meeli) became trivial (as 'Lep_sp_meeli-boule' was no longer present in the mitochondrial tree) and was thus picked from the nuclear tree. In the end, FHyNCH-MultiML produced a cherry-picking sequence of length* 36*; since the total number of taxa labeling the input trees was* 33*, the output reticulation number was* 4.

7

OBTAINING TRAINING DATA

Generating data to train our classifiers is nontrivial because of the lack, in general, of ground truth: no existing algorithm is able to find an optimal solution – let alone all optimal solutions – for sufficiently large instances. We thus rely on the following procedure. We first generate a binary network N on a set of taxa X using the LGT (lateral gene transfer) network generator of Pons et al. [169] and extract the set $\tilde{\mathcal{T}}$ of all trees that are displayed in N and have the whole X as leaf set. We then contract and delete some edges (see Definition 7.1) from each of these trees using the following criteria. We set up

used in Huson and Linz [102]; the two names (correct and misspelt) labeled two different leaves of the output networks. Similar typos occurred for another two species.

two thresholds $\mathsf{MI}, \mathsf{Me} \in (0,1)$; for each tree $T \in \tilde{\mathcal{T}}$, we choose a value $p_l^T \in (0, \mathsf{MI})$ and a value $p_e^T \in (0, \mathsf{Me})$ uniformly at random, contract each edge of T with probability p_e^T and delete each leaf (by deleting the edge that connects it to its parent) with probability p_l^T. The thresholds MI and Me thus model the maximum probability with which a leaf is deleted and an edge is contracted, respectively, in any of the trees of $\tilde{\mathcal{T}}$; and we apply these operations with a different probability for each tree. The resulting tree set $\mathcal{T}$ consists of multifurcating trees (as a result of edge contractions) with missing leaves (as a result of leaf deletions).

Once we have generated the set $\mathcal{T}$, we create a data point for the first classifier for each leaf of $\mathcal{T}$, labeling it according to whether it is in a reducible pair of N or not; and similarly, we create a data point for the second classifier for each cherry of $\mathcal{T}$, labeling it according to whether it is reducible in N or not. We then iteratively choose a reducible pair from N, reduce it both in N and in $\mathcal{T}$, and update the data points and labels of each classifier.

As noted in Section 6.4.2, it could be the case that at some point, $\mathcal{T}_S$ is no longer displayed by N_S for a partial sequence S. We terminate when N is fully reduced or when the reducible pairs of N_S are not present in $\mathcal{T}_S$, and store all the obtained data. We remark that N is not necessarily an optimal network for the generated trees [161]. However, its number $r(N)$ of reticulations provides an upper-bound estimate of the number of reticulations of an unknown optimal network, and in Section 7.3.1 we will use $r(N)$ as a reference value to evaluate the quality of our results for the synthetically generated data sets.

7.3. RESULTS

The code of all our heuristics and for generating data is written in Python and is available at `https://github.com/estherjulien/FHyNCH`. All experiments ran on a computing cluster with an AMD Genoa 9654 CPU, of which 16 cores were used. We conducted experiments on both synthetic and real data. `scikit-learn` [166] with default settings was used for the random forest model.

7.3.1. SYNTHETIC DATA

Similar to the training data, we generated each of the synthetic datasets by first growing a binary network N on a set of taxa X using the LGT network generator of Pons et al. [169], extracting some of the trees that are displayed in N and have the whole X as leaf set, and finally deleting some leaves and contracting some edges from each of the extracted trees as described in Section 7.2.3. We generated several instances for different combinations of the following parameters: the number $R \in \{10, 20, 30\}$ of reticulations of the generating network N; the number $L \in \{20, 50, 100\}$ of leaves of the generating network N (i.e., the size of the set of taxa X); the number $|\mathcal{T}| \in \{20, 50, 100\}$ of trees extracted from N; and the thresholds $\mathsf{MI}, \mathsf{Me} \in \{0, 0.1, 0.2\}$ for leaf and edge deletion, respectively (see Section 7.2.3). For each combination of the parameters R, L, MI and Me we generated 20 networks for each value of $|\mathcal{T}| \in \{20, 50, 100\}$ and as many instances for HYBRIDIZATION. The 60 instances generated for a specific combination of values for $L, R, \mathsf{MI}, \mathsf{Me}$ constitute an *instance group*.

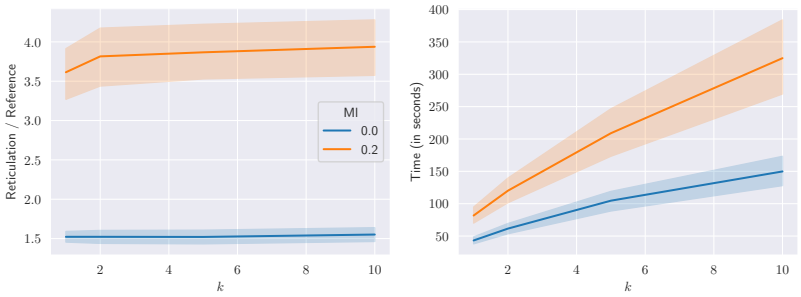


Figure 7.6: Results (left) and running time in seconds (right) for synthetic instances with $L = 100, R = 30, |\mathcal{T}| \in \{20, 50, 100\}$, Me = 0 and MI $\in \{0, 0.2\}$ for varying $k \in \{1, 2, 5, 10\}$ (k is the number of leaves chosen in Step 1 of FHyNCH-MultiML: see Section 7.2.3).

We run all our experiments setting the parameter $k = 1$, as the experiment summarized in Figure 7.6 indicates that larger values of k increase the running time of the algorithm without improving the quality of the results. Since no exact method can be applied to these instances, we compared FHyNCH-MultiML with FHyNCH-TrivialRand, a randomized heuristic proposed in Bernardini et al. [27] and here briefly summarized in Section 7.2.2 that can be straightforwardly modified to be applied to nonbinary trees with missing leaves. For each instance I, we ran FHyNCH-MultiML once, while FHyNCH-TrivialRand was run $\min\{x(I), 1000\}$ times, where $x(I)$ is the number of runs that can be executed in the same time as one run of FHyNCH-MultiML on the same instance; we then selected the best output over all such runs, and considered this value the result of FHyNCH-TrivialRand for instance I. To evaluate the quality of the methods, within each instance group we used the number R of reticulations of the generating networks as a reference value and divided the number of reticulations output by each method by this value. The results are summarized in Figure 7.7.

It is immediately apparent that the results of FHyNCH-TrivialRand rapidly degrade for increasing instance size and increasing percentages of missing leaves and multifurcating nodes in the input trees, while the performance of FHyNCH-MultiML is much more stable. Moreover, for the same number of leaves in the generating network N (parameter L) the results of both methods become worse for increasing number of reticulations of N (parameter R), the deterioration being much more marked for FHyNCH-TrivialRand than for FHyNCH-MultiML. With few exceptions (including, e.g., the instance group with $L = 20, R = 10$, MI $= 0$, Me $= 0.2$), the performance of FHyNCH-MultiML and FHyNCH-TrivialRand on the smaller instances ($L = 20$) do not seem to be significantly different, although both the median and the variance of the results of FHyNCH-MultiML are consistently smaller than those of FHyNCH-TrivialRand. In all the other instance groups, FHyNCH-MultiML substantially outperforms FHyNCH-TrivialRand, the difference being more pronounced in groups with higher percentages of missing leaves and contracted edges. Unlike what happens for FHyNCH-TrivialRand, the quality of the results of FHyNCH-MultiML is only marginally affected by increasing percentages of contracted edges. For example, the median for the results of FHyNCH-

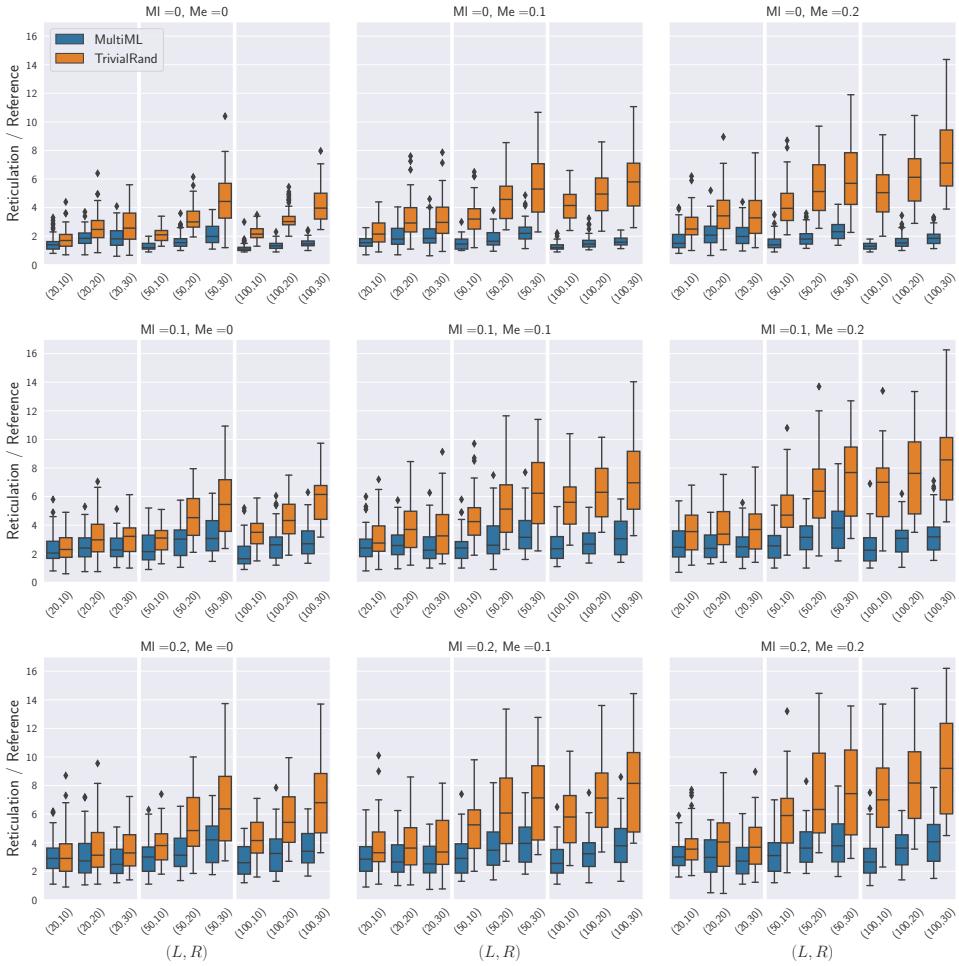


Figure 7.7: *Synthetic* instance results for different values of MI, Me, L, and R. The reference reticulation number value per instance is the network the trees were extracted from.

MultiML in the instance group with $L = 100, R = 30$ and no contracted edges nor missing leaves is 1.47, the results for 75% of the instances being within a factor 1.7 from the reference value; these values become 1.85 and 2.14, respectively, in the instance group with $L = 100, R = 30$, no missing leaves and Me = 0.2. Then, when increasing the percentage of missing leaves, the median of the results of FHyNCH-MultiML within the group with $L = 100, R = 30$, MI = Me = 0.2 increases to 4.07, with the results for 75% of the instances being within a factor of 5.28 from the reference.

In comparison, for the same instance groups the results of FHyNCH-TrivialRand are as follows: in the group with $L = 100, R = 30$, no missing leaves nor contracted edges, the median is 3.97, the results for 50% of the instances being within a factor in the range of 3.2 to 5.01 from the reference value; in the group with $L = 100, R = 30$, no missing leaves

and Me = 0.2, the median is 7.12, the results for 50% of the instances being within a factor in the range of 5.52 to 9.43 from the reference value; and finally, in the group with $L = 100, R = 30$ and Me = Ml = 0.2 the median is 9.2, the results for 50% of the instances being within a factor in the range of 6.92 to 12.35 from the reference value.

The poor performance of FHyNCH-TrivialRand on instances consisting of trees with many leaves, especially when they are nonbinary, is due to its randomized nature: the more leaves, the more cherries in the tree set, thus the smaller the probability of picking a good pair at every iteration. When the trees are nonbinary, the number of cherries increases even more, making the issue more serious; and the same holds for missing leaves - when a leaf is missing from a tree, this may originate a cherry that would not have been there in the complete tree. In contrast, the more leaves in the trees the more data are available for the machine-learned models to make good decisions.

In the next section, we show that this trend is conserved when the two methods are applied to real datasets: when the instances are small enough, FHyNCH-TrivialRand often outperforms FHyNCH-MultiML, while on larger instances the results of FHyNCH-MultiML are significantly better.

7.3.2. REAL DATA

Evaluating the performance of FHyNCH-MultiML on real data is a nontrivial task because, since no exact method exists for solving HYBRIDIZATION for more than two multifurcating trees with missing leaves, we do not have a baseline to compare against. We thus adopted two different strategies depending on the instance size. For small enough instances, consisting of up to 6 trees, we apply a procedure - described in the paragraph devoted to small instances - to make the trees binary and with equal leaf sets, to be able to apply the exact TreeChild method from van Iersel et al. [201] and use its result as a reference value. Although this is not necessarily the true optimum, both because by making the trees binary and adding missing leaves we could introduce spurious constraints that might originate unnecessary reticulations and because TreeChild is exact only for a special class of networks, this value is expected to be reasonably close to the real optimum. Larger instances cannot be processed by TreeChild nor other exact methods, thus we simply compared the performance of FHyNCH-MultiML and FHyNCH-TrivialRand and reported the *relative error* of one compared to the other, i.e., the difference between the two results divided by the best (thus the smallest) one. More details will be provided in the paragraph devoted to large instances.

Data sets. We extracted several instances of HYBRIDIZATION from the publicly available data set used in Beiko [20], consisting of phylogenetic trees for 159,905 distinct homologous gene sets from 1173 sequenced bacterial and archaeal genomes. The trees are multifurcating and have missing leaves. For different sizes of the tree set $|\mathscr{T}| \in \{2,4,6,10,20,30,40,50,60\}$, we extracted instances. For each instance, we also fixed an approximate number of leaves $L \in \{10,20,50,100,150\}$ and the maximum fraction of missing leaves (from the union of leaves from all trees in the set) Ml $\in \{0.1, 0.2, 0.3\}$.

To generate an instance, we sampled one tree at a time from the full data set uniformly at random, and depending on whether it was consistent with the fixed values for parameters L and Ml we added it to the instance or discarded it and sampled another

Table 7.3: Number of instances extracted from the *Bacterial and Archaeal Genomes* data set, for different combinations of parameters $|\mathcal{T}|$, L, and MI.

	$L=10$			$L=20$			$L=50$			$L=100$			$L=150$		
MI / $\lvert\mathcal{T}\rvert$	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
2	10	10	10	10	10	10	6	10	10	10	10	10	9	10	10
4	10	10	10	10	10	10	0	7	10	10	10	10	0	7	10
6	10	10	10	10	10	10	0	3	10	10	10	10	0	0	9
10	10	10	10	10	10	10	0	0	6	10	9	10	0	0	3
20	5	10	10	7	7	10	0	0	0	4	9	10	0	0	0
30	6	10	10	7	6	10	0	0	0	1	6	8	0	0	0
40	2	10	9	6	7	5	0	0	0	1	3	8	0	0	0
50	3	9	8	6	5	8	0	0	0	0	3	8	0	0	0
60	1	6	8	5	6	6	0	0	0	0	1	7	0	0	0

Table 7.4: Number of (modified) small instances extracted from the *Bacterial and Archaeal Genomes* data set TreeChild was able to solve within a time limit of 2 hours using 16 cores.

	$L=10$			$L=20$			$L=50$			$L=100$			$L=150$		
MI / $\lvert\mathcal{T}\rvert$	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
2	10	9	10	10	10	10	6	7	1	1	0	0	0	0	0
4	10	10	10	8	5	4	0	0	0	0	0	0	0	0	0
6	10	9	10	0	1	0	0	0	0	0	0	0	0	0	0

tree until we reached the predetermined number $|\mathcal{T}|$ of trees. In more detail, for every sampled tree T we checked whether the number of its leaves was between $(1-\mathrm{MI})L$ and L, whether the union of its leaves and the leaves of the trees already selected for that instance was of size at most L and the difference between the leaf set of T and such union was at most $100 * \mathrm{MI}\%$. If an instance could not be completed within a timeframe of 10 minutes, we aborted the search and started generating a new instance with the same parameters from scratch. We aimed at generating 10 instances for each combination of parameters $|\mathcal{T}|$, L and MI, however, for some combinations we did not find enough trees with the desired properties to generate as many instances. Table 7.3 reports the number of distinct instances that we were able to extract from the data set for each parameter combination. We consider *small* the instances with $|\mathcal{T}| \in \{2,4,6\}$ and *large* the rest.

EXPERIMENTS ON SMALL INSTANCES

We assess the performance of FHyNCH-MultiML against FHyNCH-TrivialRand on small instances using a baseline obtained by applying TreeChild [3] to a modified version of the same instance. The modification is needed because TreeChild can only be applied to a tree set of binary trees with the same leaf sets. To generate these modified instances, the first step is to add as many missing leaves as possible as follows.

Let $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ be the set of input trees and let R be the set of cherries of $\mathcal{T}$. For each T_i and each leaf ℓ missing from T_i, we compute the set $R_i^\ell = \{(\ell, y) \in R \mid y \in T_i\}$ of cherries of $\mathcal{T}$ such that one element is ℓ and the other is a label present in T_i. We then find the cherry $(\ell, z) \in R_i^\ell$ that occurs the most in $\mathcal{T}$ (ties are broken randomly),

[3] available at https://github.com/nzeh/tree_child_code

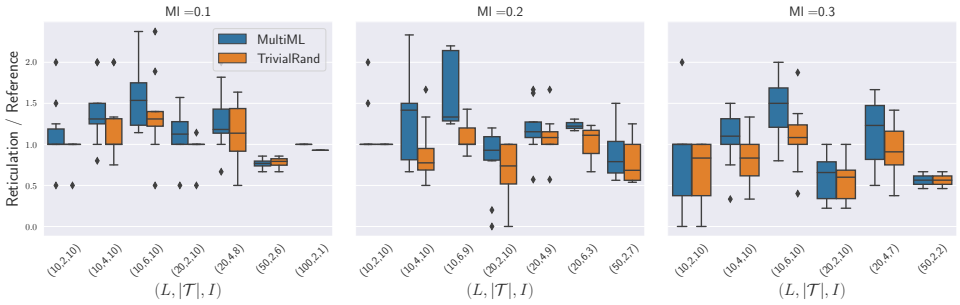


Figure 7.8: Results for the small instances extracted from the *Bacterial and Archaeal Genomes* data set for different values of MI. The reference value for each instance is the best output of TreeChild among the corresponding 10 modified instances described in Section 7.3.2. We do not show results for instances for which TreeChild could not provide a solution within 2 hours, using 16 cores, for any of the modified instances, which was the case for at least one instance within 31 out of 40 small instance groups (see Table 7.4 and compare it with Table 7.3). Parameters L and $|\mathcal{T}|$ are as described in Section 7.3.2; I denotes the number of instances that TreeChild was able to solve for each instance group. Values of the ratio results/reference smaller than 1 indicate that TreeChild did not return the true optimum for some instances because of the (unavoidable) artificial constraint introduced in the modified instances.

add ℓ as another child of the parent of z in T_i and remove ℓ from M_i. If after applying this procedure for all missing leaves of T_i some leaves are still missing, we add them randomly.

In an effort to minimize the bias introduced in the results because of this randomized step, we generated 10 different modified instances for each of the original instances, ran TreeChild on each of them and used the result for the best of these modified instances as a reference value for the original instance. Before doing so, however, we make all the trees of all the modified instances binary by using the dedicated method that can be found in the TreeChild repository [4]. We remark that the trees of the modified instances display those of the original instances by construction (they can be obtained reverting the procedure, i.e, contracting edges and deleting leaves) thus the solutions obtained by applying TreeChild to the modified instances are always feasible for the original ones, although we cannot guarantee their optimality.

For each of the original instances I, we ran FHyNCH-MultiML once, selected the best of the $\min\{100, x(I)\}$ runs of FHyNCH-TrivialRand (recall that $x(I)$ denotes the number of runs of FHyNCH-TrivialRand that can be completed in the time required for a single run of FHyNCH-MultiML) and divided these results by the best value returned by TreeChild among the 10 modified instances obtained from I. We summarize the results in Figure 7.8. From the experiments on synthetic data, it was already clear that the performance of FHyNCH-TrivialRand is close to that of FHyNCH-MultiML on small enough instances. The smallest instances of the synthetic data set, which consist of 20, 50, or 100 trees with 20 leaves each, are much larger than the small instances of the real data set, which consist of only 2, 4, or 6 trees each: for the latter, the good performance of FHyNCH-TrivialRand becomes more pronounced, its results being always comparable or better than those of FHyNCH-MultiML. This is because when the size of the leaf sets

[4] https://github.com/nzeh/tree_child_code

Table 7.5: Running times of FHyNCH-MultiML (MML) and TreeChild (TC) for the small instance groups. The first value in each pair is the average time in seconds within the group, the second value is the standard deviation. For each instance group, the average time required by the fastest method is highlighted in bold. A dash indicates an empty instance group; "> t.l." means that the time limit of 2 hours was exceeded. Only 1 core was used to run FHyNCH-MultiML on each instance; 16 cores were used to run TreeChild on each instance.

L	MI	2 trees		4 trees		6 trees	
		MML	TC	MML	TC	MML	TC
10	0.1	(5.6, 0.2)	(**0.1**, 0.1)	(1.6, 0.3)	(**0.5**, 1.0)	(**2.4**, 0.9)	(325.0, 731.9)
	0.2	(1.0, 0.1)	(**0.2**, 0.2)	(1.6, 0.3)	(**0.2**, 0.2)	(**2.4**, 0.9)	(720.5, 2159.8)
	0.3	(1.0, 0.1)	(**0.2**, 0.2)	(2.7, 1.9)	(**0.2**, 0.3)	(**2.2**, 0.6)	(15.7, 27.6)
20	0.1	(4.7, 2.0)	(**0.1**, 0.1)	(**3.1**, 0.9)	(1835.9, 2828.8)	(**6.1**, 0.9)	> t.l.
	0.2	(4.5, 1.1)	(**0.1**, 0.1)	(**3.1**, 0.6)	(1449.3, 2174.8)	(**5.8**, 1.6)	(5408.3, 2891.1)
	0.3	(1.4, 0.1)	(**0.5**, 0.7)	(**7.2**, 0.9)	(3533.1, 3310.1)	(**4.8**, 1.6)	> t.l.
50	0.1	(**3.5**, 0.2)	(82.1, 176.5)	-	-	-	-
	0.2	(**5.4**, 1.4)	(3321.8, 3021.2)	(**7.6**, 0.9)	> t.l.	(**13.7**, 0.5)	> t.l.
	0.3	(**3.2**, 0.6)	(5846.6, 2712.4)	(**8.6**, 2.4)	> t.l.	(**11.8**, 3.9)	> t.l.
100	0.1	(**8.6**, 0.8)	(6480.3, 2159.1)	(**20.4**, 1.5)	> t.l.	(**39.6**, 3.4)	> t.l.
	0.2	(**8.8**, 1.1)	> t.l.	(**21.0**, 1.6)	> t.l.	(**40.4**, 3.9)	> t.l.
	0.3	(**7.5**, 1.0)	> t.l.	(**20.8**, 4.0)	> t.l.	(**36.2**, 3.9)	> t.l.
150	0.1	(**13.7**, 0.3)	> t.l.	-	-	-	-
	0.2	(**13.1**, 1.5)	> t.l.	(**30.2**, 3.1)	> t.l.	-	-
	0.3	(**11.5**, 0.8)	> t.l.	(**26.9**, 2.4)	> t.l.	(**47.2**, 6.5)	> t.l.

and the number of trees are not too large, multiple runs of FHyNCH-TrivialRand can explore a significant part of the solution space and thus return a good enough solution.

Running time. Table 7.5 reports the average and standard deviation of the running times of FHyNCH-MultiML and TreeChild for each of the small instance groups of Table 7.3. Following van Iersel et al. [201], we imposed a time limit of 2 hours for the execution of TreeChild. We used 16 cores to run TreeChild and only 1 core to run FHyNCH-MultiML, which is single-threaded. Note that for the smallest instances consisting of 2 trees with $L \leq 20$ or 4 trees with $L = 10$ TreeChild is on average faster than FHyNCH-MultiML. However, the opposite becomes true as soon as the number of trees and leaves are increased: already for instances of 6 trees with $L = 10$, TreeChild is slower than FHyNCH-MultiML on average by one or two orders of magnitude; and it is slower by three orders of magnitude or exceeds the time limit for instances with at least 4 trees with $L \geq 20$ or just 2 trees with $L \geq 50$. FHyNCH-MultiML requires, on average less than a minute for all these instance groups.

EXPERIMENTS ON LARGE INSTANCES

In contrast with the small instances, no exact methods could solve any of the larger instances, not even when made binary and with equal leaf sets. It is thus not possible to compute any reference value for these instances. We simply compare the performance of FHyNCH-MultiML against FHyNCH-TrivialRand computing their *relative errors*, defined as follows. Let $r_{ML}(I)$ and $r_{TR}(I)$ be the number of reticulations output by FHyNCH-MultiML and FHyNCH-TrivialRand, respectively, for the same instance I, and let $m = \min\{r_{ML}(I), r_{TR}(I)\}$. The relative error of FHyNCH-MultiML against FHyNCH-TrivialRand for instance I is given by $\frac{r_{ML}(I)-m}{m}$; likewise, the relative error of FHyNCH-TrivialRand against FHyNCH-MultiML is $\frac{r_{TR}(I)-m}{m}$. The relative error of one method

Table 7.6: Results for the experiments on the large instances extracted from the *Bacterial and Archaeal Genomes* data set. FHyNCH-MultiML and FHyNCH-TrivialRand are denoted by MML and TR, respectively; the results in columns labeled MML report the mean relative error (in %) of the results of FHyNCH-MultiML against FHyNCH-TrivialRand; and symmetrically for the columns labeled TR. We highlight in bold the smallest of the two errors for each instance group, identifying the best-performing method for each group. Dashes denote empty instance groups (see also Table 7.3).

L	MI	10 trees		20 trees		30 trees		40 trees		50 trees		60 trees	
		MML	TR	MML	TR	MML	TR	MML	TR	MML	TR	MML	TR
10	0.1	**7.8**	8.0	9.0	**2.9**	**1.7**	10.4	28.6	**0.0**	7.8	**0.0**	**0.0**	6.7
	0.2	7.0	**1.4**	**11.5**	12.2	17.5	**2.3**	**8.0**	9.9	9.1	**7.3**	9.5	**7.2**
	0.3	13.1	**0.8**	7.5	**4.7**	12.0	**9.1**	9.1	**2.2**	**1.6**	9.1	8.7	**8.2**
20	0.1	7.9	**3.6**	**2.7**	7.5	**2.6**	6.0	**0.1**	11.2	**2.3**	9.2	**0.1**	11.3
	0.2	4.6	**1.8**	4.5	**3.8**	**2.0**	5.8	**0.0**	9.4	**0.2**	8.4	**0.0**	9.2
	0.3	8.4	**0.6**	**2.1**	9.2	**6.7**	9.8	3.8	**3.5**	**2.4**	19.9	**2.7**	7.8
50	0.3	5.1	**1.0**	-	-	-	-	-	-	-	-	-	-
100	0.1	**0.0**	15.9	**0.0**	19.1	**0.0**	20.4	**0.0**	16.9	-	-	-	-
	0.2	**0.0**	19.4	**0.0**	19.5	**0.0**	18.4	**0.0**	17.1	**0.0**	17.6	**0.0**	33.5
	0.3	**0.0**	20.3	**0.0**	18.7	**0.0**	19.7	**0.0**	24.3	**0.0**	22.0	**0.0**	24.0
150	0.3	**0.0**	12.2	-	-	-	-	-	-	-	-	-	-

against the other is 0 whenever the method is the best-performing one. We computed these values for each instance, averaged them over all instances within each instance group, and rescaled them to express them as a percentage. The results are shown in Table 7.6. Note that when the mean relative error is 0.0 for some method in some instance group, by definition, that method is the best-performing one for all the instances within the group. It is thus immediately evident that FHyNCH-MultiML is systematically the best method for any instance with $L \geq 100$ and any number of trees and missing leaves. For these instance groups, the mean relative error for FHyNCH-TrivialRand ranged between 12.2% and 33.5%. Once again confirming the behavior observed for synthetic data, FHyNCH-TrivialRand performs the best on small instances, its results getting worse with increasing values of parameters L and $|\mathcal{T}|$. In particular, FHyNCH-TrivialRand is the best-performing method, on average, for all instance groups with $|\mathcal{T}| = 10$ and $L \leq 50$. Increasing L and $|\mathcal{T}|$, FHyNCH-MultiML outperforms FHyNCH-TrivialRand in more and more instance groups: in particular, it is the best-performing method for all groups with $L \geq 20$ and $|\mathcal{T}| \geq 50$, and it is the best one for 7 out of 9 instance groups with $L = 20$ and $|\mathcal{T}| \in \{20, 30, 40\}$.

Running time. In Table 7.7 we report the average running time of FHyNCH-MultiML within each of the large instance groups of Table 7.3. Noticeably, the average running time for the group with the largest instances (60 trees with up to 100 leaves and 30% missing leaves) is under 15 minutes.

Table 7.7: Running times for the large instances extracted from the *Bacterial and Archaeal Genomes* data set. For each instance group, we give the average running time in seconds. Dashes indicate empty instance groups.

L	MI	10 trees	20 trees	30 trees	40 trees	50 trees	60 trees
10	0.1	7.5	9.6	9.6	12.4	18.0	17.0
	0.2	3.2	5.9	7.7	12.7	21.2	16.6
	0.3	3.5	5.0	9.3	12.6	18.6	16.1
20	0.1	11.2	24.3	39.8	64.5	79.8	124.4
	0.2	9.7	19.1	32.2	57.8	69.0	106.9
	0.3	9.4	21.1	27.1	49.2	57.5	75.6
50	0.3	23.0	-	-	-	-	-
100	0.1	75.6	200.4	348.2	458.6	-	-
	0.2	79.5	195.4	346.5	511.4	683.8	856.7
	0.3	76.6	185.4	329.7	481.6	661.8	833.3
150	0.3	106.9	-	-	-	-	-

7.4. CONCLUSIONS

We presented FHyNCH-MultiML, the first heuristic scheme specifically designed to solve the hybridization problem for large sets of multifurcating phylogenetic trees with missing leaves. FHyNCH-MultiML combines the use of two suitably designed machine-learning models with the technique of cherry picking. Experiments on synthetically generated data sets suggest that the results obtained with our method are qualitatively good, given the hardness of the problem: the number of reticulations in the generated networks is always within a small constant factor from the number of reticulations of the network the trees were sampled from. These results are particularly impressive in the case of large inputs consisting of 100 multifurcating trees on a set of 100 taxa with missing leaves. Although it is hard to evaluate the performance of the method on real data because of the lack of reference values (since, before this work, no method existed for this problem) we show that on large enough instances FHyNCH-MultiML is systematically better than repeating a randomized heuristic many times and choosing the best solution.

This work shows the potential for combining machine learning with cherry picking for phylogenetic reconstruction. The major advantage of this approach is its versatility. Indeed, the method presented here can be applied to an arbitrary phylogenetic tree data set. This is an important step forward in the field of phylogenetic networks since all previous methods were limited to restricted types of data. Hence, an important next step is to train the model on very large amounts of data to further improve its performance. Also, the use of more complex machine-learning models, such as graph neural networks, could be investigated. In addition, although our method uses branch lengths of input trees within the algorithm to predict which cherries to pick, it does not yet use them to predict the branch lengths of the output network. Finally, in this chapter, we have only evaluated the method in terms of the number of reticulations of the constructed network. In future work, it is important to analyze how close the constructed networks are to the original simulated network, topologically, for example using tail-moves [107] with edge-insertions/deletions.

8

Proximity Measure for Orchard Networks

Phylogenetic networks are used to represent the evolutionary history of species. Recently, the new class of orchard networks was introduced, which were later shown to be interpretable as trees with additional horizontal arcs. This makes the network class ideal for capturing evolutionary histories that involve horizontal gene transfers.

Studied problem: *We study the minimum number of additional leaves needed to make a network orchard. We demonstrate that computing this proximity measure for a given network is NP-hard and describe a tight upper bound. We also give an equivalent measure based on vertex labelings to construct a mixed integer linear programming formulation. Our experimental results, which include both real-world and synthetic data, illustrate the efficiency of our implementation.*

Relevance to this thesis: *Unlike the other chapters, no data-driven algorithm is proposed to solve a problem more efficiently. In this chapter, we study the complexity of the previously mentioned problem for its theoretical merit, which is also biologically motivated. Next to that, this chapter gives way for future research, in which the cherry-picking heuristics of the previous chapters and methods in this chapter can be combined. More information on this research direction is described in Chapter 9, Open Problem 5.*

This chapter is based on van Iersel et al. [203] published in the **International Workshop on Algorithms in Bioinformatics (WABI 2023)**. In collaboration with Leo van Iersel, Mark Jones, and Yukihiro Murakami. The code is available at `https://github.com/estherjulien/OrchardProximity`.

8.1. INTRODUCTION

Phylogenetic trees are used to represent the evolutionary history of species. While they are effective for illustrating speciation events through vertical descent, they are insufficient in representing more intricate evolutionary processes. Reticulate (net-like) events such as hybridization and horizontal gene transfer (HGT) can give rise to signals that cannot be represented on a single tree [89, 217]. In light of this, phylogenetic networks have gained increasing attention due to their capability in elucidating reticulate evolutionary processes.

Phylogenetic networks are often categorized into different classes based on their topological features. These are often motivated computationally, but some classes are also defined based on their biological relevance [161]. Classical examples of network classes involve the *tree-child networks* [49] and the *tree-based networks* [81]. Roughly speaking, tree-child networks are those where every vertex has passed on a gene via vertical descent to an extant species, and tree-based networks are those obtainable from a tree by adding so-called *linking arcs* between tree arcs. Recent developments have culminated in the introduction of *orchard networks*, which lie – inclusion-wise – between the two aforementioned network classes [70, 109]. The class has shown to be both algorithmically attractive and biologically relevant; they are defined as networks that can be reduced to a single leaf by a series of so-called *cherry-picking operations*, and they were shown to be networks that can be obtained by adding horizontal arcs to trees (where the tree is drawn with the root at the top and arcs pointing downwards) [200]. Such horizontal arcs can be used to model HGT events, making orchard networks especially apt in representing evolutionary scenarios where every reticulate event is a horizontal transfer. Orchard networks have also been characterized statically based on so-called *cherry covers* [202].

When considering a non-orchard network, a natural question arises: how many additional leaves are required to transform the network into one that is orchard? From a biological standpoint, this question can be interpreted as asking how many extinct species or unsampled taxa need to be introduced into the network to yield a scenario where every reticulation represents an HGT event. Given that HGT is the primary driver of reticulate evolution in bacteria [94], this is an essential inquiry. We provide a network of a few fungi species in Figure 8.1, which requires one additional leaf to make it orchard. Formally speaking, the problem of computing this leaf addition measure is as follows.

$L_{\mathcal{OR}}$-DISTANCE (DECISION)
Input: A network N on a set of taxa X and a natural number k.
Decide: Can N be made orchard with at most k leaf additions?

In related research, the leaf addition measure has been investigated for other network classes. It has been shown that tracking down the minimum leaf additions to make a network tree-based can be done in polynomial time [80]. In the same paper, it was shown that the leaf addition measure was equivalent to two other proximity measures, namely those based on spanning trees and disjoint path partitions. Linz et al. [136] studied this problem for networks with labeled linking arcs. The authors prove that the decision problem of leaf additions to turn "HGT networks", which resemble tree-based net-

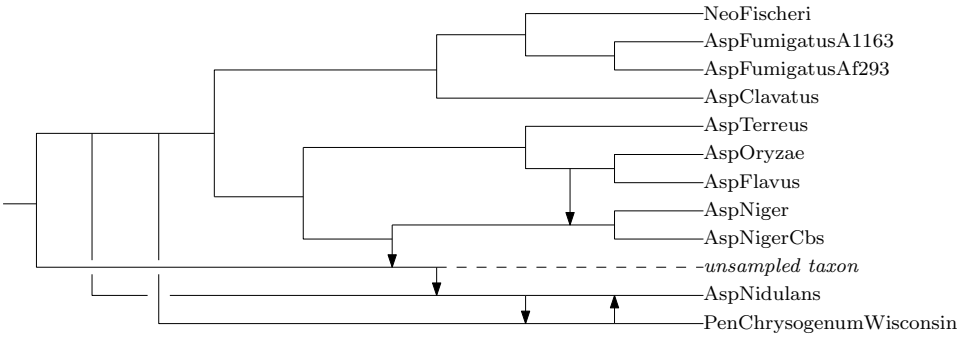


Figure 8.1: A network on 11 different taxa (excluding *unsampled taxon*) of fungi including 5 reticulations, which is part of a larger network from Szöllősi et al. [192]. The directed arcs in the figure are linking arcs, which represent gene transfer highways. In order to make all linking arcs horizontal, we require an additional leaf (*unsampled taxon*) to represent the evolutionary history. To see that the network needs the leaf *unsampled taxon* in order to have only horizontal linking arcs, we refer the interested reader to Section 8.A.

works with a fixed *base tree* and fixed linking arcs, into "temporal HGT" networks, is NP-complete [136, Theorem 3.5]. A base tree of a network is a subtree that covers all nodes and has the same leaf set as the network (see Section 8.2.3). Temporal HGT networks closely resemble orchard networks, except that orchard networks permit multiple choices for the base tree and linking arcs.

The same question was posed for the unrooted variant (where the arcs of the network are undirected), for which the problem turned out to be NP-complete [73]. A total of eight proximity measures were introduced in this latter paper, including those based on edge additions and rearrangement moves. Instead of considering leaf additions, some manuscripts have even considered leaf deletions (in general, vertex deletions) as proximity measures for the class of so-called *edge-based networks* [74]. Finally, for orchard networks, a recent bachelor's thesis compared how the leaf addition proximity measure differs in general from another proximity measure based on arc deletions [190].

In this chapter, we show that the leaf addition proximity measure can be computed in polynomial time for the class of tree-child networks, and we give a more efficient algorithm for computing the measure for tree-based networks. We show that $L_{\mathscr{OR}}$-DISTANCE is NP-complete by a polynomial-time reduction from DEGREE-3 VERTEX COVER. To model the problem as a *mixed integer linear program* (MILP), we consider a reformulation of the leaf addition measure in terms of vertex labelings. Orchard networks are known to be trees with added horizontal arcs; roughly speaking, this means we can label the vertices of an orchard network so that every vertex of indegree-2 has exactly one incoming arc whose end-vertices have the same labels. The reformulated measure, called the *vertical arcs* proximity measure, counts – over all possible vertex labelings (defined formally in Section 8.6.1) – the minimum number of indegree-2 vertices with only non-horizontal incoming arcs. Our experimental results are promising, as the real-world cases are solved in a fraction of a second. Furthermore, the model also scales well to larger synthetic data.

The structure of the chapter is as follows. In Section 8.2, we provide all necessary definitions and characterizations of orchard networks and tree-based networks. In Section 8.3, we formally introduce the leaf addition measure for the classes of tree-child, orchard, and tree-based networks. In Section 8.4 we show that $L_{\mathscr{OR}}$-DISTANCE is NP-complete (Theorem 8.6). In Section 8.5, we give a sharp upper bound for the leaf addition proximity measure. In Section 8.6 we give a reformulation of the leaf addition measure to describe the MILP to solve $L_{\mathscr{OR}}$-DISTANCE, and in Section 8.6.3, experimental results are shown for the MILP, applied to real and simulated networks. In Section 8.7, we give a brief discussion of our results and discuss potential future research directions. We include proofs for select results in Section 8.A.

8.2. PRELIMINARIES

A *binary directed phylogenetic network* on a non-empty set X is a directed acyclic graph with

- a single *root* of indegree-0 and outdegree-1;
- *tree vertices* of indegree-1 and outdegree-2;
- *reticulations* of indegree-2 and outdegree-1;
- *leaves* of indegree-1 and outdegree-0, that are labeled bijectively by elements of X.

For the sake of brevity, we shall refer to binary directed phylogenetic networks simply as *networks*. Throughout the chapter, assume that N is a network on some non-empty set X where $|X| = n$, unless stated otherwise. Networks without reticulations are called *trees*. Tree vertices and reticulations may sometimes collectively be referred to as *internal vertices*.

The arc uv of a network is a *root arc* if u is the root of the network. An arc uv of a network is a *reticulation arc* if v is a reticulation, and a *tree arc* otherwise. We say that a vertex u is a *parent* of another vertex v if uv is an arc of the network; in such instances we call v a *child* of u. Also, we say that u and v are the *tail* and the *head* of the arc uv, respectively. In other words, we may rewrite arcs as $uv = \text{tail}(uv)\,\text{head}(uv)$. The *neighbours* of v refer to the set of vertices that are parents or children of v. We also say that vertices u and v are *siblings* if they share a parent.

In what follows, we shall define graph operations based on vertex and arc deletions. To make sure resulting graphs remain networks, we follow-up every graph operation with a *cleaning up* process. Formally, we *clean up* a network by applying the following until none is applicable.

- Suppress an indegree-1 outdegree-1 vertex (e.g., if uv and vw are arcs where v is an indegree-1 outdegree-1 vertex, we suppress v by deleting the vertex v and adding an arc uw).
- Replace parallel arcs by a single arc (e.g., if uv is an arc twice in a network, delete one of the arcs uv).

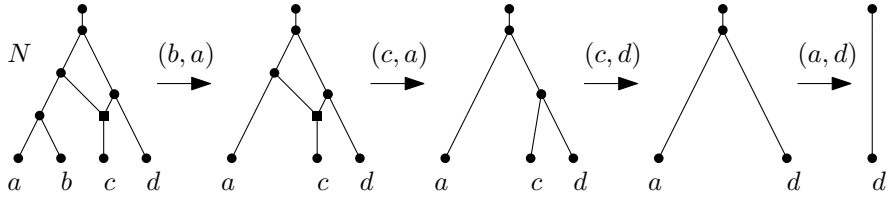


Figure 8.2: An example of an orchard network N that is reduced by a sequence $(b,a)(c,a)(c,d)(a,d)$. The network N contains a cherry (b,a) and a reticulated cherry (c,d). Subsequent networks are those obtained by a single cherry-picking reduction from the previous network. For example, the second network $N(b,a)$ is obtained from N by removing the leaf b and cleaning up. Note that the network is also tree-child.

We observe that deleting a tree arc and cleaning up results in a graph containing two indegree-0 vertices. On the other hand, deleting a reticulation arc and cleaning up results in a network. Therefore, we shall use arc deletions to mean reticulation arc deletions.

8.2.1. TREE-CHILD NETWORKS

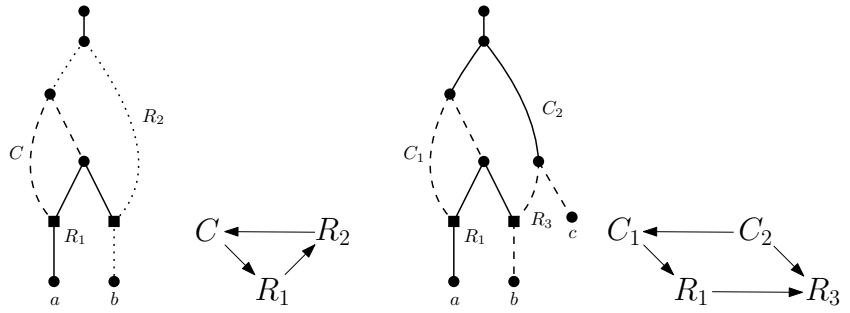
A network is *tree-child* if every non-leaf vertex has a child that is a tree vertex or a leaf. We call an internal vertex of a network an *omnian* if all of its children are reticulations [110]. It follows from definition that a network is tree-child if and only if it contains no omnians.

8.2.2. ORCHARD NETWORKS

To define orchard networks, we must first define cherries and reticulated cherries, as well as operations to reduce them. See Figure 8.2 for the illustration of the following definitions. Let N be a network. Two leaves x and y of N form a *cherry* if they are siblings. In such a case, we say that N *contains* a cherry (x,y) or a cherry (y,x). Two leaves x and y of N form a *reticulated cherry* if the parent p_x of x is a reticulation and the parent of y is also a parent of p_x. In such a case, we say that N *contains* a reticulated cherry (x,y). *Reducing the cherry* (x,y) *from* N is the process of deleting the leaf x and cleaning up. *Reducing the reticulated cherry* (x,y) *from* N is the process of deleting the arc from the parent of y to the parent of x and cleaning up. In both cases, we use $N(x,y)$ to denote the resulting network.

A network N is *orchard* if there is a sequence $S = (x_1,y_1)(x_2,y_2)\dots(x_k,y_k)$ such that NS is a network on a single leaf y_k. It has been shown that the order in which (reticulated) cherries are reduced from an orchard network does not matter, as repeatedly picking any reducible pair will create a shortest cherry-picking sequence [70, 109]. Apart from this recursive definition, orchard networks have been characterized based on cherry covers (arc decompositions) [202] and vertex labelings [200]. We include both characterizations here.

Cherry covers (see van Iersel et al. [202] for more details): A *cherry shape* is a subgraph on three distinct vertices x,y,p with arcs px and py. The *internal vertex* of a cherry shape is p, and the *endpoints* are x and y. A *reticulated cherry shape* is a subgraph on four distinct vertices x,y,p_x,p_y with arcs p_xx, p_yp_x, p_yy, such that p_x is a reticulation in the net-



(a) Network N_1 (b) Cherry cover aux. graph of N_1 (c) Network N_2 (d) Cherry cover aux. graph of N_2

Figure 8.3: A cherry cover example. (a) A network N_1 on $\{a,b\}$ with a cherry cover $\{C,R_1,R_2\}$. (b) The (cyclic) auxiliary graph of N_1 based on the cherry cover of (a). (c) The network N_2 obtained from N_1 by adding a leaf c, with a cherry cover $\{C_1,C_2,R_1,R_3\}$ (d) The (acyclic) auxiliary graph of N_2 based on the cherry cover of (d).

work. The *internal vertices* of a reticulated cherry shape are p_x and p_y, and the endpoints are x and y. The *middle arc* of a reticulated cherry shape is $p_y p_x$. We will often refer to cherry shapes and the reticulated cherry shapes by their arcs (e.g., we would denote the above cherry shape $\{p_x x, p_y y\}$ and the reticulated cherry shape $\{p_x x, p_y p_x, p_y y\}$). We say that an arc uv is covered by a cherry or reticulated cherry shape B if $uv \in B$. A *cherry cover* of a binary network is a set P of cherry shapes and reticulated cherry shapes, such that each arc except for the root arc is covered exactly once by P. In general, a network can have more than one cherry cover.

We define the *cherry cover auxiliary graph* $G = (V, A)$ of a cherry cover as follows. For all shapes $B \in P$, we have $v_B \in V$. A shape $B \in P$ is *directly above* another shape $C \in P$ if B and C contain a same vertex v, such that v is an endpoint of B and an internal vertex of C. Then, $v_B v_C \in A$ (adapted from van Iersel et al. [202, Definition 2.13]). We say that a cherry cover is *cyclic* if its auxiliary graph has a cycle. We call it *acyclic* otherwise. See fig. 8.3 for an illustration of a cyclic and acyclic cherry cover.

Theorem 8.1 (Theorem 4.3 of van Iersel et al. [202])**.** *A network N is orchard if and only it has an acyclic cherry cover.*

Non-Temporal labelings: Let N be a network with vertex set $V(N)$. A *non-temporal labeling*[1] of N is a labeling $t : V(N) \to \mathbb{R}$ such that

- for all arcs uv, $t(u) \leq t(v)$ and equality is allowed only if v is a reticulation;
- for each internal vertex u, there is a child v of u such that $t(u) < t(v)$;
- for each reticulation r with parents u and v, at most one of $t(u) = t(r)$ or $t(v) = t(r)$ holds.

[1] This is named in contrast to *temporal representations* of Baroni et al. [17]. There, it was required for the endpoints of every reticulation arc to have the same label.

Observe that every network (orchard or not) admits a non-temporal labeling by labeling each vertex by its longest distance from the root (assuming each arc is of weight 1).

Under non-temporal labelings, we call an arc *horizontal* if its endpoints have the same label; we call an arc *vertical* otherwise. By definition, only reticulation arcs can be horizontal. We say that a non-temporal labeling is an *HGT-consistent labeling* if every reticulation is incident to exactly one incoming horizontal arc. We recall the following key result.

Theorem 8.2 (Theorem 1 of van Iersel et al. [200])**.** *A network is orchard if and only if it admits an HGT-consistent labeling.*

8.2.3. TREE-BASED NETWORKS

A network N is *tree-based* with *base tree* T if it can be obtained from T in the following steps.

1. Replace some arcs of T by paths, whose internal vertices we call *attachment points*; each attachment point is of indegree-1 and outdegree-1.
2. Place arcs between attachment points, called *linking arcs*, so that the graph contains no vertices of total degree greater than 3, and so that it remains acyclic.
3. Clean up.

The relation between the classes of tree-child, orchard, and tree-based networks can be stated as follows.

Lemma 8.1 (Janssen and Murakami [109] and Corollary 1 of van Iersel et al. [200])**.** *If a network is tree-child, then it is orchard. If a network is orchard, then it is tree-based.*

We include here a static characterization of tree-based networks based on an arc partition, called *maximum zig-zag trails* [97, 230]. Let N be a network. A *zig-zag trail* of length k is a sequence $(a_1, a_2, \ldots, a_k)$ of arcs where $k \geq 1$, where either $\mathrm{tail}(a_i) = \mathrm{tail}(a_{i+1})$ or $\mathrm{head}(a_i) = \mathrm{head}(a_{i+1})$ holds for $i \in [k-1] = \{1, 2, \ldots, k-1\}$. We call a zig-zag trail Z *maximal* if there is no zig-zag trail that contains Z as a subsequence. Depending on the nature of $\mathrm{tail}(a_1)$ and $\mathrm{tail}(a_k)$, we have four possible maximal zig-zag trails.

- *Crowns*: $k \geq 4$ is even and $\mathrm{tail}(a_1) = \mathrm{tail}(a_k)$ or $\mathrm{head}(a_1) = \mathrm{head}(a_k)$.
- *M-fences*: $k \geq 2$ is even, it is not a crown, and $\mathrm{tail}(a_i)$ is a tree vertex for every $i \in [k]$.
- *N-fences*: $k \geq 1$ is odd and $\mathrm{tail}(a_1)$ or $\mathrm{tail}(a_k)$, but not both, is a reticulation. By reordering the arcs, assume henceforth that $\mathrm{tail}(a_1)$ is a reticulation and $\mathrm{tail}(a_k)$ a tree vertex.
- *W-fences*: $k \geq 2$ is even and both $\mathrm{tail}(a_1)$ and $\mathrm{tail}(a_k)$ are reticulations.

We call a set S of maximal zig-zag trails a *zig-zag decomposition* of N if the elements of S partition all arcs, except for the root arc, of N.

Lemma 8.2 (adapted from Corollary 4.6 of Hayamizu [97])**.** *Let N be a network. Then N is tree-based if and only if it has no W-fences.*

Theorem 8.3 (adapted from Theorem 4.2 of Hayamizu [97])**.** *Any network N has a unique zig-zag decomposition.*

Theorem 8.4 (adapted from Theorem 3.3 of van Iersel et al. [202])**.** *Let N be a network. Then N is tree-based if and only if it has a cherry cover.*

8.3. LEAF ADDITION PROXIMITY MEASURE

Let N be a network on X. *Adding a leaf $x \notin X$ to an arc e of N* is the process of adding a labeled vertex x, subdividing the arc e by a vertex w (if $e = uv$ then we delete the arc uv, add the vertex w, and add arcs uw and wv), and adding an arc wx. We denote the resulting network by $N + (e, x)$. When the arc e in the above is irrelevant or clear, we simply call this process *adding a leaf x to N*, and denote the resulting network by $N + x$.

In this section, $\mathscr{C}$ will be used to denote a network class. In particular, we shall use $\mathcal{TC}, \mathcal{OR}$, and $\mathcal{TB}$ to denote the classes of tree-child networks, orchard networks, and tree-based networks, respectively. Let $L_{\mathscr{C}}(N)$ denote the minimum number of leaf additions required to make the network N a member of $\mathscr{C}$. We first show that computing $L_{\mathcal{TC}}(N)$ and $L_{\mathcal{TB}}(N)$ can be done in polynomial time.

Lemma 8.3. *Let N be a network. Then $L_{\mathcal{TC}}(N)$ is equal to the number of omnians. Moreover, N can be made tree-child by adding a leaf to exactly one outgoing arc of each omnian.*

Lemma 8.4. *Let N be a network. Then $L_{\mathcal{TC}}(N)$ can be computed in $O(|N|)$ time.*

It has been shown already that $L_{\mathcal{TB}}(N)$ can be computed in $O(|N|^{3/2})$ time where $|N|$ is the number of vertices in N [80]. This was shown to be solvable in $O(|N|)$ time by adding a leaf to every W-fence [97, Corollary 5.4]. We include the result here for completeness.

Lemma 8.5. *Let N be a network. Then $L_{\mathcal{TB}}(N)$ is equal to the number of W-fences. Moreover, N can be made tree-based by adding a leaf to any arc in each W-fence in N.*

Lemma 8.6. *Let N be a network. Then $L_{\mathcal{TB}}(N)$ can be computed in $O(|N|)$ time.*

Interestingly, computing $L_{\mathcal{OR}}(N)$ proves to be a difficult problem, although the leaf addition proximity measure is easy to compute for its neighbouring network classes. We prove the following in Section 8.4.

▷ **Theorem 8.6.** *Let N be a network. Computing $L_{\mathcal{OR}}(N)$ is NP-hard.*

We also include the following theorem which states that when considering leaf addition proximity measures for orchard networks, it suffices to consider leaf additions to reticulation arcs. We shall henceforth assume that all leaf additions are on reticulation arcs.

Theorem 8.5 (Theorem 4.1 of Susanna [190])**.** *A network N is orchard if and only if the network obtained by adding a leaf to a tree arc of N is orchard.*

The rest of the chapter will now focus on the problem of computing $L_{\mathcal{OR}}(N)$.

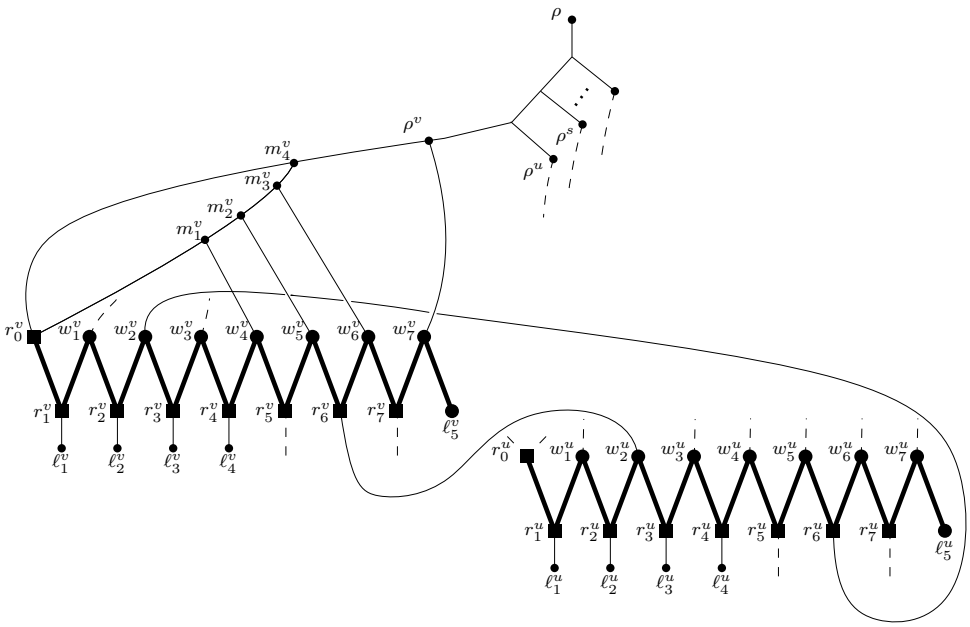


Figure 8.4: Sketch of the network N_G for the case when G contains an edge uv.

8.4. HARDNESS PROOF

In this section, we show that computing $L_{\mathscr{OR}}(N)$ is NP-hard by reducing from degree-3 vertex cover.

DEGREE-3 VERTEX COVER (DECISION)
Input: A 3-regular graph $G = (V, E)$ and a natural number k.
Decide: Does G have a vertex cover with at most k vertices?

$L_{\mathscr{OR}}$-DISTANCE (DECISION)
Input: A network N on a set of taxa X and a natural number k.
Decide: Can N be made orchard with at most k leaf additions?

We now describe the reduction from DEGREE-3 VERTEX COVER to $L_{\mathscr{OR}}$-DISTANCE. For a graph G, let $V(G)$ and $E(G)$ be its vertex and edge sets, respectively. Given an instance (G, k) of DEGREE-3 VERTEX COVER, construct an instance (N_G, k) of $L_{\mathscr{OR}}$-DISTANCE as follows (see Figure 8.4):

1. For each vertex v in $V(G)$, construct a gadget Gad(v) as described below. In what follows, vertices of the form ℓ_i^v are leaves, vertices r_i^v are reticulations, and vertices w_i^v, m_i^v and ρ^v are tree vertices.

   The key structure in Gad(v) is an N-fence with 15 arcs, starting with the arc $\boldsymbol{r_0^v r_1^v}$, then followed by arcs of the form $\boldsymbol{w_i^v r_i^v}$, $\boldsymbol{w_i^v r_{i+1}^v}$ for each $i \in [6]$, and finally the arcs

$\boldsymbol{w_7^v r_7^v, w_7^v \ell_5^v}$. This set of arcs, in bold type, is called the *principal part* of Gad(v). In addition,the reticulations $r_1^v, r_2^v, r_3^v, r_4^v$ have leaf children $\ell_1^v, \ell_2^v, \ell_3^v, \ell_4^v$ respectively.

Above the principal part of Gad(v), add a set of tree vertices $m_1^v, m_2^v, m_3^v, m_4^v, \rho^v$ with the following children: m_1^v has children r_0^v and w_4^v, m_2^v has children m_1^v and w_5^v, m_3^v has children m_2^v and w_6^v, m_4^v has children m_3^v and r_0^v, and ρ^v has children m_4^v and w_7^v (see Figure 8.4).

This completes the construction of Gad(v). Note that so far, the vertices w_1^v, w_2^v, w_3^v have no incoming arcs, and r_5^v, r_6^v, r_7^v have no outgoing arcs. Such arcs will be added later to connect different gadgets together.

2. Connect the vertices ρ^v from each Gad(v) as follows: take some ordering of the vertices $\{v_1, \ldots, v_g\}$ of G. Add a vertex ρ and vertices s_i for $i \in [g-1]$. Add arcs ρs_1 and also arcs from the set $\{s_i s_{i+1} : i \in [g-2]\}$, as well as arcs from the set $\{s_i \rho^{v_i} : i \in [g-1]\}$, and finally an arc $s_{g-1}\rho^{v_g}$.

3. Next add arcs between the gadgets corresponding to adjacent vertices in G, in the following way: for every pair of adjacent vertices u, v in G, add an arc connecting one of the vertices r_5^u, r_6^u, r_7^u in Gad(u) to one of the vertices w_1^v, w_2^v, w_3^v in Gad(v) (and, symmetrically, an arc connecting one of r_5^v, r_6^v, r_7^v to one of w_1^u, w_2^u, w_3^u). The exact choice of vertices connected by an arc does not matter, except that we should ensure each vertex is used by such an arc exactly once. Formally: for each vertex v in G with neighbours a, b, c, fix two (arbitrary) mappings $\pi_v : \{a, b, c\} \to \{1, 2, 3\}$ and $\tau_v : \{a, b, c\} \to \{5, 6, 7\}$. Then for each pair of adjacent vertices u, v in G, add an arc from $r^u_{\tau_u(v)}$ to $w^v_{\pi_v(u)}$ (and, symmetrically, add an arc from $r^v_{\tau_v(u)}$ to $w^u_{\pi_u(v)}$).

4. Finally, for each vertex v in G, label the vertices $\{\ell_i^v : i \in [5]\}$ in Gad(v) by ℓ_i^v.

Call the resulting graph N_G; it is easy to see that N_G is directed and acyclic with a single root ρ. Therefore it is a network on the leaf-set $\{\ell_i^v : i \in [5],\ v \in V(G)\}$. As the arcs of N_G are decomposed into M-fences and N-fences, we have the following observation.

Observation 8.1. *Let G be a 3-regular graph and let N_G be the network obtained by the reduction. Then N_G is tree-based.*

By Observation 8.1 and Theorem 8.4, we use freely from now on that N_G has a cherry cover. Before proving the main result, we require some notation and helper lemmas. Let N be a network and let $\hat{N}_i$ be an N-fence of N. In what follows, we shall write $\hat{N}_i := (a_1^i, a_2^i, \ldots, a_{k_i}^i)$, and we will let c_{2j-1}^i denote the child of head(a_{2j-1}^i) for $j \in \left[\frac{k_i-1}{2}\right]$. The first lemma states that although a tree-based network may have non-unique cherry covers, the reticulated cherry shapes that cover arcs of N-fences are fixed.

Lemma 8.7. *Let N be a tree-based network, and let $\hat{N}_1, \hat{N}_2, \ldots, \hat{N}_n$ denote the N-fences of N of length at least 3. Then every cherry cover of N contains the reticulated cherry shapes $\{(\text{head}(a_{2j-1}^i)c_{2j-1}^i), a_{2j}^i, a_{2j+1}^i\}$ for $i \in [n]$ and $j \in \left[\frac{k_i-1}{2}\right]$.*

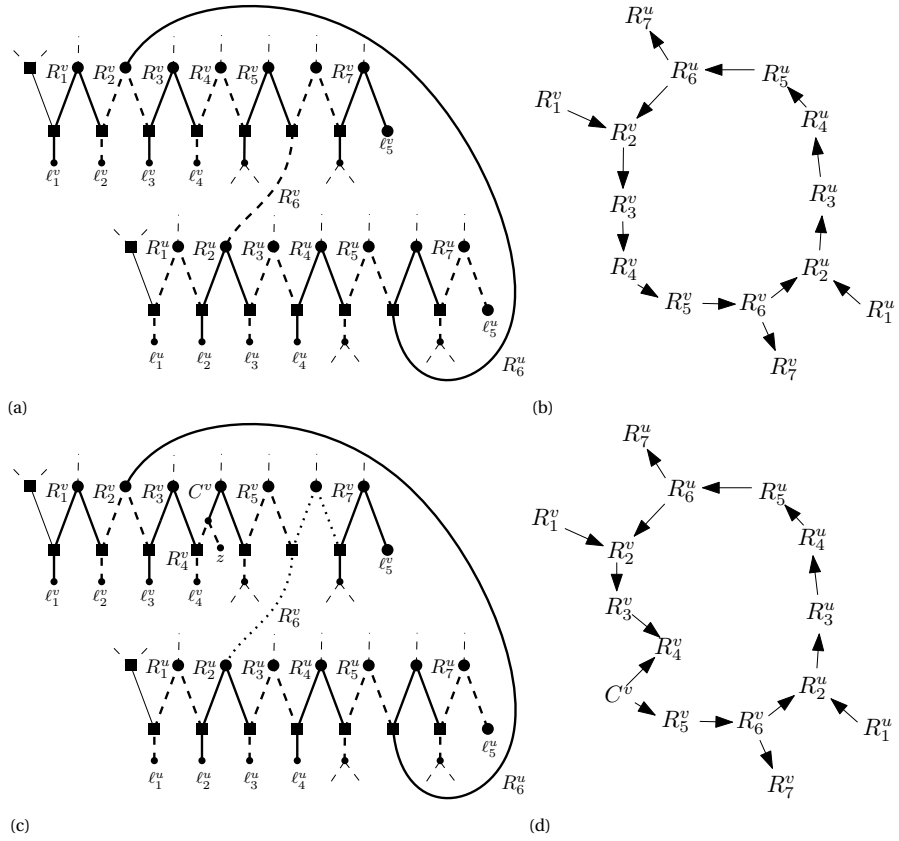


Figure 8.5: Cherry cover of Gad(v) and Gad(u). In (a), the unique cherry cover of the principal part of Gad(v) and Gad(u) is displayed, in (b), the cherry cover auxiliary graph of (a) is given. In (c), the leaf $z \notin X$ is added to the principal part of Gad(v), and one possible cherry cover of the same part of the network is given. And in (d), the cherry cover auxiliary graph of (c) is given.

Note that the principal part of a gadget Gad(v) for every $v \in V(G)$ is an N-fence. Let us denote the principal part of a gadget Gad(v) by $(a_1^v, a_2^v, \ldots, a_{15}^v)$ for all $v \in V(G)$. By Lemma 8.7, a_i^v for $i = 2, \ldots, 15$ and $v \in E(G)$ are covered in the same manner across all possible cherry covers of N_G. Let us denote the reticulated cherry shape that contains a_i^v and a_{i+1}^v by $R_{i/2}^v$ for even $i \in [15]$. Figures 8.5a and 8.5b show an example of the part of cherry cover auxiliary graph containing R_i^v and R_i^u for $i \in [7]$, for some edge uv in G. Note that the cherry shapes form a cycle. The next lemma implies that in fact, such a cycle exists for any edge uv in G.

Lemma 8.8. *Let N be a tree-based network and suppose that for two N-fences $\hat{N}_u := (a_1^u, a_2^u, \ldots, a_{k_u}^u)$ and $\hat{N}_v := (a_1^v, a_2^v, \ldots, a_{k_v}^v)$ of length at least 3, there exist directed paths in N from $\mathrm{head}(a_h^u)$ to $\mathrm{tail}(a_i^v)$ and from $\mathrm{head}(a_j^v)$ to $\mathrm{tail}(a_k^u)$, for even h, i, j, k with $k < h$ and $i < j$. Then every cherry cover auxiliary graph of N contains a cycle.*

In order to remove all possible cycles from a possible cherry cover, it is therefore necessary to disrupt the principal part of either Gad(u) or Gad(v), for any edge uv in G.

Lemma 8.9. *Let G be a 3-regular graph and let N_G be the network obtained by the reduction above. Suppose that A is a set of arcs of N_G, for which adding leaves to every arc in A results in an orchard network. For every edge $uv \in E(G)$, there exists an arc $a \in A$ that is an arc of the principal part of* Gad(u) *or* Gad(v).

To complete the proof of the validity of the reduction, we show that in order to make N_G orchard by leaf additions, it is sufficient (and necessary) to add a leaf z^v to an appropriate arc of Gad(v) for every v in a vertex cover V_{sol} of G (see Figure 8.5c). The key idea is that this splits the principal part of Gad(v) from an N-fence into an N-fence and an M-fence, and this allows us to avoid the cycle in the cherry cover auxiliary graph (see Figure 8.5d).

Lemma 8.10. *Let G be a 3-regular graph and let N_G be the network obtained by the reduction described above. Then G has a minimum vertex cover of size k if and only if $L_{\mathcal{OR}}(N_G) = k$.*

Theorem 8.6. *Let N be a network. The decision problem $L_{\mathcal{OR}}$-*DISTANCE *is NP-complete. Computing $L_{\mathcal{OR}}(N)$ is NP-hard.*

Proof. Suppose we are given a set of arcs A_{sol} of N_G of size at most k. Upon adding leaves to every arc in A_{sol}, we may check that the resulting network is orchard in polynomial time (see Section 6 of Janssen and Murakami [109]). This implies that $L_{\mathcal{OR}}$-DISTANCE is in NP. The reduction from DEGREE-3 VERTEX COVER to $L_{\mathcal{OR}}$-DISTANCE outlined at the start of the section takes polynomial time, since we add a constant number of vertices and arcs for every vertex in the DEGREE-3 VERTEX COVER instance. The NP-completeness of $L_{\mathcal{OR}}$-DISTANCE follows from the equivalence of the two problems shown by Lemma 8.10. The optimization problem of $L_{\mathcal{OR}}$-DISTANCE, i.e., the one of computing $L_{\mathcal{OR}}(N)$ is therefore NP-hard. □

8.5. UPPER BOUND

In the previous section we showed that computing $L_{\mathcal{OR}}(N)$ is NP-hard. Here, we provide a sharp upper bound for $L_{\mathcal{OR}}(N)$. We call a reticulation *lowest* if it has the largest topological distance from the root and *highest* if it has no reticulation ancestors.

Lemma 8.11. *Let N be a network. Suppose there is a highest reticulation r such that all other reticulations have a leaf sibling. Then N is orchard.*

Proof. We prove the lemma by induction on the number of reticulations k. For the base case, observe that a network with one reticulation is tree-child since it has no omnians. A tree-child network is orchard [109], and so this network must be orchard.

Suppose now that we have proven the lemma for all networks with fewer than k reticulations, where $k > 1$. Let N be a network with reticulation set R where $|R| = k$, and suppose there exists a highest reticulation r in N such that all other reticulations have a leaf sibling. Let r denote the highest reticulation as specified in the statement of the lemma. Choose a lowest reticulation $r' \in R \setminus \{r\}$. By assumption, r' has a leaf sibling c.

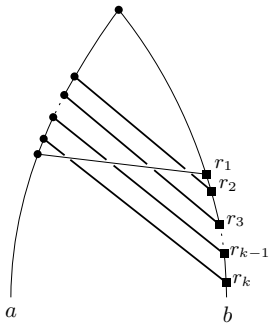


Figure 8.6: A network N on two leaves $\{a, b\}$ with k reticulations $(r_1, \ldots, r_k)$. Observe that $L_{\mathcal{OR}}(N) = k-1$, since the highest reticulation cannot be reduced by cherry picking unless the reticulations below it are first reduced. For each non-highest reticulation, we must add a leaf to one of its incoming arcs to reduce it, which leads to $L_{\mathcal{OR}}(N) = k-1$.

Every vertex below r' must be tree vertices and leaves. Reduce cherries until the child x of r' is a leaf. Then (x, c) is a reticulated cherry; the network N' obtained by reducing this reticulated cherry has $k-1$ reticulations and has a highest reticulation r such that all other reticulations have a leaf sibling. By induction hypothesis, N' must be orchard. Since a sequence of cherry reductions can be applied to N to obtain N', the network N must also be orchard. □

Theorem 8.7. *Let N be a network, and let $r(N)$ denote the number of reticulations. Then $L_{\mathcal{OR}}(N) = 0$ if N is a tree, and otherwise, $L_{\mathcal{OR}}(N) \leq r(N) - 1$, where the bound is sharp.*

Proof. If N is a tree, then it is orchard, and so $L_{\mathcal{OR}}(N) = 0$. So suppose $r(N) > 0$. Let r be a highest reticulation of N, and for every other reticulation, arbitrarily choose one incoming reticulation arc. Add a leaf to each of these reticulation arcs. By Lemma 8.11, the resulting network must be orchard. We have added a leaf for all but one reticulation in N. It follows that $L_{\mathcal{OR}}(N) \leq r(N) - 1$. The network in Figure 8.6 shows that this upper bound is sharp. □

8.6. MILP FORMULATION

To model the problem of computing the leaf addition proximity measure as an MILP, we reformulate the measure in terms of non-temporal labelings.

8.6.1. VERTICAL ARCS INTO RETICULATIONS

By Theorem 8.2, every orchard network can be viewed as a network with a base tree where each of the linking arcs are horizontal. Recall that in terms of non-temporal labelings, this means that there exists a labeling wherein every reticulation has exactly one incoming reticulation arc that is horizontal. Following this definition, we introduce a second orchard proximity measure. Given a non-temporal labeling for a network N, let us use *inrets* to refer to reticulations of N with only vertical incoming arcs. Let $V_{\mathcal{OR}}(N)$ denote the minimum number of inrets over all possible non-temporal labelings.

Observation 8.2. *Let N be a network. A network admits an HGT-consistent labeling if and only if $V_{\mathcal{OR}}(N) = 0$. In other words, a network is orchard if and only if $V_{\mathcal{OR}}(N) = 0$.*

In particular, we show a stronger result that equates the two proximity measures.

Lemma 8.12. *Let N be a network. Then $L_{\mathcal{OR}}(N) = V_{\mathcal{OR}}(N)$.*

Proof. Suppose first that we have a network N with some non-temporal labeling $t : V(N) \to \mathbb{R}$ which gives rise to h inrets. For every inret r with parents u and v, we add a leaf x to the arc ur (this addition is done without loss of generality; the argument also follows by adding the leaf to vr). Since r is an inret, we must have $t(u) < t(r)$ and $t(v) < t(r)$. Letting p_x denote the parent of x, we label $t(p_x) := t(r)$ and $t(x) := t(p_x) + 1$. This ensures that the extension of the map t that includes x and p_x is a non-temporal labeling for $N + x$. Observe that r is no longer an inret in $N + x$, since the arc $p_x r$ is horizontal. Therefore, a leaf addition to an incoming arc of an inret can reduce the number of inrets by exactly one. By repeating this procedure for every inret, it follows that $L_{\mathcal{OR}}(N) \leq V_{\mathcal{OR}}(N)$.

To show the other direction, suppose we can add ℓ leaves to N to make it orchard. By Theorem 8.5, we may assume all such leaves are added to reticulation arcs in the set $\{e_1, \ldots, e_\ell\}$. The resulting network N' has an HGT-consistent labeling $t : V(N') \to \mathbb{R}$ by Theorem 8.2. We claim that the labeling $t|_{V(N)}$ restricted to N is a non-temporal labeling, and that under $t|_{V(N)}$, the number of inrets is at most ℓ. Suppose that a leaf x_i was added to the reticulation arc $e_i = u_i r_i$. Let p_i denote the parent of x_i in the network N'. By definition of HGT-consistent labelings, we must have that $t(u_i) < t(r_i)$, since $u_i p_i r_i$ is a path in N'. Therefore, restricting the labeling to the network obtained from N' by removing the leaf x_i is non-temporal. Furthermore, if v_i is the parent of r_i that is not u_i, we have that one of $v_i r_i$ or $p_i r_i$ must be horizontal in N'. If $v_i r_i$ was horizontal, then r_i still has a horizontal incoming arc upon removing x_i, and the number of inrets does not change. On the other hand, if $p_i r_i$ was horizontal, then $v_i r_i$ must have been a vertical arc. Upon deleting x_i, the reticulation r_i becomes an inret as its other incoming arc $u_i r_i$ is also vertical. Since leaf deletions are local operations, deleting a leaf increases the number of inrets by at most one. By repeating this for each reticulation arc e_i for $i \in [\ell]$, it follows that N contains at most ℓ inrets, and therefore $V_{\mathcal{OR}}(N) \leq L_{\mathcal{OR}}(N)$. □

8.6.2. MILP FORMULATION

By Lemma 8.12 we have that $L_{\mathcal{OR}}(N) = V_{\mathcal{OR}}(N)$. In this section, we introduce an MILP formulation to obtain $V_{\mathcal{OR}}(N)$, and therefore also $L_{\mathcal{OR}}(N)$. This is done by searching for a non-temporal labeling of networks in which the number of vertical arcs is minimized.

Let N be a given network with vertex set V and arc set A. Let R denote the set of reticulations of N. We define the decision variable l_v to be the non-temporal label of the vertex $v \in V$. A tree arc and a vertical linking arc uv have the property that $l_u < l_v$. We define x_a to be one if arc $a \in A$ is vertical and zero otherwise. We define h_v to be one if $v \in R$ is a reticulation with only incoming vertical arcs and zero otherwise. Let $v \in V$ be a vertex of N. In what follows, let $P_v \subset V$ be the set of parent nodes of v, $C_v \subset V$ the set of children nodes of v, and X the set of leaves. Let ρ be the root of N.

Then, the MILP formulation is as follows:

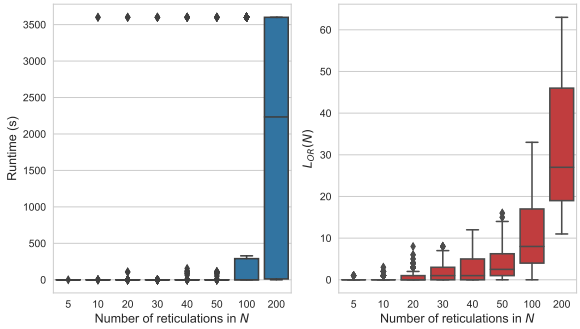
$$\begin{aligned}
\min_{x,h,l} \quad & \sum_{v\in R} h_v & & \\
\text{s.t.} \quad & \sum_{u\in P_v} x_{uv} - 1 \le h_v & \forall v \in R & (8.1)\\
& \sum_{v\in C_u} x_{uv} \ge 1 & \forall u \in V \setminus X & (8.2)\\
& \sum_{u\in P_v} x_{uv} \ge 1 & \forall v \in V \setminus \{\rho\} & (8.3)\\
& l_u \le l_v & \forall uv \in A & (8.4)\\
& l_u \le l_v - 1 & \forall v \in V \setminus R, \forall u \in P_v & (8.5)\\
& l_u \le l_v - 1 + |V|(1 - x_{uv}) & \forall v \in R, \forall u \in P_v & (8.6)\\
& l_u \ge l_v - |V| x_{uv} & \forall v \in R, \forall u \in P_v & (8.7)\\
& x_a \in \{0,1\} & \forall a \in A & \\
& h_v \in \{0,1\} & \forall v \in R & \\
& l_v \in \mathbb{R}_+ & \forall v \in V &
\end{aligned}$$

With constraint (8.1), h_v becomes one if all incoming arcs of reticulation v are vertical. With (8.2) we have that all vertices must have at least one outgoing vertical arc. Then, (8.3) guarantees that each reticulation has at least one incoming vertical arc. Constraint (8.4) creates the non-temporal labeling in the network, where with (8.5) the label of u is strictly smaller than that of v if v is not a reticulation. Then, (8.6) sets x_{uv} to one if uv is vertical, for all reticulation vertices v. Finally, with (8.7) the labels of u and v become equal if x_{uv} is zero.

8.6.3. EXPERIMENTAL RESULTS

In this section, we apply the MILP described in the previous section to a set of real binary networks and to simulated networks, in order to assess the practical running time. The code for these experiments is written in Python and is available at `https://github.com/estherjulien/OrchardProximity`. All experiments ran on an Intel Core i7 CPU @ 1.8 GHz with 16 GB RAM. For solving the MILP problems, we use the open-source solver SCIP [34].

The real data set consists of different binary networks found in a number of papers, collected on `http://phylnet.univ-mlv.fr/recophync/networkDraw.php`. These networks have a leaf set of size up to 39 and a number of reticulations up to 9, with one outlier that has 32 reticulations. All the binary instances completed within one second (at most 0.072 seconds). Based on the results, we observe that only two out of the 22 binary networks have a value of $L_{\mathcal{OR}}(N) > 0$, thus, that only two are non-orchard. The most interesting of these is the network from Szöllősi et al. [192] since its reticulations represent HGT highways. Even though each highway represents many gene transfers, it is still natural to expect these highways to be horizontal. However, our experimental results show that this network is not orchard (see section 8.A for mathematical arguments) and that its $L_{\mathcal{OR}}$ distance is 1. The most interesting part of this network is redrawn in



(a)

R	Compl. (%)
5	100.0
10	99.2
20	98.8
30	98.0
40	97.2
50	97.6
100	75.6
200	61.6

(b)

Figure 8.7: Results of simulated networks. (a) Box plots for the runtime results (blue plot) and the $L_{\mathcal{OR}}(N)$ solutions (red plot) per number of reticulations of simulated networks N. The box plots are drawn with respect to the median of the runtime and the leaf addition score. (b) A table with the percentage of instances that were solved within the one-hour time limit. In the plots of (a), we also included instances that did not complete within the one-hour time limit. For these instances, we set their runtime to one hour.

Figure 8.1, where we also indicate a way to draw it as a tree with horizontal linking arcs, after adding a single leaf. This added leaf represents a hypothesized missing taxon. In general, the $L_{\mathcal{OR}}(N)$ value gives a lower bound on the number of missing taxa that needs to be added to a network to make it HGT-consistent.

The simulated data is generated using the birth-hybridization network generator of Zhang et al. [228], which can generate all binary network topologies [108]. Hence, even though it uses a model with hybridization to construct networks, it can also generate, for example, networks where reticulations represent HGT. This generator has two user-defined parameters: λ, which regulates the speciation rate, and ν, which regulates the hybridization rate. Following Janssen and Liu [108] we set $\lambda = 1$ and we sampled $\nu \in [0.0001, 0.4]$ uniformly at random. We generated an instance group of size 50 for each pair of values (L, R), with the number of leaves $L \in \{20, 50, 100, 150, 200\}$ and the number of reticulations $R \in \{5, 10, 20, 30, 40, 50, 100, 200\}$. In our implementation, we only defined variables x_a for incoming reticulation arcs. Therefore, the number of binary variables only depends on the number of reticulations in the network. In Figure 8.7a, the runtime and $L_{\mathcal{OR}}(N)$ value results for the simulated instances are shown against the reticulation number of the networks. The time limit was set to one hour. We can observe from these results that for networks with up to 50 reticulations, almost all instances are solved to optimality within a second. Then for $R = 100, 200$ the runtime increases, mainly because only 75.6% and 61.6% of the instances could be solved within the time limit, respectively (see Table 8.7b). The completed instances are often still solved within reasonable time.

8.7. DISCUSSION

In this chapter we investigated the minimum number of leaf additions needed to make a network orchard, as a way to measure the extent to which a network deviates from being orchard. We showed that computing this measure is NP-hard (Theorem 8.6), and give a sharp upper bound by the number of reticulations minus one (Theorem 8.7). The measure was reformulated to one in terms of minimizing the number of inrets over all possible non-temporal labelings. In Section 8.6 we use this reformulation to model the problem of computing the leaf addition measure as an MILP. Experimental results show that real-world data instances were solved within a second and the formulation worked well also over synthetic instances, being able to solve almost all instances up to 50 reticulations and 200 leaves within one second. For bigger instances the runtime however increased.

In this chapter we have simulated networks using the network generator of Zhang et al. [228] in order to analyse the running time of our MILP. Alternatively, one could simulate networks by generating orchard networks and deleting leaves from them. Since the leaf addition score is finite for any network by Theorem 8.7, it is possible to obtain any network by using this method. The leaf addition score gives a lower bound on the number of leaves that must be added to make the network orchard. The actual number of missing leaves could be larger, but this value cannot be estimated from the leaf-deleted network.

Of interest is how these results can potentially be used in practice. As mentioned in Section 8.1, one can consider a scenario in which it is suspected *a priori* that species under consideration evolve under a network in which all reticulate events are horizontal. An example of such scenarios can be seen for horizontal gene transfers, for instance when one considers the evolutionary history of species in bacteria [94] and fungi [192]. If a produced network does not admit an HGT-consistent labeling, there can in general be many reasons. For one, the output network may not be accurate. It is also possible that certain species have gone extinct, or that undersampling is present in the taxon set. In these latter two potential causes, our method gives a way of quantifying the minimum number of taxa that may have gone extinct / been undersampled. Moreover, it can be used to find all optimal corresponding orchard networks with added leaves. This could, for example, be used to try to identify the missing taxa.

Our NP-hardness result is interesting when comparing it to the computational complexity of the corresponding problem for different network classes. The problem of finding the minimum number of leaves to add to make a network tree-based can be solved in polynomial time [80] (Lemma 8.6) and we showed that the same is true for the class of tree-child networks (Lemma 8.4). Interestingly, the class of tree-child networks is contained in the class of orchard networks [109] which is in turn contained in the class of tree-based networks [100]. The reason for such an NP-hardness sandwich can perhaps be attributed to the lack of forbidden shapes. Leaf additions to obtain tree-child or tree-based networks target certain forbidden shapes in the network. In the case of tree-child networks, we add a leaf to exactly one outgoing arc of each omnian; for tree-based networks, we add a leaf to exactly one arc of each W-fence. The problem of finding a characterization of orchard networks in terms of (local) forbidden shapes has been elusive thus far [109] - perhaps the NP-hardness result for the orchard variant of the problem

indicates that finding such a characterization for orchard networks may not be possible.

One can also consider the leaf addition problem for non-binary networks. Non-binary networks generalize the networks considered in this chapter by allowing vertices to have varying indegrees and outdegrees. This generalized problem remains NP-complete since the binary version is a specific case. It could be interesting to try to find an MILP formulation for the nonbinary version.

Another natural research direction is to consider different proximity measures. One that may be of particular interest is a proximity measure based on arc deletions. That is, what is the minimum number of reticulate arc deletions needed to make a network orchard? Susanna showed that this measure is incomparable to the leaf addition proximity measure [190], yet it is not known if it is also NP-hard to compute.

APPENDIX OF CHAPTER 8

8.A. REMARK AND OMITTED PROOFS

Remark 8.1. *We first elaborate on why we need an added leaf (unsampled taxon) in the network of Figure 8.1 to ensure that the network admits an HGT-consistent labeling. We know that a network has an HGT-consistent labeling if and only if it is orchard (Theorem 8.2). Let N be the network without unsampled taxon (see Figure 8.8). We will show that N is not orchard. To see this, note that the order in which cherries and reticulated cherries are reduced does not matter [109]. This means that if N were orchard, then there would exist a cherry-picking sequence starting with*

$$(AN, PCW)(PCW, AN),$$

for $AN = AspNidulans$ and $PCW = PenChrysogenumWisconsin$. After reducing these cherries, the distance between the leaf $AspNidulans$ and any other leaf remains of distance at least 4, regardless of other reductions that take place in the network. This shows that the network cannot be orchard, and therefore the network cannot have an HGT-consistent labeling.

▷ **Lemma 8.3.** *Let N be a network. Then $L_{\mathcal{TC}}(N)$ is equal to the number of omnians. Moreover, N can be made tree-child by adding a leaf to exactly one outgoing arc of each omnian.*

Proof. By definition, a network is tree-child if and only if it contains no omnians. We show that every leaf addition can result in a network with one omnian fewer than that of the original network. Let uv be an arc where u is an omnian. Add a leaf x to uv. In the

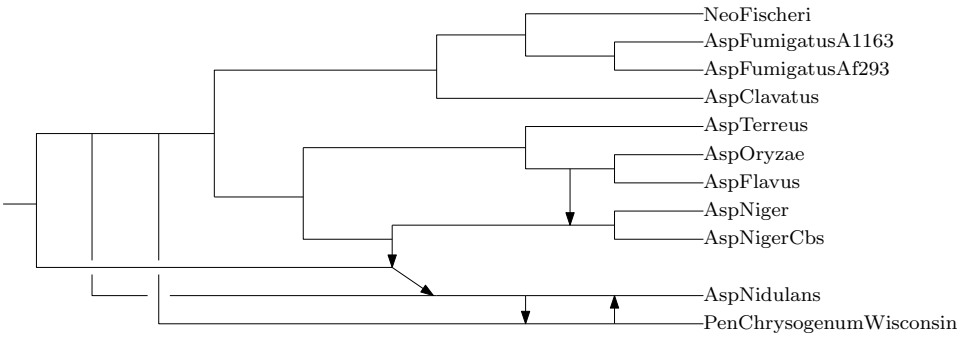


Figure 8.8: The network of Figure 8.1 without the added leaf. Observe that there exists no HGT-consistent labeling for the network, by the arguments provided in Remark 8.1.

resulting network, u has a child (the parent of x) that is a tree vertex, and it is no longer an omnian. The newly added tree vertex has a leaf child x; the parent-child combinations remain unchanged for the rest of the network, so at most one omnian (in this case u) can be removed per leaf addition. It follows that $L_{\mathcal{FC}}(N)$ is at least the number of omnians in N. By targeting arcs with omnian tails, we can remove at least one omnian per every leaf addition, so that $L_{\mathcal{FC}}(N)$ is at most the number of omnians in N. Therefore, $L_{\mathcal{FC}}(N)$ is exactly the number of omnians in N. □

▷ **Lemma 8.4.** *Let N be a network. Then $L_{\mathcal{FC}}(N)$ can be computed in $O(|N|)$ time.*

Proof. We first show that the number of omnians of N can be computed in $O(|N|)$ time, by checking, for each vertex, the indegrees of its children. A vertex is an omnian if and only if all of its children are of indegree-2. Since the degree of every vertex is at most 3, each search within the for loop takes constant time. The for loop iterates over the vertex set which is of size $O(N)$. By Lemma 8.3, since $L_{\mathcal{FC}}(N)$ is the number of omnians in N, we can compute $L_{\mathcal{FC}}(N)$ in $O(|N|)$ time. □

It has been shown already that $L_{\mathcal{FB}}(N)$ can be computed in $O(|N|^{3/2})$ time where $|N|$ is the number of vertices in N [80]. We show that this can in fact be computed in $O(|N|)$ time.

▷ **Lemma 8.5.** *Let N be a network. Then $L_{\mathcal{FB}}(N)$ is equal to the number of W-fences. Moreover, N can be made tree-based by adding a leaf to any arc in each W-fence in N.*

Proof. By Lemma 8.2, a network is tree-based if and only if it contains no W-fences. We show that every leaf addition can result in a network with one W-fence fewer than that of the original network. Suppose that N contains at least one W-fence. Otherwise we may conclude that the network is tree-based by Lemma 8.2. Let $(a_1, a_2, \ldots, a_k)$ be a W-fence in N where $a_i = u_i v_i$ for $i \in [k]$, and add a leaf x to a_1; let p_x be the tree vertex parent of x. In the resulting network, the arcs in $\{u_1 p_x, p_x v_1, p_x x, a_2, a_3, a_4, \ldots, a_k\}$ are decomposed into their unique maximal zig-zag trails (Theorem 8.3) as two N-fences $(u_1 p_x)$ and $(a_k, a_{k-1}, \ldots, a_3, a_2, p_x v_1, p_x x)$. All other arcs remain in the same maximal zig-zag trails as that of N. Therefore the number of W-fences has gone down by exactly one. This can be repeated for all W-fences in the network; it follows that $L_{\mathcal{FB}}(N)$ is the number of W-fences in N.

A quick check shows that adding a leaf to any arc in the W-fence decomposes the W-fence into two N-fences. □

▷ **Lemma 8.6.** *Let N be a network. Then $L_{\mathcal{FB}}(N)$ can be computed in $O(|N|)$ time.*

Proof. Finding the maximal zig-zag decomposition takes $O(|N|)$ time (Proposition 5.1 of Hayamizu [97]). Counting the number of W-fences in the decomposition gives $L_{\mathcal{FB}}(N)$ by Lemma 8.5. □

▷ **Observation 8.1.** *Let G be a 3-regular graph and let N_G be the network obtained by the reduction. Then N_G is tree-based.*

Proof. It is easy to check that the arcs of N_G are decomposed into M-fences and N-fences (the principal part of each gadget $\mathrm{Gad}(v)$ is an N-fence; each arc leaving the principal part of a gadget $\mathrm{Gad}(v)$ is an N-fence of length 1; the remaining arcs decompose into M-fences of length 2). By Lemma 8.2, N_G must be tree-based. □

▷ **Lemma 8.7.** *Let N be a tree-based network, and let $\hat{N}_1, \hat{N}_2, \ldots, \hat{N}_n$ denote the N-fences of N of length at least 3. Then every cherry cover of N contains the reticulated cherry shapes $\{(\mathrm{head}(a^i_{2j-1})c^i_{2j-1}), a^i_{2j}, a^i_{2j+1}\}$ for $i \in [n]$ and $j \in \left[\frac{k_i-1}{2}\right]$.*

Proof. Let $\hat{N}_i = (a^i_1, a^i_2, \ldots, a^i_{k_i})$ be an N-fence of length $k_i \geq 3$. Observe that in every cherry cover, exactly one incoming arc of every reticulation is covered by a reticulated cherry shape as a middle arc (since the network is binary; for non-binary networks, this is not true in general [202]). Since $\mathrm{head}(a^i_1)$ is a reticulation, one of a^i_1 or a^i_2 must be in a reticulated cherry shape as a middle arc. But $\mathrm{tail}(a^i_1)$ is a reticulation; therefore, a^i_2 must be in a middle arc of a reticulated cherry shape. The other two arcs of the same reticulated cherry shapes are then fixed to be $\mathrm{head}(a^i_1)c^i_1$ and a^i_3. Repeating this argument for the reticulations $\mathrm{head}(a^i_{2j+1})$ for $j \in \left[\frac{k_i-1}{2}\right]$ gives the required claim for the N-fence $\hat{N}_i$; further repeating this argument for every N-fence gives the required claim. □

▷ **Lemma 8.8.** *Let N be a tree-based network and suppose that for two N-fences $\hat{N}_u := (a^u_1, a^u_2, \ldots, a^u_{k_u})$ and $\hat{N}_v := (a^v_1, a^v_2, \ldots, a^v_{k_v})$ of length at least 3, there exist directed paths in N from $\mathrm{head}(a^u_h)$ to $\mathrm{tail}(a^v_i)$ and from $\mathrm{head}(a^v_j)$ to $\mathrm{tail}(a^u_k)$, for even h, i, j, k with $k < h$ and $i < j$. Then every cherry cover auxiliary graph of N contains a cycle.*

Proof. Let us again denote the reticulated cherry shape that contains a^u_h and a^u_{h+1} by $R^u_{h/2}$, and similarly for $R^v_{i/2}$, $R^v_{j/2}$,and $R^u_{k/2}$. By Lemma 8.7, all of $R^u_{h/2}$, $R^v_{i/2}$, $R^v_{j/2}$, $R^u_{k/2}$ appear in the cherry cover auxiliary graph. Moreover $R^u_{k/2}$ is above $R^u_{h/2}$, and $R^v_{i/2}$ is above $R^v_{j/2}$. Now observe that for any consecutive arcs on the path from $\mathrm{head}(a^u_h)$ to $\mathrm{tail}(a^v_i)$, either they are part of the same reticulated cherry shape in the cherry cover, or they are part of different cherry shapes with one cherry shape directly above the other. This implies that there is a path from $R^u_{h/2}$ to $R^v_{i/2}$ in the cherry cover auxiliary graph. A similar argument shows that there is a path from $R^v_{j/2}$ to $R^u_{k/2}$. But then we have that $R^u_{h/2}$ is above $R^v_{i/2}$, which is above $R^v_{j/2}$, which is above $R^u_{k/2}$, which is above $R^u_{h/2}$ and we have a cycle. □

▷ **Lemma 8.9.** *Let G be a 3-regular graph and let N_G be the network obtained by the reduction above. Suppose that A is a set of arcs of N_G, for which adding leaves to every arc in A results in an orchard network. For every edge $uv \in E(G)$, there exists an arc $a \in A$ that is an arc of the principal part of $\mathrm{Gad}(u)$ or $\mathrm{Gad}(v)$.*

Proof. We prove this lemma by contraposition. Let us assume that there is an edge $uv \in E(G)$, such that no arcs of the principal part of $\mathrm{Gad}(u)$ or $\mathrm{Gad}(v)$ are in A. We shall show that the network obtained by adding leaves to all $a \in A$ in N_G – which we denote $N_G + A$ – is not orchard. From Theorem 8.1 we know that N_G is orchard if and only if N_G has an acyclic cherry cover. We show here that $N_G + A$ will not have an acyclic cherry cover, thereby showing that $N_G + A$ is not orchard. As no arcs were added to the principal

part of Gad(u) or Gad(v), these principal parts remain N-fences in $N_G + A$. Furthermore by construction N has an arc from some head(a_h^v) to tail(a_i^u) for even $h \geq 10$ and even $i \leq 6$, and so $N_G + A$ has a path from head(a_h^v) to tail(a_i^u). Similarly $N_G + A$ has a path from head(a_j^u) to tail(a_k^v) for some even $j \geq 10$ and $h \leq 6$. Then Lemma 8.8 implies that the auxiliary graph of any cherry cover of $N_G + A$ contains a cycle. By Theorem 8.1, we have that $N_G + A$ is not orchard. □

▷ **Lemma 8.10.** *Let G be a 3-regular graph and let N_G be the network obtained by the reduction described above. Then G has a minimum vertex cover of size k if and only if $L_{\mathcal{OR}}(N_G) = k$.*

Proof. Suppose first that V_{sol} is a vertex cover of G with at most k vertices. We shall show that adding a leaf to an arc of the principal part of each Gad(v) for $v \in V_{sol}$ makes N_G orchard. This will show that the minimum vertex cover of G is at least $L_{\mathcal{OR}}(N_G)$. In the remainder of this proof, we will refer to vertices and arcs of N_G as introduced above in the reduction.

For every $v \in V_{sol}$, we add a leaf z^v to the arc $w_4^v r_4^v$ of Gad(v) (see Figure 8.5c). Let q^v be the parent of z^v. The key idea is that this splits the principal part of Gad(v) from an N-fence into an N-fence and an M-fence, and this allows us to avoid the cycle in the cherry cover auxiliary graph (see Figure 8.5d). Let us call the new network M. To formally show that M is orchard, we give an HGT-consistent labeling $t : V(M) \to \mathbb{R}$. Begin by setting $t(\rho) = 0$, and for any vertex in $s_1, \ldots, s_{g-1}$ or $\rho^v, m_4^v, \ldots, m_2^v$ for any v in $V(G)$, let this vertex have label equal to the label of its parent plus 1. Let h be the maximum value assigned to a vertex so far, and now adjust t by subtracting $(h+1)$ from each label. Thus, we may now assume that all vertices in $\rho, s_1, \ldots, s_{g-1}$ or $m_4^v, \ldots, m_2^v$ for any v in $V(G)$ have label ≤ -1. Now set $t(m_1^v) = 0$ and $t(r_0^v) = 0$, for each v in $V(G)$. It is easy to see that so far t satisfies the properties of an HGT-consistent labeling. It remains to label the vertices in the principal part of each gadget Gad(v), and the leaves of each gadget, and the new vertices q^v and z^v for $v \in V_{sol}$. We do this as follows. For $v \in V_{sol}$, set $t(r_1^v) = t(w_1^v) = 12, t(r_2^v) = t(w_2^v) = 13, t(r_3^v) = t(w_3^v) = 14$, and $t(r_4^v) = t(q^v) = 15$. Set $t(w_4^v) = 1, t(r_5^v) = t(w_5^v) = 2, t(r_6^v) = t(w_6^v) = 3$, and $t(r_7^v) = t(w_7^v) = 4$. For $v \notin V_{sol}$, set $t(r_1^v) = t(w_1^v) = 5$, and $t(r_i^v) = t(w_i^v) = i + 4$ for every i up to $t(r_7^v) = t(w_7^i) = 11$.

Finally, for each leaf ℓ with parent p set $t(\ell) = t(p) + 1$. It remains to observe that t is a non-temporal labeling of M and for every reticulation r in M, r has exactly one parent p with $t(p) = t(r)$. Thus t is an HGT-consistent labeling of M, and it follows from theorem 8.2 that M is orchard.

Suppose now that we have a set of arcs A_{sol} of N_G of size at most k, such that adding leaves to the arcs in A_{sol} makes N_G orchard. By Lemma 8.9, for every edge $uv \in E(G)$, there exists an arc $a \in A_{sol}$ that is an arc of the principal part of Gad(u) or Gad(v). It follows immediately that the set $\{v \in V(G) : A_{sol}$ contains an arc of the principal part of Gad(v)$\}$ is a vertex cover of G. Since this is true for any such set of arcs A_{sol}, it follows that if there is such an A_{sol} of size at most k, then there must exist a vertex cover of G of size at most k. □

9

CONCLUSION & DISCUSSION

With the rise of machine learning (ML) in recent years, many new types of algorithmic operations have become available. This thesis explores the applicability of ML in optimization by designing data-driven algorithms to improve the overall performance of finding solutions. While the problems studied – bilevel optimization, two-stage robust optimization, and the reconstruction of phylogenetic networks – may seem diverse at first glance, they share a common trait: existing algorithms for solving these problems require overly excessive time. Finding an optimal solution for these problems can easily take a couple of hours for small instances. Moreover, while applying ML to single-level problems and inferring phylogenetic trees have already been studied, the problems explored in this thesis were mostly untouched. The proposed data-driven algorithms leverage neural network (NN) embedding and integrate other ML models to guide algorithmic steps. At the same time, we aim to mitigate the effects of the added complexity caused by ML. One approach to achieve this is by ensuring that the training data generation process avoids relying on finding optimal solutions for many problem instances, and instead requires simpler methods.

FUTURE DIRECTIONS: IN HIERARCHICAL OPTIMIZATION

In Chapter 3, the method NEUR2BILO for solving bilevel optimization problems was introduced, and in Chapter 4 the method NEUR2RO for solving mixed-integer two-stage robust optimization was proposed. The latter algorithm does not make any assumptions on the linearity of the problem, and can therefore easily be extended to non-linear problems as well. However, this was not experimentally studied. Another way both methods could be generalized is by extending the methodology, such that it is independent of the number of levels or stages. Hence, interesting further work could be to design data-driven algorithms for multi-level optimization and multi-stage robust optimization. The first main open question is:

Open Problem 1
Can NEUR2BILO and NEUR2RO respectively be generalized to multi-level problems and to multi-stage robust problems with more than two levels or stages?

The next direction pertains to mathematical programming more generally. In our research, we aimed at solving huge problems. Especially for bilevel optimization this should be achievable, albeit with some alterations, as the algorithm we propose requires solving a single-level problem. We mostly refrained from experimenting on very large instances as we needed to test our method against baseline methods, and these could not solve such problems within reasonable time. Moreover, the NN embedding requires that each variable of the problem is contained in the NN's input. For huge problems with 1,000s of variables, this would easily create all sorts of problems, related to both training the NN, embedding the NN, and solving the resulting MIP. During this research phase, ideas came to mind of what are suitable embeddings for the task of solving huge problems. It might be possible to design an encoding within the NN architecture that eliminates the need to include all variables as inputs, allowing a fixed-dimensional vector to suffice. Additional preprocessing steps may be necessary, as long as the entire procedure ensures the original problem remains well-represented. This leads to the next open problem:

Open Problem 2
Is there an encoding method for huge optimization problems, via NN embeddings, that shrinks the size of the MIP formulation by eliminating some variables?

FUTURE DIRECTIONS: IN PHYLOGENETICS

In this thesis, a class of heuristics for reconstructing phylogenetic networks from a set of multifurcating trees on overlapping taxa is introduced. It uses a combination of ML and cherry picking to first reduce all trees in the tree set and subsequently construct a network via the resulting CPS. The ML method decides on which cherry to pick in each iteration. We have not developed a *robust* algorithm that controls for wrong moves. In fact, we do not know what wrong moves would entail in the solving procedure. A wrong move in this setting is whenever a *wrong cherry* is picked. A cherry is wrong if picking it prevents the heuristic from obtaining an optimal network. It is important to be aware of the possible implications of picking a wrong cherry, for both determining approximation results and making the heuristic more robust. We feel that picking 'wrong' cherries in earlier stages of the algorithm can create more complications later in the procedure. Selecting the wrong cherries could trigger a chain reaction, affecting the remaining cherries in the trees and making it more probable that additional wrong ones will be picked. This results in the following possible future direction:

Open Problem 3
Theoretically analyze the accuracy and robustness of the cherry-picking heuristic.

Resolving the next open problem would be a great contribution to the field and would improve the cherry-picking heuristics (CPHs) presented in Chapter 6 and Chapter 7. There is a clear characterization for optimal tree-child networks on a given tree set, based on cherry-picking sequences (CPSs) [134]. Namely, a minimum-length tree-child CPS (i.e., a CPS with extra restrictions) that reduces a tree set $\mathcal{T}$ can reconstruct an optimal tree-child network. Such a characterization does not yet exist for orchard networks, which are evidently defined as networks that can be reduced by a (regular) CPS.

We have that a CPS that reduces $\mathcal{T}$ reconstructs a feasible network, but we do not know whether a minimum-sized sequence necessarily reconstructs an optimal orchard network. Since cherry picking is by definition such a big component of orchard networks, as future work, it would be interesting to identify a possibly similar characterization as we have for tree-child networks. This idea results in the following open problem:

Open Problem 4
Is there a characterization of optimal orchard networks for a set of trees based on cherry-picking sequences, similar to the characterization for tree-child?

If such a characterization exists, we can possibly improve the CPHs in various ways. The heuristic itself is quite flexible; extra procedures can be incorporated, as was also done for the tree expansion step. Thus, if next to cherry picking, additional steps are required to find an optimal orchard network, these can possibly be included in the CPH. The data generation procedure for CPH consists of creating a synthetic orchard network to be used as an output of the HYBRIDIZATION problem, and taking all (or a sample of) displayed trees as input of the problem. In this way, labeling which cherries are (probably) good to pick can be done quickly. For tree-child CPSs and networks, one can always fully reduce both the network and sampled display trees. As is noticed by Janssen and Murakami [109, Figure 13] for orchard networks, after simultaneous cherry-picking reductions in the network and the tree set of correct cherries, it could be the case that the partly reduced network no longer represents all the partly reduced trees of the set. If a solid characterization for orchard networks based on CPSs exists, we could change the data generation scheme with possibly additional steps such that we can always reduce N and $\mathcal{T}$ fully, and consequently gather better data.

Another interesting direction, that would benefit from some of the techniques proposed in this thesis, is solving the following important problem. Linz and Semple [134] introduce cherry picking for the reconstruction of tree-child networks. Another characterization is also introduced. Cherry picking can in theory be used to construct an *overall* optimal network, instead of an optimal network within a restrictive class, by attaching auxiliary leaves to the trees. The rough idea is as follows, let $\mathcal{T}$ be the initial tree set on X, and let $\mathcal{T}'$ be a tree set on $X \cup Z$, where Z are the auxiliary leaves; and all leaves of Z are attached to each tree. Based on this new tree set $\mathcal{T}'$, a tree-child network N'_{tc} is generated via cherry picking, and the auxiliary leaves are deleted from N'_{tc} to create N, which is the network that displays the initial tree set $\mathcal{T}$. The authors prove there exists a set Z, and a way to attach the leaves from Z to $\mathcal{T}$, such that the previously explained procedure results in obtaining N from $\mathcal{T}$ and that this is an optimal network for $\mathcal{T}$. An algorithm for assembling Z and $\mathcal{T}'$ is not provided. However, if N is known, the procedure is quite straightforward. It uses simple operations to eliminate the forbidden substructures of tree-child networks: stacks (i.e., the tail and head of an edge are both reticulations) and tree nodes with only reticulations as its children. Namely,

1. Eliminate the forbidden structures of tree-child networks by contracting edges (for stacks) and adding leaves (for reticulate offspring). These leaves are attached to N such that no new reticulations are created, giving a tree-child network N'_{tc}.

2. Create $\mathcal{T}'$ by attaching the leaves of Z to each tree $T \in \mathcal{T}$ such that $\mathcal{T}'$ is displayed by N'_{tc}.

Perhaps ML could be useful here, with the "expert" data being similarly generated as in Chapter 6. First, a synthetic non-tree-child network is generated and its displayed trees are sampled. Subsequently, new leaves are added to the network (to eliminate forbidden substructures) and additionally to feasible positions in the trees. In this way, the leaves and their locations in $\mathcal{T}'$ are known. The learning task is non-obvious, and leads to the following open problem:

Open Problem 5
Given a tree set $\mathcal{T}$ on X, can ML be used to find a set Z of auxiliary leaves and simultaneously add these leaves to the right positions in $\mathcal{T}$ to reconstruct an optimal network N using cherry picking?

As a final note on this research direction, this methodology can be extended to orchard networks. In that case, we do not assume that a tree-child network is generated via cherry picking (which uses more restrictions), but an orchard network. Even though there does not yet exist a characterization of optimal orchard networks based on CPSs (Open Problem 4), we have shown with the introduced CPH that obtaining good networks can be done within reasonable time. Perhaps we can then apply attaching auxiliary leaves to these obtained orchard networks. However, as we both want to find an auxiliary leaf set, and aim to attach them to the tree set in the best possible way, finding a smallest possible leaf set Z could be favorable. At the training data generation stage where we find "expert" labels, for tree-child networks, finding a smallest leaf set can be done in polynomial time. For orchard networks, as is shown in Chapter 8, finding a smallest set is NP-hard. The upper bound, of $r(N)-1$, can be derived in polynomial time, with $r(N)$ the reticulation number of N. This is possibly a much higher value than the one found for tree-child networks.

In recent years, graph neural networks (GNNs) [221] have been utilized in optimization in various ways [114]. Its graph-like architecture can be especially useful for combinatorial optimization as many problems share this structure by construction. Applying GNNs to phylogenetics is a natural extension, where it can aid in constructing phylogenetic networks. Recently, in a Master's thesis, Broeders [43] has developed a learning-based heuristic to construct 'temporal' networks, using cherry picking [101] and a GNN. Next to that, in Dushatskiy et al. [68] a learning-based operation is proposed in which tree containment in a network is guided by a GNN. The results are promising, and bring forth other ideas of utilizing GNNs. For example, perhaps a GNN can be used for direct phylogenetic network construction from a set of gene trees or directly from sequence data. The GNN can take the shape of a network with an overestimation of possible reticulations and arcs. The GNN can then predict which reticulation nodes and arcs should be present in the phylogenetic network representing the data. This results in the last open problem:

Open Problem 6
Can graph neural networks be utilized to learn the whole phylogenetic network at once?

Vision for ML for optimization

In this thesis, the focus is on data-driven algorithms to solve optimization problems more effectively and efficiently. In Section 1.1 we also discussed other ways in which ML can play a role in optimization. Namely, by predicting unknown parts of the problem description or its parameters. In this section, we discuss another form that is even more general than the ones discussed: predict the formulation of the entire optimization model. Even though many would argue this is very naive, research in this field would not only be important for people who are already experts, but more importantly, for people who are not. We view this research as being critical for bringing the techniques of mathematical optimization to the public and thus making this field accessible.

Many decision problems that people (in companies) have to deal with in a day-to-day setting, can be solved with operations research (OR) methods; e.g., to solve logistics or other planning problems. Modeling a given problem as a mathematical program is a very hard task for non-experts and therefore less attainable and often overseen by smaller companies or organizations that do not have the capacity to hire an expensive consultancy team. On the contrary, data science and ML are much easier for non-experts to apply, in part because of the many open-source packages that have come out. But also importantly, because it is not hard to program a (very) simple machine-learning model; one only needs an input and a label. Whether this is sufficient is part of another debate, but we could agree that due to its accessibility, we observe a growing trend where these methods are being used much more frequently than mathematical programming techniques over all domains.

To make OR as easily accessible as ML, providing open-source software has not been enough, as the difficulty in modeling itself already requires experts. A translation is required: from the decision problem to the mathematical program, without the use of expert minds. In recent years, large language models (LLMs) have been increasingly present at, and almost overtaking, ML conferences. And ChatGPT has been dominating dinner conversations in many households. In large part because it is extremely easy to use for the public. An automatic translation from text to model would be ideal, where LLMs perform this step. Recently, papers have been presented that move towards design automation of heuristics [137, 223]. Anyone who has ever worked with OR for real-world applications, or with data science, knows that the initial model rarely works adequately. Here data science experts and OR specialists can be extremely useful. But small companies most likely deal with smaller problems, for which less complicated models suffice. Even though the model might not be perfect, it will become easier for the users to get a grasp of the existence of mathematical programming and the benefits one can obtain by using it.

ACKNOWLEDGEMENTS

These past four years have known many different phases. From starting my PhD position in the middle of the pandemic where I mainly connected to new faces online, to events that made me familiarize myself with the ins and outs of hospitals a lot more than I thought I would ever be. But apart from these events, the previous years have taught me so much and brought me a lot of joy. I have fallen in love with research, became obsessed with it at times, became obsessed with other things, was able to visit many new places during conferences and a research visit, and most importantly, met a lot of wonderful people.

First of all, I would like to thank my supervisors Leo and Leen for their incredible support throughout these years. For both personal and work-related issues you were always there for me. I vividly recall when Leo reminded me in my last year, during a particularly overwhelming period, that stress is not caused by working too much, but by thinking too much. Your empathy and understanding perfectly describe you as a supervisor and person, for which I want to thank you. Both of you have always been very considerate during the personal hardships of these past years, but also able to (very moderately) push me when you knew I was mainly lying to myself. Due to the freedom you have given me in initiating my own projects and working with others, I have been able to meet and closely work with many amazing people. Giulia, thank you for your incredible kindness when we worked together on the cherry-picking papers and for being a mentor to me. Krzysztof and Ilker, thank you for inspiring me to continue my work in K-adaptability after my master's by making it fun to work with you and always helping me swiftly and enthusiastically whenever I had a question. Also thanks to Yuki and Mark, from whom I learned so much about phylogenetics during these past years. I fondly look back at the spirited whiteboard sessions we had with Leo. These meetings showed me a different way of collaboration and research, one I thoroughly enjoyed. During my last years, I have mainly worked on the "Neur2" series. Elias and Jannis, thank you for putting faith in me to collaborate with you on multiple subsequent papers, after only meeting me once during a nice conference in Nice. Together with Justin, you have taught me a tremendous amount about ML for optimization, two-stage robust optimization, bilevel optimization, coffee, and fun spots in Toronto, Montreal, and Vancouver. Thank you for your kindness, enthusiasm, the immense determination you all showed during the around-the-clock working sessions to meet conference deadlines, and for facilitating my visit to Toronto.

I also want to thank all of my colleagues in the Optimization group for the welcoming atmosphere you all created, the fun outings, and lively lunch conversations. In particular, Lara, thank you for being there throughout these four years, as a lovely office mate, conference buddy, and ML for optimization oracle. Ananth, even though you only joined last year, you have become very important to me during my PhD journey and beyond. Thank you for the many inspiring conversations we shared and will continue to share. To

my office mates, Maaike, Lara, Merel, Willem, Bram, Tom, and Cindy: thank you for the fun moments and for helping me whenever I was in need. Perhaps more importantly, sorry for not letting you continue with your work with all my yapping. And to all the many new friends I made at TU Delft while raising our voices in unison: you are some of my biggest inspirations. Thank you for your determination, incredible kindness, empathy, and hope. And perhaps most admirably, for showing me how to laugh while dealing with unimaginable personal circumstances.

Outside of my work/campus life, I was immensely supported by my dear friends in and outside of Delft. Claire and Tim, thank you for being kind, moving back, appreciating my quirks, and being so present and important in my daily life. Guido and Shen, thank you for supporting me throughout the last couple of years and for all the fun moments we share. To Maud, Rens, Gianmarco, and Olivia, even though we do not live that close anymore, seeing each other again always feels like never having been apart. To my dear family, especially my parents, Rick, and Núria, thank you for your love and support throughout these years. I appreciate your willingness to understand what I am actually doing, your presence during special moments, and for being proud of me. Finally, to Ben, thank you for everything. Thank you for being my biggest support, encouraging me in whatever decision I make, and being incredibly loving and kind. You make me so happy, and I am looking forward to the next phase of our life.

Curriculum Vitæ

Esther Anna Theresia Julien

28-09-1996 Born in Geldrop, The Netherlands.

Education

2015–2019 BSc. Econometrics & Operations Research
Tilburg University, The Netherlands

2019–2021 MSc. Econometrics & Management Science
Erasmus University Rotterdam, The Netherlands

2021–2025 PhD. Applied Mathematics
Delft University of Technology, The Netherlands
Thesis: Leveraging Data in Algorithm Design: for Problems in Bilevel Optimization, Adaptable Robust Optimization, and Phylogenetics
Promotor: Dr.ir. L.J.J. van Iersel
Promotor: Prof.dr. L. Stougie

List of Publications

Journal Papers

1. **Esther Julien**, Krzysztof Postek, and Ş İlker Birbil. "Machine learning for K-adaptability in two-stage robust optimization". In: *INFORMS Journal on Computing*. 2024.

2. Giulia Bernardini, Leo van Iersel, **Esther Julien**, and Leen Stougie. "Inferring phylogenetic networks from multifurcating trees via cherry picking and machine learning". In: *Molecular Phylogenetics and Evolution 199*, 2024.

3. Arkadiy Dushatskiy, **Esther Julien**, Leen Stougie, and Leo van Iersel. "Solving the tree containment problem using graph neural networks". In: *Transactions on Machine Learning Research*. 2024.

4. Giulia Bernardini, Leo van Iersel, **Esther Julien**, and Leen Stougie. "Constructing phylogenetic networks via cherry picking and machine learning". In: *Algorithms for Molecular Biology 18.1*. 2023.

Conference Papers

5. Justin Dumouchelle, **Esther Julien**, Jannis Kurtz, and Elias B Khalil. "Neur2BiLO: Neural bilevel optimization". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*. 2024.

6. Justin Dumouchelle, **Esther Julien**, Jannis Kurtz, and Elias B Khalil. "Neur2RO: Neural two-stage robust optimization". In: *The Twelfth International Conference on Learning Representations (ICLR)*. 2024.

7. Leo van Iersel, Mark Jones, **Esther Julien**, and Yukihiro Murakami. "Making a network orchard by adding leaves". In: *23rd International Workshop on Algorithms in Bioinformatics (WABI)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2023.

8. Giulia Bernardini, Leo van Iersel, **Esther Julien**, and Leen Stougie. "Reconstructing phylogenetic networks via cherry picking and machine learning". In: *22nd International Workshop on Algorithms in Bioinformatics (WABI)*. Vol. 242. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2022.

Preprints

9. Justin Dumouchelle, **Esther Julien**, Jannis Kurtz, and Elias B Khalil. "Deep learning for two-stage robust integer optimization" In: *arXiv preprint arXiv:2310.04345* (2024)

BIBLIOGRAPHY

[1] Shiran Abadi, Oren Avram, Saharon Rosset, Tal Pupko, and Itay Mayrose. "ModelTeller: model selection for optimal phylogenetic reconstruction using machine learning". In: *Molecular Biology and Evolution* 37.11 (2020), pp. 3338–3352.

[2] Tobias Achterberg and Roland Wunderling. "Mixed integer programming: Analyzing 12 years of progress". In: *Facets of combinatorial optimization: Festschrift for martin grötschel*. Springer, 2013, pp. 449–481.

[3] Benjamin Albrecht. "Computing all hybridization networks for multiple binary phylogenetic input trees". In: *BMC Bioinformatics* 16.1 (2015), pp. 1–15.

[4] Benjamin Albrecht, Céline Scornavacca, Alberto Cenci, and Daniel H. Huson. "Fast computation of minimum hybridization networks". In: *Bioinformatics* 28.2 (2012), pp. 191–197. DOI: 10.1093/bioinformatics/btr618.

[5] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. "Strong mixed-integer programming formulations for trained neural networks". In: *Mathematical Programming* (2020), pp. 1–37.

[6] Ayşe N Arslan and Boris Detienne. "Decomposition-based approaches for a class of two-stage robust binary optimization problems". In: *INFORMS Journal on Computing* 34.2 (2022), pp. 857–871.

[7] Ayşe N Arslan, Michael Poss, and Marco Silva. "Min-sup-min robust combinatorial optimization with few recourse solutions". In: *INFORMS Journal on Computing* 34.4 (2022), pp. 2212–2228.

[8] David Avis, David Bremner, Hans Raj Tiwary, and Osamu Watanabe. "Polynomial size linear programs for problems in P". In: *Discrete Applied Mathematics* 265 (2019), pp. 22–39.

[9] Dana Azouri, Shiran Abadi, Yishay Mansour, Itay Mayrose, and Tal Pupko. "Harnessing machine learning to guide phylogenetic-tree search algorithms". In: *Nature Communications* 12.1 (2021), pp. 1–9.

[10] Dana Azouri, Oz Granit, Michael Alburquerque, Yishay Mansour, Tal Pupko, and Itay Mayrose. "The tree reconstruction game: phylogenetic reconstruction using reinforcement learning". In: *Molecular Biology and Evolution* 41.6 (2024).

[11] Saeed Asadi Bagloee, Mohsen Asadi, Majid Sarvi, and Michael Patriksson. "A hybrid machine-learning and optimization method to solve bi-level problems". In: *Expert Systems with Applications* 95 (2018), pp. 142–152.

[12] Maria-Florina Balcan. "Data-driven algorithm design". In: *arXiv preprint arXiv:2011.07177* (2020).

[13] Eric Bapteste, Leo van Iersel, Axel Janke, Scot Kelchner, Steven Kelk, James O McInerney, David A Morrison, Luay Nakhleh, Mike Steel, Leen Stougie, and James Whitfield. "Networks: expanding evolutionary thinking". In: *Trends in Genetics* 29.8 (2013), pp. 439–441.

[14] Jonathan F Bard. *Practical bilevel optimization: algorithms and applications*. Vol. 30. Springer Science & Business Media, 2013.

[15] Mihaela Baroni, Stefan Grünewald, Vincent Moulton, and Charles Semple. "Bounding the number of hybridisation events for a consistent evolutionary history". In: *Journal of Mathematical Biology* 51.2 (2005), pp. 171–182.

[16] Mihaela Baroni, Charles Semple, and Mike Steel. "A framework for representing reticulate evolution". In: *Annals of Combinatorics* 8 (2005), pp. 391–408.

[17] Mihaela Baroni, Charles Semple, and Mike Steel. "Hybrids in real time". In: *Systematic Biology* 55.1 (2006), pp. 46–56.

[18] Yasmine Beck, Ivana Ljubić, and Martin Schmidt. "A survey on bilevel optimization under uncertainty". In: *European Journal of Operational Research* (2023).

[19] Yasmine Beck and Martin Schmidt. "A gentle and incomplete introduction to bilevel optimization". In: *Lecture notes* (2021).

[20] Robert G Beiko. "Telling the whole story in a 10,000-genome world". In: *Biology Direct* 6.1 (2011), pp. 1–36.

[21] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Vol. 28. Princeton University Press, 2009.

[22] Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer, and Arkadi Nemirovski. "Adjustable robust solutions of uncertain linear programs". In: *Mathematical Programming* 99.2 (2004), pp. 351–376.

[23] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: a methodological tour d'horizon". In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[24] David Bergman, Teng Huang, Philip Brooks, Andrea Lodi, and Arvind U Raghunathan. "JANOS: an integrated predictive and prescriptive modeling framework". In: *INFORMS Journal on Computing* 34.2 (2022), pp. 807–816.

[25] Giulia Bernardini, Leo van Iersel, Esther Julien, and Leen Stougie. "Constructing phylogenetic networks via cherry picking and machine learning". In: *Algorithms for Molecular Biology* 18.1 (2023), p. 13.

[26] Giulia Bernardini, Leo van Iersel, Esther Julien, and Leen Stougie. "Inferring phylogenetic networks from multifurcating trees via cherry picking and machine learning". In: *Molecular Phylogenetics and Evolution* 199 (2024), p. 108137.

[27] Giulia Bernardini, Leo van Iersel, Esther Julien, and Leen Stougie. "Reconstructing phylogenetic networks via cherry picking and machine learning". In: *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*. Vol. 242. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2022, p. 16.

[28] Dimitris Bertsimas and Constantine Caramanis. “Finite adaptability in multistage linear optimization”. In: *IEEE Transactions on Automatic Control* 55.12 (2010), pp. 2751–2766.

[29] Dimitris Bertsimas, Jack Dunn, and Yuchen Wang. “Near-optimal nonlinear regression trees”. In: *Operations Research Letters* 49.2 (2021), pp. 201–206.

[30] Dimitris Bertsimas and Iain Dunning. “Multistage robust mixed-integer optimization with adaptive partitions”. In: *Operations Research* 64.4 (2016), pp. 980–998.

[31] Dimitris Bertsimas and Angelos Georghiou. “Binary decision rules for multistage adaptive mixed-integer optimization”. In: *Mathematical Programming* 167 (2018), pp. 395–433.

[32] Dimitris Bertsimas and Angelos Georghiou. “Design of near optimal decision rules in multistage adaptive mixed-integer optimization”. In: *Operations Research* 63.3 (2015), pp. 610–627.

[33] Dimitris Bertsimas, Vishal Gupta, and Nathan Kallus. “Data-driven robust optimization”. In: *Mathematical Programming* 167 (2018), pp. 235–292.

[34] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. *The SCIP Optimization Suite 8.0*. Technical Report. Optimization Online, Dec. 2021. URL: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[35] Burcu Beykal, Styliani Avraamidou, Ioannis PE Pistikopoulos, Melis Onel, and Efstratios N Pistikopoulos. “Domino: Data-driven optimization of bi-level mixed-integer nonlinear problems”. In: *Journal of Global Optimization* 78 (2020), pp. 1–36.

[36] Ananya Bhattacharjee and Md Shamsuzzoha Bayzid. “Machine learning based imputation techniques for estimating phylogenetic trees from incomplete distance matrices”. In: *BMC Genomics* 21 (2020), pp. 1–14.

[37] Magnus Bordewich, Simone Linz, Katherine St John, and Charles Semple. “A reduction algorithm for computing the hybridization number of two trees”. In: *Evolutionary Bioinformatics* 3 (2007), p. 117693430700300017.

[38] Magnus Bordewich and Charles Semple. “Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 4.3 (2007), pp. 458–466.

[39] Magnus Bordewich and Charles Semple. “Computing the minimum number of hybridization events for a consistent evolutionary history”. In: *Discrete Applied Mathematics* 155.8 (2007), pp. 914–928.

[40] Sander Borst, Leo van Iersel, Mark Jones, and Steven Kelk. “New FPT algorithms for finding the temporal hybridization number for sets of phylogenetic trees”. In: *Algorithmica* (2022).

[41] Luis Boto. “Horizontal gene transfer in evolution: facts and challenges”. In: *Proceedings of the Royal Society B: Biological Sciences* 277.1683 (2010), pp. 819–827.

[42] Jerome Bracken and James T McGill. “Mathematical programs with optimization problems in the constraints”. In: *Operations Research* 21.1 (1973), pp. 37–44.

[43] Theo Broeders. “A graph-neural-network approach for reconstructing temporal networks”. Master’s Thesis. Delft University of Technology, 2024. URL: https://resolver.tudelft.nl/uuid:35a897a9-b80a-471e-aaf5-5f9294212955.

[44] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. “A survey of monte carlo tree search methods”. In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[45] Christoph Buchheim and Jannis Kurtz. “Min–max–min robust combinatorial optimization”. In: *Mathematical Programming* 163.1 (2017), pp. 1–23.

[46] Christoph Buchheim and Jannis Kurtz. “Robust combinatorial optimization under convex and discrete cost uncertainty”. In: *EURO Journal on Computational Optimization* 6.3 (2018), pp. 211–238.

[47] Trevor Campbell and Jonathan P How. “Bayesian nonparametric set construction for robust optimization”. In: *2015 American Control Conference (ACC)*. IEEE. 2015, pp. 4216–4221.

[48] Alberto Caprara, Margarida Carvalho, Andrea Lodi, and Gerhard J Woeginger. “Bilevel knapsack with interdiction constraints”. In: *INFORMS Journal on Computing* 28.2 (2016), pp. 319–333.

[49] Gabriel Cardona, Francesc Rosselló, and Gabriel Valiente. “Comparison of tree-child phylogenetic networks”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 6.4 (2008), pp. 552–569.

[50] Margarida Carvalho, Gabriele Dragotto, Andrea Lodi, and Sriram Sankaranarayanan. “Integer programming games: a gentle computational overview”. In: *Tutorials in Operations Research: Advancing the Frontiers of OR/MS: From Methodologies to Applications*. INFORMS, 2023, pp. 31–51.

[51] Francesco Ceccon, Jordan Jalving, Joshua Haddad, Alexander Thebelt, Calvin Tsay, Carl D Laird, and Ruth Misener. “OMLT: Optimization & machine learning toolkit”. In: *arXiv preprint arXiv:2202.02414* (2022).

[52] Timothy CY Chan, Bo Lin, and Shoshanna Saxe. “A machine learning approach to solving large bilevel and stochastic programs: Application to cycling network design”. In: *arXiv preprint arXiv:2209.09404* (2022).

[53] Wenbo Chen, Seonho Park, Mathieu Tanneau, and Pascal Van Hentenryck. “Learning optimization proxies for large-scale security-constrained economic dispatch”. In: *Electric Power Systems Research* 213 (2022), p. 108566.

[54] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. “Maximum resilience of artificial neural networks”. In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2017, pp. 251–268.

[55] Benny Chor and Tamir Tuller. “Maximum likelihood of evolutionary trees is hard”. In: *Annual International Conference on Research in Computational Molecular Biology*. Springer. 2005, pp. 296–310.

[56] Joshua Collins, Simone Linz, and Charles Semple. “Quantifying hybridization in realistic time”. In: *Journal of Computational Biology* 18.10 (2011), pp. 1305–1318.

[57] George B Dantzig. “Discrete-variable extremum problems”. In: *Operations Research* 5.2 (1957), pp. 266–288.

[58] James H Degnan and Noah A Rosenberg. “Gene tree discordance, phylogenetic inference and the multispecies coalescent”. In: *Trends in Ecology and Evolution* 24.6 (2009), pp. 332–340.

[59] Stephan Dempe. “Bilevel optimization: theory, algorithms, applications and a bibliography”. In: *Bilevel Optimization: Advances and Next Challenges* (2020), pp. 581–672.

[60] Scott DeNegre. *Interdiction and discrete bilevel linear programming*. Lehigh University, 2011.

[61] Scott T DeNegre and Ted K Ralphs. “A branch-and-cut algorithm for integer bilevel linear programs”. In: *Operations Research and Cyber-Infrastructure*. Springer. 2009, pp. 65–78.

[62] Boris Detienne, Henri Lefebvre, Enrico Malaguti, and Michele Monaci. “Adjustable robust optimization with objective uncertainty”. In: *European Journal of Operational Research* 312.1 (2024), pp. 373–384.

[63] Gabriele Dragotto, Amine Boukhtouta, Andrea Lodi, and Mehdi Taobane. “The critical node game”. In: *Journal of Combinatorial Optimization* 47.5 (2024), p. 74.

[64] Justin Dumouchelle, Esther Julien, Jannis Kurtz, and Elias B Khalil. “Deep learning for two-stage robust integer optimization”. In: *arXiv preprint arXiv:2310.04345* (2024).

[65] Justin Dumouchelle, Esther Julien, Jannis Kurtz, and Elias B Khalil. “Neur2BiLO: Neural bilevel optimization”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=esVleaqkRc.

[66] Justin Dumouchelle, Esther Julien, Jannis Kurtz, and Elias B Khalil. “Neur2RO: Neural two-stage robust optimization”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=T5Xb0iGCCv.

[67] Justin Dumouchelle, Rahul Patel, Elias B Khalil, and Merve Bodur. “Neur2SP: Neural Two-Stage Stochastic Programming”. In: *Advances in Neural Information Processing Systems* 35 (2022).

[68] Arkadiy Dushatskiy, Esther Julien, Leen Stougie, and Leo van Iersel. “Solving the tree containment problem using graph neural networks”. In: *Transactions on Machine Learning Research* (2024). ISSN: 2835-8856. URL: https://openreview.net/forum?id=nK5MazeIpn.

[69] Adam Elmachtoub and Paul Grigas. “Smart "predict, then optimize"”. In: *Management Science* 68.1 (2022), pp. 9–26.

[70] Péter L Erdős, Charles Semple, and Mike Steel. “A class of phylogenetic networks reconstructable from ancestral profiles”. In: *Mathematical Biosciences* 313 (2019), pp. 33–40.

[71] Adejuyigbe O Fajemisin, Donato Maragno, and Dick den Hertog. “Optimization with constraint learning: a framework and survey”. In: *European Journal of Operational Research* (2023).

[72] Joseph Felsenstein. *Inferring phylogenies*. Vol. 2. Sinauer Associates Sunderland, MA, 2004.

[73] Mareike Fischer and Andrew Francis. “How tree-based is my network? Proximity measures for unrooted phylogenetic networks”. In: *Discrete Applied Mathematics* 283 (2020), pp. 98–114.

[74] Mareike Fischer, Tom Niklas Hamann, and Kristina Wicke. “How far is my network from being edge-based? Proximity measures for edge-basedness of unrooted phylogenetic networks”. In: *arXiv preprint arXiv:2207.01370* (2022).

[75] Matteo Fischetti and Jason Jo. “Deep neural networks and mixed integer linear optimization”. In: *Constraints* 23.3 (2018), pp. 296–309.

[76] Matteo Fischetti, Ivana Ljubić, Michele Monaci, and Markus Sinnl. “A new general-purpose algorithm for mixed-integer bilevel linear programs”. In: *Operations Research* 65.6 (2017), pp. 1615–1637.

[77] Matteo Fischetti, Ivana Ljubić, Michele Monaci, and Markus Sinnl. “Intersection cuts for bilevel optimization”. In: *Integer Programming and Combinatorial Optimization: 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings 18*. Springer. 2016, pp. 77–88.

[78] Christodoulos A Floudas. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[79] Pirmin Fontaine and Stefan Minner. “Benders decomposition for discrete–continuous linear bilevel problems with application to traffic network design”. In: *Transportation Research Part B: Methodological* 70 (2014), pp. 163–172.

[80] Andrew Francis, Charles Semple, and Mike Steel. “New characterisations of tree-based networks and proximity measures”. In: *Advances in Applied Mathematics* 93 (2018), pp. 93–107.

[81] Andrew R Francis and Mike Steel. “Which phylogenetic networks are merely trees with additional arcs?” In: *Systematic Biology* 64.5 (2015), pp. 768–777.

[82] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. “Exact combinatorial optimization with graph convolutional neural networks”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[83] Alireza Ghahtarani, Ahmed Saif, Alireza Ghasemi, and Erick Delage. “A double-oracle, logic-based Benders decomposition approach to solve the K-adaptability problem”. In: *Computers & Operations Research* 155 (2023), p. 106243.

[84] Shraddha Ghatkar, Ashwin Arulselvan, and Alec Morton. “Solution techniques for bi-level knapsack problems”. In: *Computers & Operations Research* 159 (2023), p. 106343.

[85] Marc Goerigk and Jannis Kurtz. “Data-driven Prediction of Relevant Scenarios for Robust Optimization”. In: *arXiv e-prints* (2022), arXiv–2203.

[86] Marc Goerigk and Jannis Kurtz. “Data-driven robust optimization using deep neural networks”. In: *Computers & Operations Research* 151 (2023), p. 106087.

[87] Marc Goerigk, Jannis Kurtz, Martin Schmidt, and Johannes Thürauf. “Connections between robust and bilevel optimization”. In: *Open Journal of Mathematical Optimization* 6 (2025), pp. 1–17.

[88] Marc Goerigk, Stefan Lendl, and Lasse Wulf. “On the complexity of robust multi-stage problems with discrete recourse”. In: *Discrete Applied Mathematics* 343 (2024), pp. 355–370.

[89] Benjamin E Goulet, Federico Roda, and Robin Hopkins. “Hybridization in plants: old ideas, new techniques”. In: *Plant Physiology* 173.1 (2017), pp. 65–78.

[90] Bjarne Grimstad and Henrik Andersson. “ReLU networks as surrogate models in mixed-integer linear programs”. In: *Computers & Chemical Engineering* 131 (2019), p. 106580.

[91] Zeynep H Gümüş and Christodoulos A Floudas. “Global optimization of mixed-integer bilevel programming problems”. In: *Computational Management Science* 2 (2005), pp. 181–212.

[92] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2020. URL: http://www.gurobi.com.

[93] Gurobi Optimization, LLC. *Gurobi Machine Learning*. 2024. URL: https://gurobi-machinelearning.readthedocs.io/en/stable/.

[94] Carlton Gyles and Patrick Boerlin. “Horizontally transferred genetic elements and their role in pathogenesis of bacterial disease”. In: *Veterinary Pathology* 51.2 (2014), pp. 328–340.

[95] Grani A Hanasusanto, Daniel Kuhn, and Wolfram Wiesemann. “K-adaptability in two-stage robust binary programming”. In: *Operations Research* 63.4 (2015), pp. 877–891.

[96] Dov Harel and Robert Endre Tarjan. “Fast algorithms for finding nearest common ancestors”. In: *SIAM Journal on Computing* 13.2 (1984), pp. 338–355. DOI: 10.1137/0213024.

[97] Momoko Hayamizu. “A structure theorem for rooted binary phylogenetic networks and its implications for tree-based networks”. In: *SIAM Journal on Discrete Mathematics* 35.4 (2021), pp. 2490–2516.

[98] He He, Hal Daume III, and Jason M Eisner. “Learning to search in branch and bound algorithms”. In: *Advances in Neural Information Processing Systems* 27 (2014), pp. 3293–3301.

[99] Katharina T Huber, Vincent Moulton, and Andreas Spillner. “Phylogenetic consensus networks: Computing a consensus of 1-nested phylogenetic networks”. In: *arXiv preprint arXiv:2107.09696* (2021).

[100] Katharina T Huber, Leo van Iersel, Remie Janssen, Mark Jones, Vincent Moulton, Yukihiro Murakami, and Charles Semple. “Orienting undirected phylogenetic networks”. In: *Journal of Computer and System Sciences* 140 (2024), p. 103480.

[101] Peter J Humphries, Simone Linz, and Charles Semple. “Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies”. In: *Bulletin of Mathematical Biology* 75.10 (2013), pp. 1879–1890.

[102] Daniel H Huson and Simone Linz. “Autumn algorithm—computation of hybridization networks for realistic phylogenetic trees”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 15.2 (2016), pp. 398–410.

[103] Daniel H Huson, Regula Rupp, Vincent Berry, Philippe Gambette, and Christophe Paul. “Computing galled networks from real data”. In: *Bioinformatics* 25.12 (2009), pp. i85–i93.

[104] Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

[105] Daniel H Huson and Celine Scornavacca. “Dendroscope 3: an interactive tool for rooted phylogenetic trees and networks”. In: *Systematic Biology* 61.6 (2012), pp. 1061–1067.

[106] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. “Beyond manual tuning of hyperparameters”. In: *KI-Künstliche Intelligenz* 29 (2015), pp. 329–337.

[107] Remie Janssen, Mark Jones, Péter L Erdős, Leo van Iersel, and Celine Scornavacca. “Exploring the tiers of rooted phylogenetic network space using tail moves”. In: *Bulletin of Mathematical Biology* 80.8 (2018), pp. 2177–2208.

[108] Remie Janssen and Pengyu Liu. “Comparing the topology of phylogenetic network generators”. In: *Journal of Bioinformatics and Computational Biology* 19.06 (2021), p. 2140012.

[109] Remie Janssen and Yukihiro Murakami. “On cherry-picking and network containment”. In: *Theoretical Computer Science* 856 (2021), pp. 121–150.

[110] Laura Jetten and Leo van Iersel. “Nonbinary tree-based phylogenetic networks”. In: *IEEE/ACM transactions on computational biology and bioinformatics* 15.1 (2016), pp. 205–217.

[111] Esther Julien, Krzysztof Postek, and Ş İlker Birbil. “Machine learning for K-adaptability in two-stage robust optimization”. In: *INFORMS Journal on Computing* (2024).

[112] Nicolas Kämmerling and Jannis Kurtz. “Oracle-based algorithms for binary two-stage robust optimization”. In: *Computational Optimization and Applications* 77 (2020), pp. 539–569.

[113] Justin Katz, Iosif Pappas, Styliani Avraamidou, and Efstratios N. Pistikopoulos. "The integration of explicit MPC and ReLU based neural networks". In: *IFAC-PapersOnLine* 53.2 (2020), pp. 11350–11355.

[114] Elias B Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. "Learning combinatorial optimization algorithms over graphs". In: *Advances in Neural Information Processing Systems* 30 (2017).

[115] Elias B Khalil, Christopher Morris, and Andrea Lodi. "MIP-GNN: A data-driven framework for guiding combinatorial solvers". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 9. 2022, pp. 10219–10227.

[116] Elias B Khalil, Pashootan Vaezipoor, and Bistra Dilkina. "Finding backdoors to integer programs: A Monte Carlo Tree Search framework". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 4. 2022, pp. 3786–3795.

[117] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[118] Thomas Kleinert, Martine Labbé, Ivana Ljubić, and Martin Schmidt. "A survey on mixed-integer programming techniques in bilevel optimization". In: *EURO Journal on Computational Optimization* 9 (2021), p. 100007.

[119] Polyxeni-M Kleniati and Claire S Adjiman. "A generalization of the branch-and-sandwich algorithm: from continuous to mixed-integer nonlinear bilevel problems". In: *Computers & Chemical Engineering* 72 (2015), pp. 373–386.

[120] Stephan Koblmüller, Nina Duftner, Kristina M Sefc, Mitsuto Aibara, Martina Stipacek, MicLamprologinihel Blanc, Bernd Egger, and Christian Sturmbauer. "Reticulate phylogeny of gastropod-shell-breeding cichlids from Lake Tanganyika–the result of repeated introgressive hybridization". In: *BMC Evolutionary Biology* 7.1 (2007), pp. 1–13.

[121] Alyssa Kody, Samuel Chevalier, Spyros Chatzivasileiadis, and Daniel Molzahn. "Modeling the AC power flow equations with optimally compact neural networks: Application to unit commitment". In: *Electric Power Systems Research* 213 (2022), p. 108282. ISSN: 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2022.108282. URL: https://www.sciencedirect.com/science/article/pii/S0378779622004771.

[122] Sungsik Kong, Joan Carles Pons, Laura Kubatko, and Kristina Wicke. "Classes of explicit phylogenetic networks and their biological and mathematical significance". In: *Journal of Mathematical Biology* 84.6 (2022), p. 47.

[123] James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. "Learning hard optimization problems: A data generation perspective". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 24981–24992.

[124] Jan Kronqvist, Boda Li, Jan Rolfes, and Shudian Zhao. "Alternating mixed-integer programming and neural network training for approximating stochastic two-stage problems". In: *International Conference on Machine Learning, Optimization, and Data Science*. Springer. 2023, pp. 124–139.

[125] Nikita Kulikov, Fatemeh Derakhshandeh, and Christoph Mayer. “Machine learning can be as good as maximum likelihood when reconstructing phylogenetic trees and determining the best evolutionary model on four taxon alignments”. In: *Molecular Phylogenetics and Evolution* 200 (2024), p. 108181.

[126] Sudhir Kumar and Sudip Sharma. “Evolutionary sparse learning for phylogenomics”. In: *Molecular Biology and Evolution* 38.11 (2021), pp. 4674–4682.

[127] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. “Adversarial Machine Learning at Scale”. In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=BJm4T4Kgx.

[128] Jannis Kurtz. “Approximation algorithms for min-max-min robust optimization and K-adaptability under objective uncertainty”. In: *arXiv preprint arXiv:2106.03107* (2023).

[129] Sunhyeon Kwon, Hwayong Choi, and Sungsoo Park. “Solving bilevel knapsack problem using graph neural networks”. In: *arXiv preprint arXiv:2211.13436* (2022).

[130] Henri Lefebvre, Enrico Malaguti, and Michele Monaci. “Adjustable robust optimization with discrete uncertainty”. In: *INFORMS Journal on Computing* (2023).

[131] Markus Leitner, Ivana Ljubić, Michele Monaci, Markus Sinnl, and Kübra Tanınmış. “An exact method for binary fortification games”. In: *European Journal of Operational Research* 307.3 (2023), pp. 1026–1039.

[132] C Randal Linder, Bernard ME Moret, Luay Nakhleh, and Tandy Warnow. “Network (reticulate) evolution: biology, models, and algorithms”. In: *The Ninth Pacific Symposium on Biocomputing (PSB)*. 2004.

[133] C Randal Linder and Loren H Rieseberg. “Reconstructing patterns of reticulate evolution in plants”. In: *American journal of botany* 91.10 (2004), pp. 1700–1708.

[134] Simone Linz and Charles Semple. “Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies”. In: *Advances in Applied Mathematics* 105 (2019), pp. 102–129.

[135] Simone Linz and Charles Semple. “Caterpillars on three and four leaves are sufficient to reconstruct binary normal networks”. In: *Journal of Mathematical Biology* 81.4 (2020), pp. 961–980.

[136] Simone Linz, Charles Semple, and Tanja Stadler. “Analyzing and reconstructing reticulation networks under timing constraints”. In: *Journal of Mathematical Biology* 61 (2010), pp. 715–737.

[137] Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. “Evolution of heuristics: towards efficient automatic algorithm design using large language model”. In: *Forty-first International Conference on Machine Learning*. 2024.

[138] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “DARTS: differentiable architecture search”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=S1eYHoC5FX.

[139] Risheng Liu, Jiaxin Gao, Jin Zhang, Deyu Meng, and Zhouchen Lin. “Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.12 (2021), pp. 10045–10067.

[140] Michele Lombardi, Michela Milano, and Andrea Bartolini. “Empirical decision model learning”. In: *Artificial Intelligence* 244 (2017), pp. 343–367.

[141] Manuel Loth, Michele Sebag, Youssef Hamadi, and Marc Schoenauer. “Bandit-based search for constraint programming”. In: *Principles and Practice of Constraint Programming: 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings 19*. Springer. 2013, pp. 464–480.

[142] Leonardo Lozano and J Cole Smith. “A value-function-based exact approach for the bilevel mixed-integer programming problem”. In: *Operations Research* 65.3 (2017), pp. 768–786.

[143] James Mallet. “Hybridization as an invasion of the genome”. In: *Trends in Ecology and Evolution* 20.5 (2005), pp. 229–237.

[144] James Mallet, Nora Besansky, and Matthew W Hahn. “How reticulated are species?” In: *BioEssays* 38.2 (2016), pp. 140–149.

[145] Donato Maragno, Holly Wiberg, Dimitris Bertsimas, Ş İlker Birbil, Dick den Hertog, and Adejuyigbe O Fajemisin. “Mixed-integer optimization with constraint learning”. In: *Operations Research* (2023).

[146] Tom V Mathew and KV Krishna Rao. “Introduction to Transportation Engineering, Traffic Assignment”. In: *Lecture notes* (2006).

[147] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. “Reinforcement learning for combinatorial optimization: a survey”. In: *Computers & Operations Research* 134 (2021), p. 105400.

[148] Chen Meng and Laura Salter Kubatko. “Detecting hybrid speciation in the presence of incomplete lineage sorting using gene tree incongruence: a model”. In: *Theoretical Population Biology* 75.1 (2009), pp. 35–45.

[149] Sajad Mirzaei and Yufeng Wu. “Fast construction of near parsimonious hybridization networks for multiple phylogenetic trees”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 13.3 (2015), pp. 565–570.

[150] Ioana Molan and Martin Schmidt. “Using neural networks to solve linear bilevel problems with unknown lower level”. In: *Optimization Letters* (2023), pp. 1–21.

[151] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. “On the number of linear regions of deep neural networks”. In: *Advances in Neural Information Processing Systems* 27 (2014).

[152] Alec Morton, Ashwin Arulselvan, and Ranjeeta Thomas. “Allocation rules for global donors”. In: *Journal of Health Economics* 58 (2018), pp. 67–75.

[153] Ilgiz Murzakhanov, Andreas Venzke, George S Misyris, and Spyros Chatzivasileiadis. “Neural networks for encoding dynamic security-constrained optimal power flow”. In: *arXiv preprint arXiv:2003.07939* (2020).

[154] Almir Mutapcic and Stephen Boyd. “Cutting-set methods for robust convex optimization with pessimizing oracles”. In: *Optimization Methods & Software* 24.3 (2009), pp. 381–406.

[155] Luay Nakhleh. “Evolutionary phylogenetic networks: models and issues”. In: *Problem Solving Handbook in Computational Biology and Bioinformatics*. Springer, 2010, pp. 125–158.

[156] Luay Nakhleh, Guohua Jin, Fengmei Zhao, and John Mellor-Crummey. “Reconstructing phylogenetic networks using maximum parsimony”. In: *2005 IEEE Computational Systems Bioinformatics Conference (CSB'05)*. IEEE. 2005, pp. 93–102.

[157] Masatoshi Nei and Sudhir Kumar. *Molecular evolution and phylogenetics*. Oxford university press, 2000.

[158] George L Nemhauser and Laurence A Wolsey. *Integer and combinatorial optimization*. New York: Wiley-Interscience, 1988. ISBN: 978-0471359432.

[159] Chao Ning and Fengqi You. “Data-driven adaptive nested robust optimization: general modeling framework and efficient computational algorithm for decision making under uncertainty”. In: *AIChE Journal* 63.9 (2017), pp. 3790–3817.

[160] Chao Ning and Fengqi You. “Data-driven decision making under uncertainty integrating robust optimization with principal component analysis and kernel smoothing methods”. In: *Computers & Chemical Engineering* 112 (2018), pp. 190–210.

[161] Fabio Pardi and Celine Scornavacca. “Reconstructible phylogenetic networks: do not distinguish the indistinguishable”. In: *PLoS Computational Biology* 11.4 (2015), e1004135.

[162] H Park, G Jin, and L Nakhleh. “Algorithmic strategies for estimating the amount of reticulation from a collection of gene trees”. In: *Proceedings of the 9th Annual International Conference on Computational Systems Biology*. Citeseer. 2010, pp. 114–123.

[163] Hyun Jung Park and Luay Nakhleh. “Inference of reticulate evolutionary histories by maximum likelihood: the performance of information criteria”. In: *BMC Bioinformatics*. Vol. 13. 19. BioMed Central. 2012, S12.

[164] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An imperative style, high-performance deep learning library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035.

[165] Remigijus Paulavičius and Claire S Adjiman. “New bounding schemes and algorithmic options for the Branch-and-Sandwich algorithm”. In: *Journal of Global Optimization* 77.2 (2020), pp. 197–225.

[166] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[167] Fotios Petropoulos, Gilbert Laporte, Emel Aktas, Sibel A Alumur, Claudia Archetti, Hayriye Ayhan, Maria Battarra, Julia A Bennell, Jean-Marie Bourjolly, John E Boylan, et al. "Operational research: Methods and applications". In: *arXiv preprint arXiv:2303.14217* (2023).

[168] Teresa Piovesan and Steven M Kelk. "A simple fixed parameter tractable algorithm for computing the hybridization number of two (not necessarily binary) trees". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 10.1 (2012), pp. 18–25.

[169] Joan Carles Pons, Celine Scornavacca, and Gabriel Cardona. "Generation of Level-k LGT networks". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 17.1 (2019), pp. 158–164.

[170] Krzysztof Postek and Dick den Hertog. "Multistage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set". In: *INFORMS Journal on Computing* 28.3 (2016), pp. 553–574.

[171] David Rey. "Computational benchmarking of exact methods for the bilevel discrete network design problem". In: *Transportation Research Procedia* 47 (2020), pp. 11–18.

[172] David Rey. "Optimization and game-theoretical methods for transportation systems". PhD thesis. Toulouse 3 Paul Sabatier, 2023.

[173] Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy. "Guiding combinatorial optimization with UCT". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 9th International Conference, CPAIOR 2012, Nantes, France, May 28–June1, 2012. Proceedings 9*. Springer. 2012, pp. 356–361.

[174] Buser Say, Ga Wu, Yu Qing Zhou, and Scott Sanner. "Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming". In: *IJCAI*. 2017, pp. 750–756.

[175] Lara Scavuzzo, Karen Aardal, Andrea Lodi, and Neil Yorke-Smith. "Machine learning augmented branch and bound for mixed integer linear programming". In: *Mathematical Programming* (2024), pp. 1–44.

[176] Charles Semple and Gerry Toft. "Trinets encode orchard phylogenetic networks". In: *Journal of Mathematical Biology* 83.3 (2021), pp. 1–20.

[177] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. "Bounding and counting linear regions of deep neural networks". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4558–4566.

[178] Chao Shang, Xiaolin Huang, and Fengqi You. "Data-driven robust optimization based on kernel learning". In: *Computers & Chemical Engineering* 106 (2017), pp. 464–479.

[179] Chao Shang and Fengqi You. “A data-driven robust optimization approach to scenario-based stochastic model predictive control”. In: *Journal of Process Control* 75 (2019), pp. 24–39.

[180] Feifei Shen, Liang Zhao, Wenli Du, Weimin Zhong, and Feng Qian. “Large-scale industrial energy systems optimization under uncertainty: A data-driven robust optimization approach”. In: *Applied Energy* 259 (2020), p. 114199.

[181] Ankur Sinha, Samish Bedi, and Kalyanmoy Deb. “Bilevel optimization based on kriging approximations of lower level optimal value function”. In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2018, pp. 1–8.

[182] Ankur Sinha, Zhichao Lu, Kalyanmoy Deb, and Pekka Malo. “Bilevel optimization based on iterative approximation of multiple mappings”. In: *Journal of Heuristics* 26 (2020), pp. 151–185.

[183] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. “A review on bilevel optimization: from classical to evolutionary approaches and applications”. In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 276–295.

[184] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. “Solving optimistic bilevel programs by iteratively approximating lower level optimal value function”. In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2016, pp. 1877–1884.

[185] Megan L Smith and Matthew W Hahn. “Phylogenetic inference using generative adversarial networks”. In: *Bioinformatics* 39.9 (2023), btad543.

[186] Claudia Solís-Lemus, Paul Bastide, and Cécile Ané. “PhyloNetworks: a package for phylogenetic networks”. In: *Molecular Biology and Evolution* 34.12 (2017), pp. 3292–3298.

[187] Shannon M Soucy, Jinling Huang, and Johann Peter Gogarten. “Horizontal gene transfer: building the web of life”. In: *Nature Reviews Genetics* 16.8 (2015), pp. 472–482.

[188] Anirudh Subramanyam. “A Lagrangian dual method for two-stage robust optimization with binary uncertainties”. In: *Optimization and Engineering* 23.4 (2022), pp. 1831–1871.

[189] Anirudh Subramanyam, Chrysanthos E Gounaris, and Wolfram Wiesemann. “K-adaptability in two-stage mixed-integer robust optimization”. In: *Mathematical Programming Computation* 12.2 (2020), pp. 193–224.

[190] Merel Susanna. “Making phylogenetic networks orchard: Algorithms to determine if a phylogenetic network is orchard and to transform non-orchard to orchard networks”. Bachelor’s Thesis. Delft University of Technology, 2022. URL: http://resolver.tudelft.nl/uuid:724ac2af-e569-4586-b367-288fef890252.

[191] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. “Monte Carlo tree search: A review of recent modifications and applications”. In: *Artificial Intelligence Review* (2022), pp. 1–66.

[192] Gergely J Szöllősi, Adrián Arellano Davín, Eric Tannier, Vincent Daubin, and Bastien Boussau. "Genome-scale phylogenetic analysis finds extensive gene transfer among fungi". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 370.1678 (2015), p. 20140335.

[193] Sahar Tahernejad, Ted K Ralphs, and Scott T DeNegre. "A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation". In: *Mathematical Programming Computation* 12 (2020), pp. 529–568.

[194] Bo Tang and Elias B Khalil. "Pyepo: A pytorch-based end-to-end predict-then-optimize library for linear and integer programming". In: *Mathematical Programming Computation* 16.3 (2024), pp. 297–335.

[195] Yen Tang, Jean-Philippe P Richard, and J Cole Smith. "A class of algorithms for mixed-integer bilevel min–max optimization". In: *Journal of Global Optimization* 66 (2016), pp. 225–262.

[196] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. "Evaluating robustness of neural networks with mixed integer programming". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HyGIdiRqtm.

[197] Man Yiu Tsang, Karmel S Shehadeh, and Frank E Curtis. "An inexact column-and-constraint generation method to solve two-stage robust optimization problems". In: *Operations Research Letters* 51.1 (2023), pp. 92–98.

[198] Theja Tulabandhula and Cynthia Rudin. "Robust optimization using machine learning for uncertainty sets". In: *arXiv preprint arXiv:1407.1097* (2014).

[199] Mark Turner, Antonia Chmiela, Thorsten Koch, and Michael Winkler. "PySCIPOpt-ML: Embedding trained machine learning models into mixed-integer programs". In: *arXiv preprint arXiv:2312.08074* (2023).

[200] Leo van Iersel, Remie Janssen, Mark Jones, and Yukihiro Murakami. "Orchard networks are trees with additional horizontal arcs". In: *Bulletin of Mathematical Biology* 84.8 (2022), pp. 1–21.

[201] Leo van Iersel, Remie Janssen, Mark Jones, Yukihiro Murakami, and Norbert Zeh. "A practical fixed-parameter algorithm for constructing tree-child networks from multiple binary trees". In: *Algorithmica* 84 (2022), pp. 917–960.

[202] Leo van Iersel, Remie Janssen, Mark Jones, Yukihiro Murakami, and Norbert Zeh. "A unifying characterization of tree-based networks and orchard networks using cherry covers". In: *Advances in Applied Mathematics* 129 (2021), p. 102222. DOI: 10.1016/j.aam.2021.102222.

[203] Leo van Iersel, Mark Jones, Esther Julien, and Yukihiro Murakami. "Making a network orchard by adding leaves". In: *23rd International Workshop on Algorithms in Bioinformatics (WABI 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2023.

[204] Leo van Iersel and Steven Kelk. "Constructing the simplest possible phylogenetic network from triplets". In: *Algorithmica* 60.2 (2011), pp. 207–235.

[205] Leo van Iersel, Steven Kelk, Regula Rupp, and Daniel Huson. “Phylogenetic networks do not need to be complex: using fewer reticulations to represent conflicting clusters”. In: *Bioinformatics* 26.12 (2010), pp. i124–i131.

[206] Leo van Iersel and Simone Linz. “A quadratic kernel for computing the hybridization number of multiple trees”. In: *Information Processing Letters* 113.9 (2013), pp. 318–323.

[207] Heinrich von Stackelberg. *Market structure and equilibrium*. Springer Science & Business Media, 2010.

[208] Heinrich von Stackelberg. *Marktform und Gleichgewicht*. German. Doctoral dissertation, University of Cologne. Vienna: Springer, 1934.

[209] Irina Wang, Cole Becker, Bart Van Parys, and Bartolomeo Stellato. “Learning Decision-Focused Uncertainty Sets in Robust Optimization”. In: *arXiv preprint arXiv:2305.19225* (2023).

[210] Keliang Wang, Leonardo Lozano, Carlos Cardonha, and David Bergman. “Optimizing over an ensemble of trained neural networks”. In: *INFORMS Journal on Computing* 35.3 (2023), pp. 652–674.

[211] Alan Washburn and Kevin Wood. “Two-person zero-sum games for network interdiction”. In: *Operations Research* 43.2 (1995), pp. 243–251.

[212] Dingqiao Wen, Yun Yu, and Luay Nakhleh. “Bayesian inference of reticulate phylogenies under the multispecies network coalescent”. In: *PLoS Genetics* 12.5 (2016), e1006006.

[213] Dingqiao Wen, Yun Yu, Jiafan Zhu, and Luay Nakhleh. “Inferring phylogenetic networks using PhyloNet”. In: *Systematic Biology* 67.4 (2018), pp. 735–740.

[214] Noah Weninger and Ricardo Fukasawa. “A fast combinatorial algorithm for the bilevel knapsack problem with interdiction constraints”. In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2023, pp. 438–452.

[215] Chris Whidden, Robert G Beiko, and Norbert Zeh. “Fast FPT algorithms for computing rooted agreement forests: theory and experiments”. In: *Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings 9*. Springer. 2010, pp. 141–153.

[216] Chris Whidden, Robert G. Beiko, and Norbert Zeh. “Fixed-parameter algorithms for maximum agreement forests”. In: *SIAM Journal on Computing* 42.4 (2013), pp. 1431–1466.

[217] David A Wickell and Fay-Wei Li. “On the evolutionary significance of horizontal gene transfers in plants”. In: *New Phytologist* 225.1 (2020), pp. 113–117.

[218] Stephen Willson. “Regular networks can be uniquely constructed from their trees”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8.3 (2010), pp. 785–796.

[219] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[220] Yufeng Wu. “Close lower and upper bounds for the minimum reticulate network of multiple phylogenetic trees”. In: *Bioinformatics* 26.12 (2010), pp. i140–i148.

[221] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. “A comprehensive survey on graph neural networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2020), pp. 4–24.

[222] İhsan Yanıkoğlu, Bram L Gorissen, and Dick den Hertog. “A survey of adjustable robust optimization”. In: *European Journal of Operational Research* 277.3 (2019), pp. 799–813.

[223] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. “ReEvo: large language models as hyper-heuristics with reflective evolution”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=483IPG0HWL.

[224] Yun Yu, Jianrong Dong, Kevin J Liu, and Luay Nakhleh. “Maximum likelihood inference of reticulate evolutionary histories”. In: *Proceedings of the National Academy of Sciences* 111.46 (2014), pp. 16448–16453.

[225] Yun Yu, Cuong Than, James H Degnan, and Luay Nakhleh. “Coalescent histories on phylogenetic networks and detection of hybridization despite incomplete lineage sorting”. In: *Systematic Biology* 60.2 (2011), pp. 138–149.

[226] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. “Deep sets”. In: *Advances in Neural Information Processing Systems* 30 (2017).

[227] Bo Zeng and Long Zhao. “Solving two-stage robust optimization problems using a column-and-constraint generation method”. In: *Operations Research Letters* 41.5 (2013), pp. 457–461.

[228] Chi Zhang, Huw A Ogilvie, Alexei J Drummond, and Tanja Stadler. “Bayesian inference of species networks from multilocus sequence data”. In: *Molecular Biology and Evolution* 35.2 (2018), pp. 504–517.

[229] Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. “A survey for solving mixed integer programming via machine learning”. In: *Neurocomputing* 519 (2023), pp. 205–217.

[230] Louxin Zhang. “On tree-based phylogenetic networks”. In: *Journal of Computational Biology* 23.7 (2016), pp. 553–565.

[231] Louxin Zhang, Niloufar Abhari, Caroline Colijn, and Yufeng Wu. “A fast and scalable method for inferring phylogenetic networks from trees by aligning lineage taxon strings”. In: *Genome Research* 33.7 (2023), pp. 1053–1060.

[232] Long Zhao and Bo Zeng. “An exact algorithm for two-stage robust optimization with mixed integer recourse problems”. In: *Optimization-Online. org* (2012).

[233] Bo Zhou, Ruiwei Jiang, and Siqian Shen. “Learning to solve bilevel programs with binary tender”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=PsDFgTosqb.

[234] Tujin Zhu and Yunpeng Cai. “Applying neural network to reconstruction of phylogenetic tree”. In: *2021 13th International Conference on Machine Learning and Computing*. ICMLC 2021. Shenzhen, China: Association for Computing Machinery, 2021, pp. 146–152. ISBN: 9781450389310. DOI: 10.1145/3457682.3457704.

[235] Marco Zugno, Juan Miguel Morales, Pierre Pinson, and Henrik Madsen. “A bilevel model for electricity retailers’ participation in a demand response market environment”. In: *Energy Economics* 36 (2013), pp. 182–197.