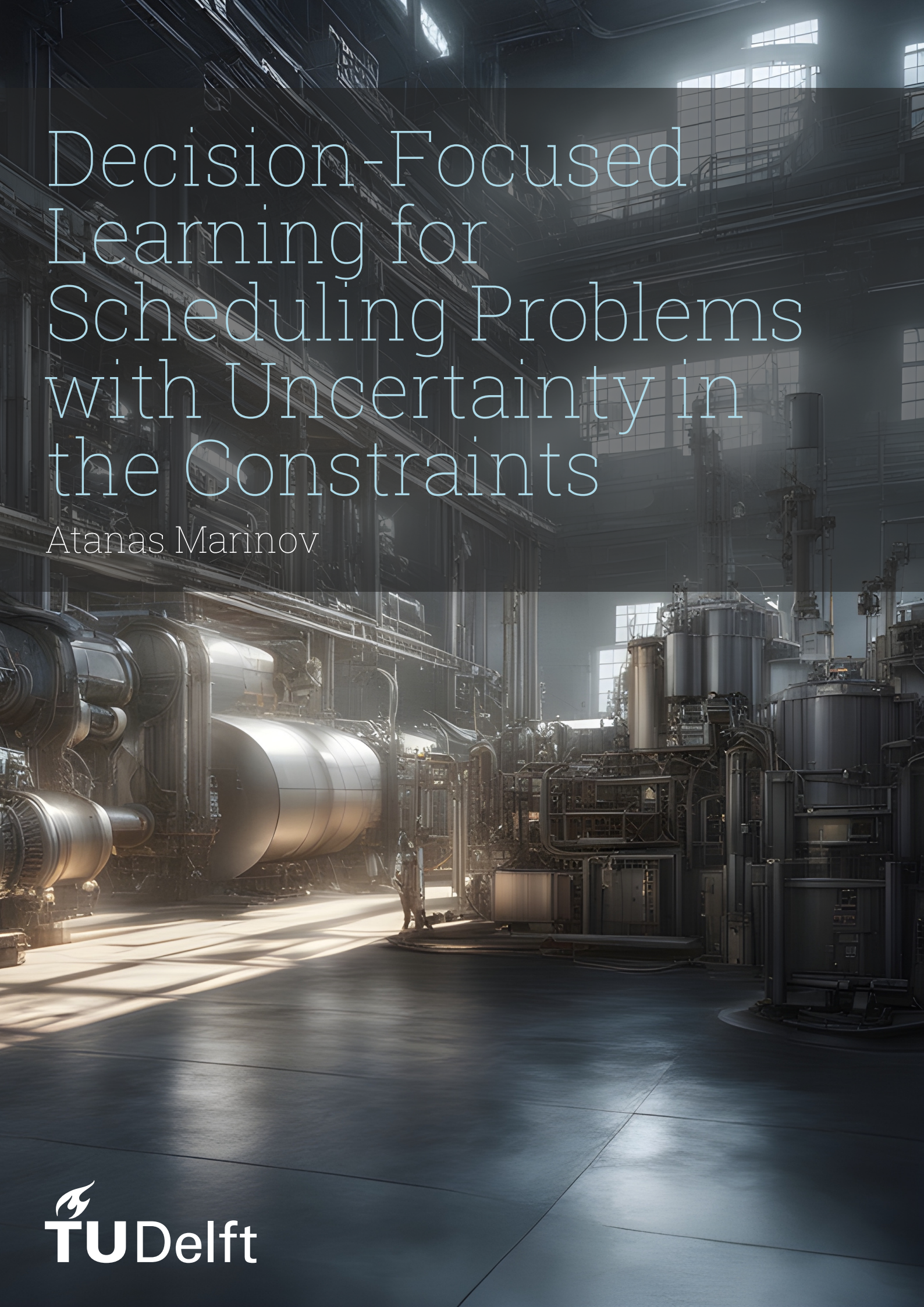


Decision-Focused Learning for Scheduling Problems with Uncertainty in the Constraints

Atanas Marinov

TU Delft

Decision-Focused Learning for Scheduling Problems with Uncertainty in the Constraints

by

Atanas Marinov

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday June 28, 2024 at 9:00 AM.

Student number:	4946251
Project duration:	November 13, 2023 – June 28, 2024
Thesis supervisor:	Prof. Dr. M. M. de Weerdt
Daily supervisor:	K. C. van den Houten
Thesis Committee Member:	Dr. D. M. J. Tax

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

TU Delft

Preface

This thesis represents the final step in my journey toward obtaining a Master of Science degree in Computer Science at Delft University of Technology. A journey that began six years ago in 2018 when I moved to the Netherlands to pursue my studies. A journey that is sadly nearing its end.

I am deeply grateful to my thesis coordinator, Prof. Dr. Mathijs de Weerdt, and my daily supervisor, Kim van den Houten, for their unwavering support and invaluable guidance throughout this process. Their insights and feedback have been instrumental in shaping this thesis, and their encouragement has been a constant source of motivation. I sincerely appreciate the time and effort they dedicated to guiding me through my research.

I also want to thank everyone who participated in the P+O meetings and the Decision-Focused Learning workshop organized by Kim and Noah. These events provided exceptional opportunities to deepen my understanding of the subject by hearing diverse perspectives from various researchers. Presenting my work during these occasions was not only a chance to receive invaluable feedback but also an opportunity to experience being a part of an inspiring research community.

Last but not least, I express my profound gratitude to my family. I was able to feel their support despite the fact that we were so far apart. It was vital for me to be able to cope with the challenges and stresses I faced. Their encouragement and belief in my abilities have been crucial in achieving this milestone. Thank you again for your unwavering love and support! I love you all!

Atanas Marinov
Delft, June 2024

Abstract

When addressing combinatorial optimization problems, the focus is predominantly on their computational complexity, and it is often forgotten to look at the bigger picture. As a result, it is common to miss critical details which could play a major role in the overall process. One such detail is the presence of uncertainty in the real world. A naive approach might directly predict values for the uncertain parameters, without taking into account that the ultimate goal is to obtain sound decisions. Consequently, in many cases, the resulting solutions are suboptimal. This challenge is precisely the premise behind Decision-Focused Learning (DFL) framework, which is a core of this work. This study pioneers the application of the DFL framework to scheduling problems with uncertain processing times, utilizing contextual features to predict these uncertainties. By employing the promising Score Function Gradient Estimation method, the research tackles the issue of non-differentiable regret loss functions in DFL. Key contributions include the development of techniques to enhance the performance of the Score Function, an in-depth analysis of DFL's applicability to complex scheduling scenarios, and a detailed evaluation of its strengths and weaknesses. This work not only demonstrates the potential of DFL in this new context but also lays the groundwork for future research and improvements in handling uncertainty in combinatorial optimization.

Contents

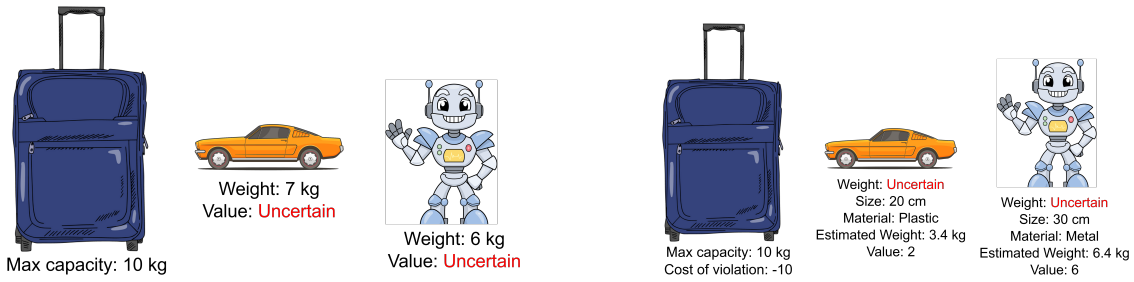
1

Introduction

The combinatorial optimization class encompasses a broad spectrum of problems, many of which have practical real-world applications. For instance, the traveling salesman problem (TSP) is employed in logistics, the minimum spanning tree is utilized in networking, and scheduling is crucial in manufacturing. A key distinction between their theoretical descriptions and real-world implementations is the presence of uncertainties in real-world scenarios (Arora et al., 2023; Sarin et al., 2010). In the textbook version of TSP, the time distances between the cities are predetermined and fixed. However, in reality, factors such as time of day, traffic, and road conditions can significantly affect travel times. Similarly, in scheduling, the processing times of tasks in a factory can be heavily influenced by operational issues such as machine malfunction and human error, as well as environmental factors like temperature and humidity.

We should acknowledge that we live in a world filled with uncertainties, and we cannot know the exact values that the parameters of a problem will take. Nevertheless, we still need to plan how to make our decisions efficiently when facing such uncertain situations. To better understand how to tackle uncertainty and the associated challenges, consider a slight variation of the knapsack problem where a mother needs to pack the suitcase of her child before an international flight. She needs to choose between a set of toys to put in the suitcase, where each toy has a weight and a value associated with it. The goal is to maximize the value of the toys picked while adhering to the maximum weight of the suitcase allowed by the flight operator. Assuming that the child is asleep while she is packing, she cannot know exactly which toys it would want, i.e., the value of each item is uncertain to her. This is an example of a knapsack problem where each item has an uncertain value. The problem might seem intricate from an optimization point of view, but it is definitely manageable if we consider it from the perspective of the mother. What she does in practice is to use her knowledge of her child, gathered from observing it playing with its toys or from their past trips, to decide which toys to put in the suitcase. Formally, the decision-maker (the mother) uses feature data and historical realizations available (her past interaction with the child) to approximate the values of the uncertain parameters (the value of each toy) in order to compute the optimal solution (which toys to take). Such methodology is often used in practice because of the commonplace occurrence of such contextual features. If we look at the example with concrete values as illustrated in Figure 1.1a, because of the maximum weight constraint the mother can select to take either the car or the robot, but not both. The decision is based on whether the child would prefer one or the other. If she incorrectly predicts the values of the toys, her choice would lead to a suboptimal selection with a lower objective value, i.e., the child will be unhappy because it would have preferred the other toy.

The knapsack problem with uncertain values is an example of an optimization problem where the uncertainty appears only in the objective function. As illustrated in the previous example, in such problems the uncertainty only impacts the objective value of the computed solution. An additional level of complexity is added if the uncertainty also appears in the constraints. Returning to the knapsack



(a) Knapsack with uncertain values

(b) Knapsack with uncertain weights

Figure 1.1: Knapsack examples. In Figure 1.1a, an example knapsack problem is presented where the values of the items are uncertain while the weights are known. Conversely, in Figure 1.1b, the weights are uncertain while the values are given. In the latter case, the mother can use the provided contextual features (size and material) to estimate the weights of the items. However, these estimates are NOT the actual true weights.

example from before, assume that now the child is awake and thus the value of each toy is known. However, this time the weighing scale is broken, and there is no way to know the exact weight of each item. Similar to before, an approach to tackle uncertainty could be the utilization of contextual features. This time, the mother could observe the items and use some of their features such as their size and the material they are made of to approximate their weight. An example of such an instance is presented in Figure 1.1b. However, these estimates are not the actual true values. Relying blindly on her estimates would mean that the optimal choice is to take both toys as their combined weight is 9.8 kg, which is just below the maximum allowed weight of 10 kg. Selecting both toys achieves an objective value of 8. However, estimating the exact weight of an item without a scale is practically impossible. In this example, if the mother is off by just 0.2 kg, or about 2% of the total weight, this would mean that the items are overweight. Then, depending on the policy of the flight operator, the mother would need to discard an item from the suitcase, consequently losing it, or would need to pay an additional charge. If we assume this additional penalty to have an objective value of -10, then the total objective value would be -2, meaning that a better decision could have been to select none of the toys or just one if she is sure that it is lighter than the maximum weight allowed. Ultimately, the presence of uncertainty in the constraints led to an infeasible solution (the overweight suitcase), which might not only be suboptimal but also incur an additional penalty because of the constraint violation (the lost item or paying further charges).

Fortunately, in most cases where uncertainty appears in practice, there are such feature data and historical realizations that can be utilized to estimate the uncertain parameter values. Such problems, which require first making predictions about the uncertain values and then solving, are commonly referred to as ”predict-then-optimize” problems. The earlier examples were simplified by assuming that predicting parameter values from features was straightforward. Actually, achieving good predictions is far from simple and is, in fact, the core challenge addressed in this work. Traditionally, machine learning models are used to predict the uncertain values, and these predicted values are then fed directly into an optimization algorithm to obtain a solution. The learning models are typically trained to minimize the prediction error based on historical realizations of the uncertain process, for example, by utilizing loss functions such as mean squared error, which measure the difference between the estimated values and the actual values. The issue with this approach is that the predictions from these models will never be perfectly accurate. When discrepancies between the predicted and the actual values arise, the outcome can be a suboptimal solution, resembling the knapsack examples mentioned earlier. While accurate predictions are important for the decision-making pipeline, the most critical aspect is making *sound* decisions. This is precisely the premise behind Decision-Focused Learning (DFL) (Elmachtoub and Grigas, 2022; Wilder et al., 2019; Mandi et al., 2023; Demirović et al., 2019). Instead of training models to make predictions that are merely close to the actual values, DFL focuses on making predictions that lead to good decisions by incorporating the decision error into the training process. In DFL, the parameters of the machine learning model are updated not based on the difference between predictions and actual values, but rather by using an optimization oracle (most commonly an optimization solver) to determine the difference between the true optimal solution and the optimal solution obtained using

the predicted values. This difference, known as regret, serves as a loss function in DFL that is used to update the parameters of the predictive model during training. By incorporating the decision error into the training process, DFL aims to produce models that facilitate effective decision-making. This approach addresses the shortcomings of traditional methods, where even minor prediction inaccuracies can lead to suboptimal or infeasible decisions. In essence, DFL shifts the focus from merely minimizing prediction error to enhancing the overall decision quality.

As mentioned above, the calculation of DFL's loss function relies on the optimal solutions obtained from an optimization oracle. This optimization oracle can be viewed as a black box that maps the input optimization parameters to the resulting optimal solutions. An important aspect of this process is that the mapping is discrete, leading to abrupt changes in the objective function of the optimization. Consequently, since the loss function in DFL incorporates the outputs of such non-continuous objective functions, it is inherently non-continuous and, therefore, non-differentiable. This non-differentiability is the primary challenge in DFL research because it prevents the direct computation of the gradients needed to update the parameters of the predictive models. As reviewed in the survey by Mandi et al. (2023), various methods have been developed to address this issue. A common approach is to use smoothing techniques through stochastic perturbations. One promising method in this regard is the Score Function Gradient Estimation (SFGE). The log trick, upon which SFGE is based, provides the necessary flexibility to be applied to problems with uncertainty in both the objective and the constraints. While SFGE has been previously applied to address uncertainty in constraints, its applications have either been on overly simplistic problems (Silvestri et al., 2023) or in settings that deviate from the standard DFL format (van den Houten et al., 2024).

This research aims to extend the application of Decision-Focused Learning to more complex optimization problems, specifically scheduling problems, in the presence of contextual features. The concrete problem selected for evaluating the applicability of DFL is the Resource-Constrained Project Scheduling Problem (RCPSP). The use of this problem is motivated by the additional layer of complexity added by the resource and the precedence constraints, which are prone to violations that can trigger a cascade of further constraint violations. Additionally, RCPSP is widely utilized not only for research purposes (Kolisch and Sprecher, 1997), but also in practice (Klein, 2001).

This new application necessitated more complicated recourse actions, where the process of transforming an infeasible solution into a feasible one often involved multiple adjustments to the original infeasible solution. This added complexity introduced a new challenge: in instances with multiple distinct optimal solutions, which is common in scheduling, these solutions can require different recourse actions. Since the recourse action and the transformed solution are components of the decision loss, different optimal solutions can result in different decision losses. The major consequence of this discovery is a reconsideration of the solving technology's role—it should not be treated merely as a black box that can be easily swapped out. Instead, the choice of solution output by the solver now directly impacts the training process, as different solutions correspond to different losses. Solving this issue completely is beyond the scope of this thesis; however, this work aims to shed light on the problem and justify the decisions made regarding how to advance while taking into account this challenge and the limitations of the thesis.

The contributions of this research can be summarized as follows:

1. Proposing extensions to improve the performance of the currently used SFGE methodology.
2. Applying for the first time the traditional DFL framework with contextual features to a scheduling problem.
3. Conducting an in-depth evaluation of the suitability of DFL and the specific methodology for more complex problems.

By addressing these points, this study aims to advance the application of Decision-Focused Learning in more complex and realistic scheduling scenarios, potentially paving the way for more effective solutions in practical settings.

The remainder of this thesis is organized as follows: firstly, Chapter 2 provides the necessary foundation needed to understand the theoretical intricacies behind the rest of the thesis. This chapter can be divided into two parts: Section 2.1, 2.2 and 2.3 formally introduce the concepts of combinatorial optimization, scheduling, and the causes of uncertainty in scheduling; then, Section 2.4 and 2.5 explain DFL and the use of SFGE in DFL. Next, Chapter 3 formalizes the explicit research goals of the thesis. Then, Chapter 4 explains in detail the concrete implementation of the SFGE-based DFL methodology used in this study to solve the scheduling problem. Furthermore, it puts forward extensions to the standard SFGE method. Chapter 5 presents the experiments that are conducted to answer the research questions and their results. Chapter 6 discusses the discoveries about the inner workings of the solver and how they influence this thesis and the general DFL pipeline. The thesis concludes with Chapter 7, which summarizes the work that has been done, provides suggestions for future research, and outlines a vision for the future of the DFL framework.

2

Background

In this chapter, we discuss the relevant theoretical concepts to provide a solid foundational background for integrating machine learning and combinatorial optimization into a coherent pipeline, specifically addressing challenges posed by uncertainty in constraint parameters. The chapter is divided into two logical parts: first, an exploration of scheduling and its inherent uncertainties, and second, methodologies for addressing uncertainty in scheduling.

The thesis focuses on the scheduling problem, but it is a useful initial step to begin by introducing the general notion of combinatorial optimization and relating it to scheduling, which is done in Section 2.1. Following this, Section 2.2 delves deeper into the specifics of scheduling and touches on the different types of scheduling problems that appear. Consequently, it introduces the scheduling problem that the thesis tries to solve, Resource-Constrained Project Scheduling Problem (RCPSP), by describing the problem along with providing a Constraint Programming (CP) model to solve it. Section 2.3 discusses the important problems that arise when applying combinatorial optimization problems in practice, namely the inherent uncertainty present in the real world, emphasizing how it could impact the modeling of the scheduling problem.

The second logical part begins with Section 2.4, which introduces the more direct method for addressing uncertainty in combinatorial optimization problems, namely Prediction-Focused Learning (PFL). Following this, Section 2.5 presents Decision-Focused Learning (DFL) as a more advanced methodology. It compares its workflow with Prediction-Focused Learning and introduces the concepts of infeasible solutions and penalties. Additionally, it describes the zero-valued gradient problem inherent in DFL and highlights Score Function Gradient Estimation (SFGE) as a potential method to address this issue.

2.1. Combinatorial Optimization

To better explain the focal problem of the thesis, the scheduling problem, this section will take a step back and first introduce the more general concept of combinatorial optimization problems. ”Combinatorial optimization has its roots in combinatorics, operations research, and theoretical computer science” as stated in the book by Korte and Vygen (2012, p.XIII). It is a subfield of computer science and mathematical optimization. Its goal is to find the optimal solution from a finite set of possibilities, where the set of feasible solutions can be reduced to a discrete set. The term ”combinatorial” refers to the fact that these problems often involve selecting a combination of elements. Combinatorial optimization problems can be formulated as:

$$\arg\max_{X} \quad C \cdot X \tag{2.1a}$$

$$\text{s.t.} \quad X \in \mathcal{CS} \tag{2.1b}$$

There are three main elements in Equation 2.1. First, the decision variables, which are denoted by X. Then, the objective function Equation 2.1a, which expresses that the problem tries to find the X such that the combination (which in this example is linear) of X and its coefficients C takes maximum value. Lastly, Equation 2.1b states the constraints, i.e., X must satisfy the constraint set $\mathcal{CS}$.

Some example combinatorial optimization problems besides the scheduling problem are:

1. Traveling Salesman Problem: Given a list of cities and the distances between them, find the shortest possible route (order of cities) that visits each city exactly once and returns to the original city.
2. Knapsack Problem: Given a set of items, each with a weight and a value, determine which items to include so that the total weight is less than or equal to a given limit and the total value is maximized.
3. Graph Coloring: Given a graph, assign colors to its vertices so that no two adjacent vertices have the same color and the number of colors used is minimized.
4. Minimum Spanning Tree: Given a connected, undirected graph with weighted edges, find a tree that spans all the vertices of the graph with the minimum possible total edge weight.

Combinatorial optimization problems find broad use. They are especially important in fields such as Logistics, Supply Chain Optimization, and Operations Research. Combinatorial optimization problems can be solved using exhaustive searching for the best element from the discrete set of possible solutions. However, because the number of combinations often increases exponentially with the number of items, such an approach is computationally infeasible. Therefore, tackling them requires special attention. Dynamic programming can efficiently solve the problems by breaking them down into smaller subproblems and reusing solutions. Linear programming offers a powerful framework for modeling and optimizing combinatorial problems with linear constraints, while constrained programming extends this capability to handle more complex constraints, including nonlinear and discrete variables, thus providing versatile approaches to finding optimal solutions.

2.2. Resource-Constrained Project Scheduling Problem

Now that we have formalized the concept of combinatorial optimizations, we can focus specifically on the scheduling problem. Scheduling, sometimes also referred to as job scheduling, is a class of combinatorial optimization problems. The problem involves a provided set of jobs (also commonly referred to as tasks) and a list of machines (also called processors or workers). The exact settings differ depending on the specific purpose for which the problem is designed, but there are some common elements that often appear. A processing time p_i (commonly referred to as duration) is associated with each task i, which is assumed to be integer and positive. In some variations, tasks are also associated with start times and end times.

The execution of a task is subject to various types of constraints. The most popular constraints considered are temporal constraints and resource constraints. Common types of temporal constraints are availability, due date, and precedence constraints. The existence of a precedence constraint for the pair of tasks (i, j) signifies that the start of the execution of task j must occur after the end of the execution of task i. Resource constraints limit the availability of resources (e.g., machines, workers) required

for executing tasks. In such settings, there are M different resource types, a total available resource $\mathcal{R}^M$, and the specific consumption of each task. The resources could be of two types: renewable and non-renewable. Renewable resources are available on a period-by-period basis, and the total available resource resets at the start of the next period. On the other hand, non-renewable resources represent a single value for the entire duration.

There is also a wide variety of objectives associated with scheduling. The most common one is to minimize the total completion time of all tasks (also known as makespan). Others include: minimize total (weighted) tardiness – minimize the total (weighted) amount by which tasks are completed after their due dates, ensuring that tasks are completed on time or as close to their due dates as possible; minimize total flow time: minimize the sum of the time each task spends in the system, from the moment it enters the system until it is completed. It is also possible to optimize for multiple objectives simultaneously or to have no objective, which just presents the satisfaction version of the problem.

The exact scheduling problem in question is the Single-Mode Resource-Constrained Project Scheduling Problem (RCPSP) as described in (Kolisch and Sprecher, 1997). The objective is to minimize the makespan in the presence of precedence and renewable resource constraints, while being given only the processing times of the tasks. A suitable method for solving such a problem is to use Constraint Programming (CP). In constraint programming, users declaratively state the constraints on the feasible solutions for a set of decision variables, allowing the definition of complex problems in a clear and concise way. Additionally, CP allows to efficiently navigate the search space for solutions through domain reduction and constraint propagation. A CP model for RCPSP can be compiled as shown in the work by van den Houten et al. (2024): given J, a set of jobs, R, a set of resources, and S, a set of successors, with n_{rj} indicating the resource need of type r for task j, y_j denoting the processing time of task j, b_r, the cumulative maximum resource of type r available at each iteration, and S_j, the set of successors of task j, find the schedule with the minimum total makespan. The decision variable of the model is x_j, which indicates the interval length of task j. The CP model (using IBM CPLEX (Cplex, 2009) function notation) is given by:

$$\arg\min_{x} \quad Makespan \tag{2.2a}$$

$$\text{s.t.} \quad \max\left(\text{endOf}\left(\text{x}_\text{j}\right)\right) \leq \text{ Makespan} \qquad j \in J, \tag{2.2b}$$

$$\text{startOf}\left(\text{x}_\text{i}\right) \geq \text{endOf}\left(\text{x}_\text{j}\right) \qquad \forall j \in J, \forall i \in S_j, \tag{2.2c}$$

$$\sum_{j \in J} \text{Pulse}\left(\text{x}_\text{j}, \text{n}_\text{rj}\right) \leq \text{b}_\text{r} \qquad \forall r \in R, \tag{2.2d}$$

$$x_j : \text{IntervalVar}\left(\text{J}, \text{y}_\text{j}\right) \qquad \forall j \in J \tag{2.2e}$$

Equation 2.2a gives the objective of the optimization problem – to minimize the total makespan. Constraint 2.2b defines the makespan as the end time of the last job. Constraint 2.2c enforces the precedence relation by requiring each successor of task j to have a starting time greater than or equal to the ending time of j. The shared resource usage is modeled using the pulse CP constraint (Cplex, 2009) in 2.2d. The pulse function returns a cumulative expression equal to a certain value over an interval. In this case, the value is the specific resource usage n_{rj} of task j, and the interval is the interval decision variable x_j. The pulse values are then aggregated and restricted to have a sum less than or equal to the total available resource b_r.

2.3. Uncertainty in Scheduling

The provided CP model is generally correct but could be effectively applied on its own only in a deterministic environment, i.e., where all of the factors are perfectly known beforehand, and the process

always unfolds as intended. However, this scenario is rarely the case in practice, where uncertainty emerges due to the inherent complexity and dynamic nature of real-world environments. For instance, when dealing with a manufacturing facility, it is possible that a task takes longer because of human error, machine malfunction, or just because it includes a process which is inherently stochastic, such as a complex biochemical reaction. A change in the processing time of a task could easily alter the optimal schedule that the CP model would have produced if it had known the actual value beforehand. Moreover, in the presence of tight constraints, such a change could render the remainder of the schedule infeasible, leading to negative consequences for the production, ranging from just minor disruptions to severe impacts such as system failure, customer dissatisfaction, and loss of revenue (Sarin et al., 2010).

Before turning our heads to methods dealing with uncertainty, it is useful to quickly introduce the two types of uncertainty that occur – epistemic uncertainty, derived from a lack of knowledge about a parameter, and aleatory uncertainty, referring to uncertainty caused by probabilistic variants in a random event. To build an intuition about them, consider an example from meteorology. Aleatory uncertainty could arise from the inherent randomness or variability in atmospheric processes, such as the unpredictable movement of individual raindrops despite perfect knowledge of current conditions. On the other hand, epistemic uncertainty emerges from gaps in knowledge or understanding, such as incomplete data or insufficient comprehension of complex atmospheric dynamics. For instance, if there's a new phenomenon in atmospheric science that hasn't been fully understood or incorporated into forecasting models. Since the CP model is provided only with a single value for the processing time of the tasks, it inherently favors parameters predominantly characterized by epistemic uncertainty. Epistemic uncertainty suggests predictability to a single value when information is available. Conversely, uncertain parameters with substantial aleatory uncertainty may yield greater insights through learning the underlying distribution of the parameters.

2.4. Prediction-Focused Learning

The presence of uncertain parameters when applying combinatorial optimization problems in the real world is not a newly discovered phenomenon. Uncertain parameters can be dealt with via different approaches. Robust optimization (Ben-Tal et al., 2009) aims to minimize the adverse effects of uncertainty by ensuring that solutions perform well across different realizations of uncertain events. It does so by considering the worst-case scenarios or optimizing against a range of possible outcomes. A different alternative is stochastic optimization (Nemhauser et al., 1989) which formulates optimization problems with probabilistic constraints or objectives and seeks decision rules that are optimal or satisfactory under various probabilistic scenarios.

Both robust and stochastic optimization rely on historical realizations of the unknown parameters. However, this work adopts a different approach, which assumes that the provided information is in the form of contextual features from which uncertain parameter values can be predicted. The survey by Sadana et al. (2023) provides a broad overview of the different methods that can be used to solve such contextual optimization problems. A common and direct approach for handling this uncertainty is to employ machine learning algorithms to estimate the uncertain quantities. The general pipeline of this approach is depicted on the right in Figure 2.1. Here, the model makes predictions based on contextual features, which are then fed into the CP solver to optimize the objective and compute the optimal solution. Traditionally, these two steps, prediction and optimization, were treated as independent procedures. This approach is known as Prediction-Focused Learning (PFL), also referred to in the literature as a "two-stage" approach (Mandi et al., 2023).

The fundamental distinction between PFL and Decision-Focused Learning (DFL) lies in the training routine, as illustrated on the left in Figure 2.1. PFL focuses solely on predictive error, striving to make predictions as close as possible to the actual true values of the uncertain parameters. Therefore, the most commonly used loss function in the PFL setting is the mean squared error (MSE):

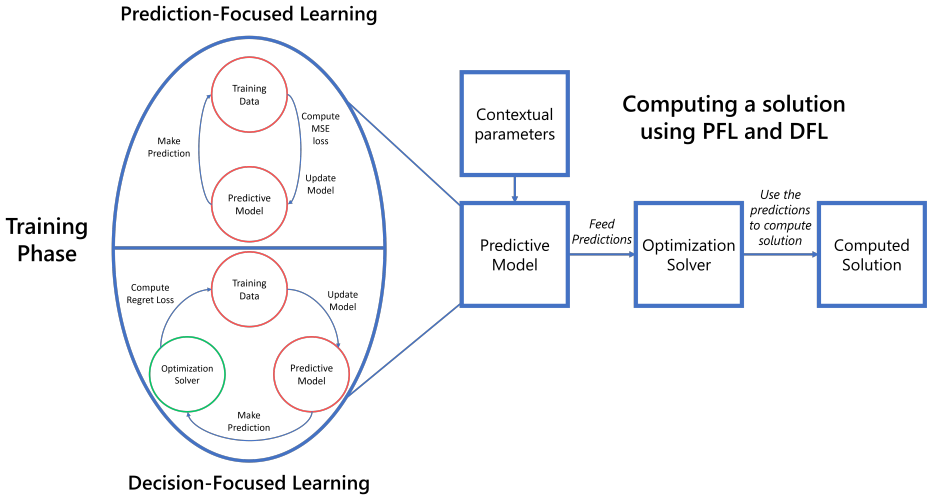


Figure 2.1: PFL and DFL pipeline. On the left side, the distinction in the training phase between PFL (top) and DFL (bottom) is illustrated. DFL incorporates an additional component, the Optimization Solver (highlighted in the green circle), which is not present in PFL. This solver is utilized in DFL to compute its regret loss. On the right side, the shared pipeline of PFL and DFL is visualized. This common pipeline outlines the standard procedure followed in both learning approaches, highlighting the integration of machine learning models with optimization solvers to achieve comprehensive decision-making capabilities.

$$\text{MSE}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N}$$

where y denotes the true value of the parameter and $\hat{y}$ denotes the predicted one. Once the MSE is computed, it is directly used to update the model. Essentially, PFL follows the conventional approach for training a machine learning model, involving the interaction between two primary components: the data and the model (as illustrated at the top left of Figure 2.1). Consider machine learning methods that use gradient-based optimization techniques as an example. In these processes, the model utilizes the data to make predictions, which are then employed to update the model. This cycle of prediction and model updating continues iteratively until the process converges or is stopped.

2.5. Decision-Focused Learning

The goal of Decision-Focused Learning (DFL) is to address the main weakness of PFL, namely, the lack of integration between machine learning training and the quality of the produced decision. As decisions are realized after the optimization stage, this requires the integration of the optimization component into the training loop (as depicted in the bottom left of Figure 2.1), resulting in a composite model that produces comprehensive decisions. This is achieved by incorporating the quality of the decision into the loss function of the model. The fundamental concept underpinning DFL is exactly the formulation of this novel loss function called *regret*, designed to evaluate decision quality. Regret quantifies the disparity between the objective value obtained from solving the combinatorial optimization problem using parameter values predicted by the machine learning model, $\hat{y}$, and the objective value derived from solving the optimization problem without uncertainty, i.e., using the true parameter values, y:

$$\text{Regret}(\hat{y}, y) = f(x^*(\hat{y})) - f(x^*(y))$$

where $x^*(y)$ denotes the optimal solution with the true parameters y, $x^*(\hat{y})$ denotes the optimal solution with the predicted parameters $\hat{y}$, and $f(\cdot)$ denotes the objective value of a solution.

A significant shortcoming of regret is that it can deal with problems that have uncertainty only in the objective function. If uncertainty appears in the constraints, it is possible that the solution calculated with the estimated parameters is infeasible under the true parameters. To generalize it for problems where uncertainty may also appear in the constraints, Hu et al. (2022) propose an extension of regret called *post-hoc regret*, formulated as:

$$\text{Post-Hoc Regret}(y, \hat{y}) = f(x^*(\hat{y})) - f(x^*(y)) + \text{Pen}(x^*(\hat{y}) \to x^*_{\text{corr}}(\hat{y}, y))$$

In the formula above, $x^*_{\text{corr}}(\hat{y}, y)$ represents the correction function that transforms $\hat{y}$ into a feasible solution given the true parameters y. The $\text{Pen}(\cdot)$ function serves as a penalty function, assigning a non-negative penalty value to the transformation from the estimated solution to the corrected one. The use of correction and penalty functions offers a significant advantage in practical applications by allowing infeasible solutions to be transformed into feasible ones when the true parameters are revealed, rather than discarding them outright. The specific choice of these functions depends on the particular problem. In the context of scheduling with uncertainty, as discussed in Section 2.3, the correction function adjusts the schedule based on the actual processing times of the tasks. As noted earlier, the consequences of these adjustments can vary widely. Therefore, the penalty function can be customized to assign higher penalties to more impactful changes, thereby increasing post-hoc regret proportionally.

A notable drawback of DFL is its reliance on solvers to compute regret. This dependency results in significant computational overhead, as regret calculations are required for each example and during each epoch of the training process. Moreover, the computational cost escalates with the complexity of the optimization problem, which typically grows exponentially for most combinatorial optimization problems. While DFL excels in generating optimal decisions, its reliance on solving backends highlights a limitation that should be carefully considered when implementing the framework.

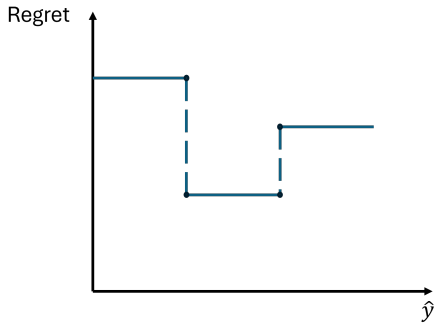


Figure 2.2: Regret as a function of $\hat{y}$. The graph shows that regret is a step function and is thus non-differentiable.

The primary challenge in DFL arises from the non-differentiability of its regret loss function, which prevents the application of gradient-based learning methods. This issue originates from the regret definition itself, which depends on the outcome of a combinatorial optimization procedure, which entails a discrete mapping. Figure 2.2 visually depicts the regret as a function of $\hat{y}$, illustrating its characteristic abrupt changes. This figure provides insight into the zero-gradient problem, highlighting that small changes in the optimization input $\hat{y}$ do not necessarily result in a new optimal solution with a different objective value. The scheduling problem can serve as a good example: most small changes in the processing times $\hat{y}$ lead to the same solution, with only a few points causing non-continuous, step-like changes. Consequently, the entire loss function may exhibit undefined or zero-valued gradients with respect to the predictive model's parameters. This limitation also becomes evident when deriving the gradient:

$$\frac{\partial \mathcal{L}\left(x^\star(\hat{y}), y\right)}{\partial \omega} = \frac{\partial \mathcal{L}\left(x^\star(\hat{y}), y\right)}{\partial x^\star(\hat{y})} \cdot \frac{\partial x^\star(\hat{y})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \omega}$$

Here, ω represents the parameters of the predictive model. The term, $\frac{\partial x^\star(\hat{y})}{\partial \hat{y}}$, measures the sensitivity of the optimal solution $x^\star(\hat{y})$ to changes in $\hat{y}$. In combinatorial problems like scheduling, this term is nearly zero across most points, as depicted in Figure 2.2. Consequently, the gradient is effectively zero across most points and undefined at the points where the jumps occur, rendering it uninformative and unsuitable for gradient-based learning.

To address this challenge, various methods have been proposed. Mandi et al. (2023) provides a comprehensive survey categorizing these methods into four groups based on how they handle the differentiation issue: 1) analytical differentiation of optimization mappings, 2) analytical smoothing of optimization mappings, 3) smoothing using random perturbations, and 4) differentiation of surrogate loss functions. One notable method from the third category is Score Function Gradient Estimation (SFGE), also known as the REINFORCE algorithm in the context of reinforcement learning (Sutton et al., 1999). SFGE finds applications in reinforcement learning and other areas requiring gradient-based optimization where direct computation of gradients is not feasible. Recently, SFGE has been adapted to DFL in the works of Silvestri et al. (2023) and van den Houten et al. (2024). SFGE estimates gradients using Monte Carlo sampling and the log-derivative trick. The method provides a high level of flexibility because it does not make any assumptions about the structure of the problem or the loss function used. As a consequence, it is compatible with post-hoc regret, which makes it promising for addressing problems involving uncertainty in constraints. If successful in scheduling, SFGE could be easily applied to other optimization problems with minimal modifications. Following the recommendations from the survey and after reviewing the works that adapt it for DFL, we decided to base our approach on SFGE. Further details about the strengths of SFGE and our specific implementation are discussed in Section 4.2.

3

Research Questions: Formulation and Motivation

This research aims to extend the Decision-Focused Learning (DFL) framework to address more complex optimization problems, specifically focusing on scheduling problems. To assess the suitability of DFL for such problems, it is essential to first design an effective method to deal with them. As foreshadowed in Section 2.5, we have chosen to base our DFL method on the Score Function Gradient Estimation (SFGE) approach. Given that SFGE is a relatively new and unexplored method, the first two research questions (Section 3.1 and Section 3.2) focus on investigating potential improvements to SFGE. After refining the method, the next two research questions take a step back and shift the focus to the general DFL framework itself, evaluating the impact of environmental factors – specifically, the penalty added for infeasible solutions (Section 3.3) and the level of uncertainty (Section 3.4). This evaluation aims to identify the strengths and weaknesses of the DFL framework and to determine which problems are most suitable for it.

3.1. Research Question 1: Impact of the Sampling Distribution

Formulation: How does the choice of sampling distribution in the Score Function Gradient Estimation method impact the training performance of the overall algorithm?

Motivation: SFGE-based DFL methods have shown to be a prominent solution for the problems with uncertainty in the constraints. However, both the works of Silvestri et al. (2023) and van den Houten et al. (2024) directly assume Normal distribution because of its popularity in stochastic policy gradient without considering alternatives. This question aims to investigate the impact of the sampling distribution more thoroughly. Specifically, it will try to derive which is the optimal configuration when using Normal distribution in the scheduling setting, i.e., using a single standard deviation for all jobs or separate ones. Additionally, it will assess how alternative options such as Beta, Poisson, and Half-Normal distributions match up.

3.2. Research Question 2: RL-Inspired Extensions

Formulation: Would the addition of reinforcement learning-based techniques improve the convergence speed of the Score Function Gradient Estimation method when applied to DFL?

Sub-question 1: Would the incorporation of a baseline term in the post-hoc regret improve the convergence of the SFGE algorithm?

Motivation: In reinforcement learning, adding a baseline term to the loss function is a standard practice in the REINFORCE algorithm to reduce variance in gradient estimates (Sutton and Barto, 2018; Mohamed et al., 2019). However, currently, there is no method that applies this extension in the DFL context. This sub-question aims to analyze whether such an improvement would also be beneficial in the context of DFL.

Sub-question 2: Would the integration of Proximal Policy Optimization updates improve the convergence of the SFGE algorithm?

Motivation: The traditional REINFORCE algorithm is often criticized for its sample inefficiency, making it computationally demanding, particularly in high complexity search spaces. This inefficiency would be a significant bottleneck in the DFL pipeline, especially due to the computational time needed by the solver. Employing a PPO-inspired approach could address this issue by facilitating multiple model updates with a single solver call, thereby potentially enhancing the algorithm's convergence speed and overall efficiency in the context of DFL.

3.3. Research Question 3: Impact of the Added Penalty

Formulation: What is the impact of the penalty incurred due to constraint violations on the Decision-Focused Learning algorithm?

Sub-question 1: Does there exist a difference in the performance of Decision-Focused Learning relative to Prediction-Focused Learning for the different penalty settings?

Motivation: Previous research (van den Houten et al., 2024) has indicated that the amount of added penalty influences the difference in normalized regret between PFL and DFL. This sub-question aims to explore whether this trend holds true when using contextual features instead of historical realizations.

Sub-question 2: What is the effect of the penalty setting on the structure of the predictions made by Decision-Focused Learning?

Motivation: If there is a difference between the predictions made by PFL and DFL, is it because DFL can learn a structure within the problem? If so, what specific structure does DFL learn, and how does the penalty setting influence this learning process?

3.4. Research Question 4: Effect of Uncertainty

Formulation: How does uncertainty affect the performance of the Decision-Focused Learning method for scheduling with uncertain processing times?

Motivation: As discussed in Section 2.3, the scheduling problem involves two types of uncertainty: epistemic and aleatory. These uncertainties vary across different scheduling environments. The objective of this research question is to investigate the extent to which the algorithm is effectively applicable in environments characterized by each type of uncertainty, aiming to determine in which environments the algorithm performs better.

4

Methodology

The goal of this study is to assess the applicability of Decision-Focused Learning in addressing scheduling problems, specifically RCPSP. This chapter provides a detailed explanation of the method designed to achieve this goal. The method is tailored to fit within the classical DFL framework with contextual features. Given these features, the objective is to train a DFL model capable of predicting uncertain processing times while minimizing the decision error. Notably, addressing uncertainty in problem constraints is crucial in method design.

This chapter is structured as follows: first, Section 4.1 gives a high-level overview of the proposed DFL method. Subsequently, the next four sections (Section 4.2 - Section 4.5) each detail one of the four core stages of the method. These sections follow a consistent structure: introduction of input, description of output, and specifics of the transformation within each stage. Additionally, Section 4.2 includes an explanation of the Score Function method. The chapter concludes with Section 4.6, which presents the novel extensions proposed for the general methodology of applying SFGE in DFL.

4.1. Method Overview

The problem addressed involves scheduling with uncertainty in the constraints. As discussed in Section 2.5, managing uncertainty in the constraints requires special attention and the development of a new loss function to handle potential infeasible solutions due to constraint violations. Post-hoc regret is particularly well-suited for this task. However, it exhibits the zero-gradient problem. To address this challenge, Score Function Gradient Estimation emerges as a promising solution. SFGE is suitable due to its inherent flexibility – it does not impose specific assumptions about problem structures (e.g., convex or linear problems) and operates independently of the solver used. Moreover, SFGE can accommodate various loss functions, including post-hoc regret.

A schematic of the workflow for the proposed DFL training routine is depicted in Figure 4.1. This workflow is structured into four stages: prediction, solving, correction, and update. The underlying machine learning model is linear regression, trained through gradient-based model updates. The presence of prediction and update stages is not unique to our approach but fundamental to all gradient-based methods. In the prediction stage, the input is the observed contextual features of the training example, and the goal is to arrive at predicted values for the uncertain processing times. Rather than directly using the output of the predictive model, this stage employs SFGE, discussed further in Section 4.2.

Consistent with DFL principles, the method integrates an optimization component into the training

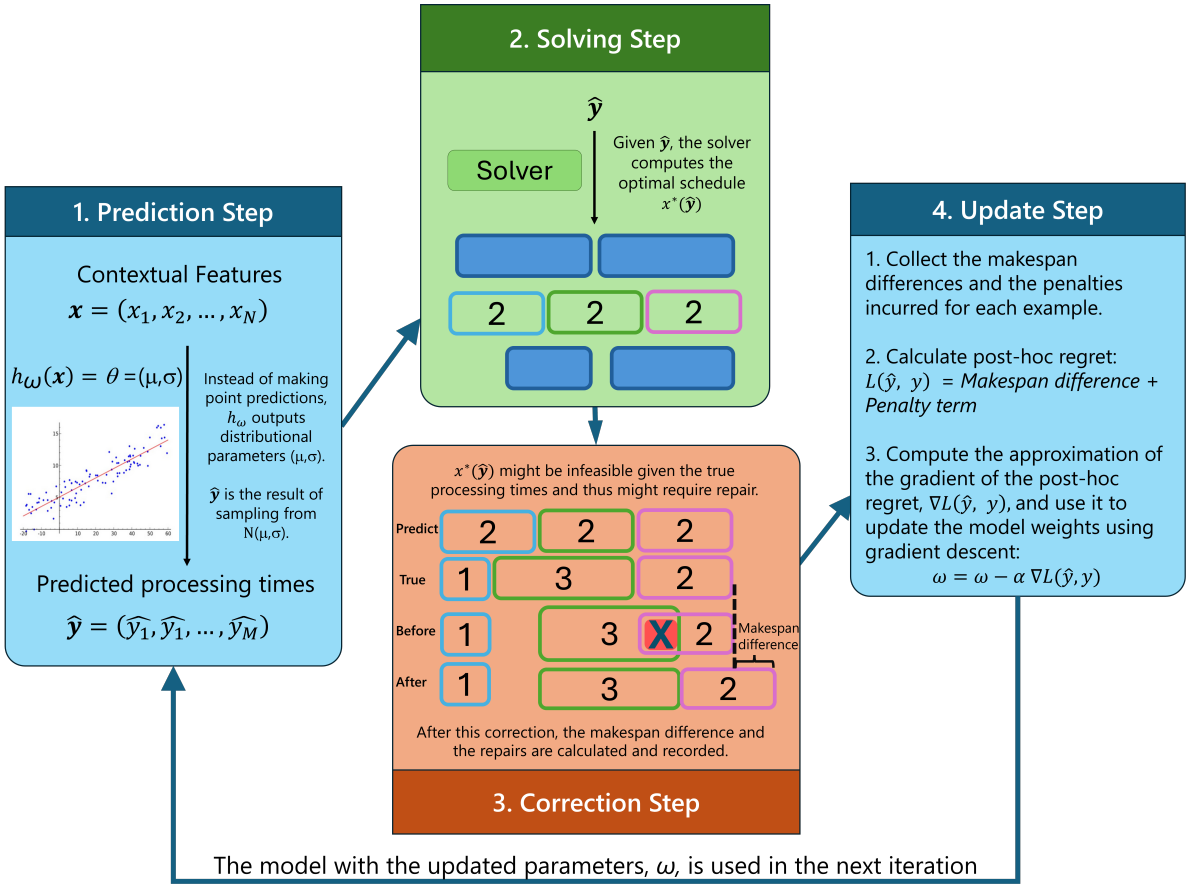


Figure 4.1: Workflow of the SFGE method. It has four steps: prediction, solving, correction, and update, which continuously interact with each other during training. *Note:* The chart shows that the prediction step uses Normal distribution for the sampling. However, this is not a hard requirement and is used here for illustrative purposes. The optimal choice for the sampling distribution is subject to evaluation and is discussed in Section 5.2.

process. This optimization occurs during the solving stage, where the predicted processing times are used to compute a deterministic schedule. Due to potential constraint violations, schedules may require correction. Hence, the correction stage ensures that the schedule aligns with the constraints of the optimization problem, producing feasible solutions. The last remaining stage is the update stage. It consolidates all components of the loss and performs gradient descent. Once the model gets updated, all of the steps repeat iteratively.

4.2. Prediction Step and the Score Function Method

Input: Contextual features

Output: Predicted deterministic processing times for each task

Workflow: The prediction step serves as the initial stage of the algorithm, where the predictive model plays the central role. In this study, linear regression has been chosen as the predictive model. This selection is motivated by the need to operate in an environment where uncertain processing times cannot be perfectly predicted. If perfect predictions were achievable, DFL would be redundant, as the problem would reduce to a standard deterministic problem. Moreover, the inability to achieve perfect predictions more accurately mirrors real-world conditions where noise is always present. A lower-complexity model like linear regression accommodates some prediction inaccuracies. This setup enables the evaluation

of the framework's objective: assessing whether DFL makes predictions that result in sound decisions, rather than focusing on developing a complex model that is able to perfectly fit the specific training data.

Figure 4.2: Stochastic smoothing applied to the non-continuous regret loss. The left side shows regret as a function of $\hat{y}$, which is non-differentiable. The right side illustrates the result of the stochastic smoothing transformation. The blue distributions depict stochastic estimators for $\hat{y}$, while the expected regret is represented by the smoothed red curve.

The prediction step can be viewed as a black box that takes the contextual features as input and outputs predictions for the uncertain processing times $\hat{y}$. What this black box hides is the fact that the predictions are not just the output of the predictive model. Instead, the method incorporates SFGE as an intermediary layer that transforms the model output into predicted values. Starting from the beginning, contextual features are fed into the predictive model. Rather than generating point predictions, the model predicts distributional parameters, which initialize a probability distribution. The actual output of the whole predictive step (predicted processing times) is obtained by sampling from this distribution. By using samples instead of the point predictions, the approach performs stochastic smoothing to the non-continuous post-hoc regret, as illustrated in Figure 4.2.

Since the predictions are stochastic during training, the loss function becomes an expectation (Equation 4.1a). This approach aims to ensure that by predicting distributions, the gradient of the loss is non-zero. Computing the gradient of this expectation is non-trivial, and this is where Score Function becomes instrumental. SFGE estimates Equation 4.1a by:

$$\nabla_\theta L(\hat{y}, y) = \nabla_\theta \mathbb{E}_{\hat{y}\sim p_\theta(y)}[\mathcal{L}(\hat{y}, y)] \tag{4.1a}$$
$$= \nabla_\theta \int p_\theta(y)\mathcal{L}(\hat{y}, y)d\hat{y} \tag{4.1b}$$
$$= \int \mathcal{L}(\hat{y}, y)\nabla_\theta p_\theta(\hat{y})d\hat{y} \tag{4.1c}$$
$$= \int \mathcal{L}(\hat{y}, y)p_\theta(\hat{y})\nabla_\theta \log p_\theta(\hat{y})d\hat{y} \tag{4.1d}$$
$$= \mathbb{E}_{\hat{y}\sim p_\theta(y)}\left[\mathcal{L}(\hat{y}, y)\nabla_\theta \log p_\theta(\hat{y})\right] \tag{4.1e}$$
$$\approx \frac{1}{S}\sum_{i=1}^{S} \mathcal{L}\left(\hat{y}^{(i)}, y\right)\nabla_\theta \log p_\theta\left(\hat{y}^{(i)}\right) \tag{4.1f}$$

Here, Equation 4.1b is the result of applying the definition of expected value for continuous random variables. In Equation 4.1c, the gradient is brought inside the integral. This operation is valid because of the Leibniz rule (L'Ecuyer, 1995; Mohamed et al., 2019). The log-derivative trick is applied in Equation 4.1d: first, we apply the chain rule to differentiate the logarithm, $\nabla_\theta \log p_\theta(\hat{y}) = \frac{\nabla_\theta p_\theta(\hat{y})}{p_\theta(\hat{y})}$. Then, we rearrange the terms to get $\nabla_\theta p_\theta(\hat{y}) = p_\theta(\hat{y})\nabla_\theta \log p_\theta(\hat{y})$. Monte Carlo sampling is then used in Equation 4.1e to estimate the gradient, resulting in Equation 4.1f. This estimate of the gradient can now effectively be used in training. Note that it is not specific to any loss function and thus the method can be easily combined with different loss functions, including post-hoc regret.

4.3. Solving Step

Input: Predicted processing times for each task

Output: Optimal schedule given the predicted processing times

Workflow: The solving step integrates the solver into the training loop. This step is straightforward: it takes the predicted processing times from the prediction step and inputs them into a deterministic optimization solver, which uses an underlying Constraint Programming model (as described in Section 2.2). The output is the optimal schedule produced by the solver. Notably, this stage is the most computationally expensive part of the training loop, as depicted in Figure 4.3. This is due to the NP-Hard nature of the scheduling problem and the requirement to solve a scheduling problem for each training example.

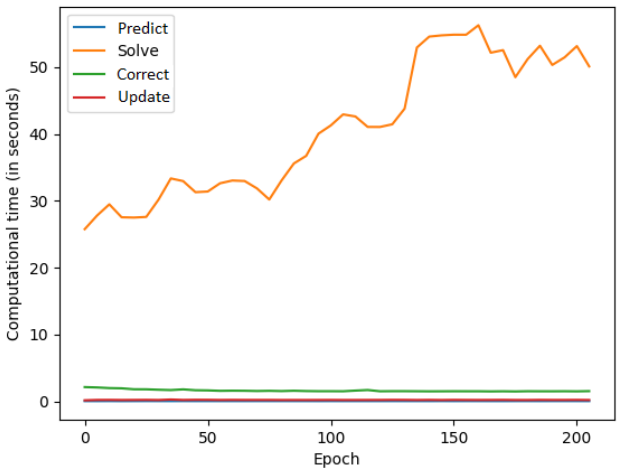


Figure 4.3: Time taken by each of the four steps during the training routine. The solving step (in orange) takes significantly more time compared to the other steps.

4.4. Correction Step

Input: Optimal schedule for the predicted processing times

Output: The components of the post-hoc regret – makespan difference and penalty term

Workflow: The main objective of this step is to repair the potentially infeasible solutions in order to compute the post-hoc regret. It takes as input the optimal solution for the predicted processing times computed by the solver in the solving step and outputs a feasible version of the initially proposed schedule. Each correction procedure, regardless of the problem, follows a consistent outline:

1. The true values for the uncertain parameters are revealed.
2. A draft solution is created, which follows the decisions made in the optimal solution for the predicted values but uses the true values.
3. This draft solution is then cleared of all infeasibilities using the repair method.

As discussed in Section 2.5, the repair method is problem-specific. For our scheduling problem, the

repair works as follows: each task is initially attempted to be started at the earliest time calculated by the optimal solution. Tasks that cannot start without violating constraints under the true processing times are shifted to a later time until all jobs can start without constraint violations. The process then repeats until all jobs can be started.

A simple example of the correction process is visualized in the correction step of Figure 4.1. Focus on the three numbered tasks in blue, green, and pink. Assume these tasks follow a precedence constraint and that only this order of execution is correct. The prediction step predicted processing times of 2 for all tasks, leading the solver to start them consecutively at times 0, 2, and 4, respectively. This schedule is labeled as "Predict." During the correction step, the true processing times are revealed (labeled as "True"). The "Before" schedule presents the schedule that uses the starting times computed by the solver in the "Predict" schedule but applies them to the true values. This version has a collision between the green and pink tasks due to the green task taking longer than predicted. To repair it, the pink task is delayed to start after the green task finishes. Note that no repair is needed for the blue and the green tasks. The "After" schedule is the corrected version of the "Before" schedule, where all infeasibilities have been resolved. Despite the true processing times having the same total of 6 as the predicted values, the overlapping task required a repair, resulting in the "After" schedule having a larger makespan than the "True" schedule. The difference between these two makespans is later referred to as makespan difference.

4.5. Update Step

Input: The components of the post-hoc regret – makespan difference and penalty term

Output: Updated version of the predictive model

Workflow: The goal of this stage is to update the weights of the machine learning model. To achieve this, we first calculate the loss, which in our case is the post-hoc regret. As described in Section 2.5, post-hoc regret is defined as:

$$phr(y, \hat{y}) = \underbrace{f(x^*(\hat{y})) - f(x^*(y))}_{\text{Makespan difference}} + \lambda \underbrace{\text{Pen}(x^*(\hat{y}) \to x^*_{corr}(\hat{y}, y))}_{\text{Penalty from the repairs}}$$

The first component of the loss conveys the difference between the true optimal objective value and the corrected objective value. It is also referred to as the makespan difference in the previous scheduling example. The second component is the penalty term, which increases by one for each move a task makes while fixing the infeasibility. The λ coefficient in the penalty term is the penalty factor, which models the impact of the correction. A higher penalty factor implies a higher cost for the correction, making the penalty term more significant in the post-hoc regret.

After computing the post-hoc regret, SFGE can be employed as described in Section 4.2 to approximate the gradient of the loss. Once the gradient is computed, it can be used to update the model parameters using gradient descent, represented by $\omega = \omega - \alpha \cdot \nabla L(\hat{y}, y)$. Various types of gradient descent exist, and the one used in this study is mini-batch gradient descent. After updating the weights of the predictive model, the training loop returns to the prediction step and repeats until all epochs are completed or the process times out. The pseudocode of the algorithm is presented in Algorithm 1.

Algorithm 1 DFL with SFGE Training

for *epoch* $= 1, 2, \ldots$ **do**
  for batch $\in$ Training Set **do**
    Predict the distributional parameters μ and σ from the contextual features ▷ Prediction step
    Sample the predicted processing times $\hat{y} \sim N(\mu, \sigma)$
    Compute the log probability of $\hat{y}$, $\log p_{\mu,\sigma}(\hat{y})$
    Determine $x^*(\hat{y})$, the optimal solution using $\hat{y}$ ▷ Solving step
    Correct potential infeasibilities in $x^*(\hat{y})$ and calculate the penalty ▷ Correction step
    Compute the post-hoc regret, $PHR(y, \hat{y}) = f(x^*(\hat{y})) - f(x^*(y)) + \lambda \cdot \text{Pen}(x^*(\hat{y}) \rightarrow x^*_{corr}(\hat{y}, y))$
    $\mathcal{L} = PHR \times \log p_{\mu,\sigma}(\hat{y})$
    Update the parameters of the predictive model using $\nabla\mathcal{L}$ ▷ Update step
  end for
end for

4.6. Extensions to the Base SFGE Methodology

The application of SFGE within the DFL framework is a novel technique, having been investigated only in the works of Silvestri et al. (2023) and van den Houten et al. (2024). As a result, there remains significant potential for further refinement and enhancement of this method. A notable issue with SFGE is the high variance in the gradient estimates, which can destabilize the learning process (Mandi et al., 2023). In the course of our research, we identified two promising extensions to improve SFGE. The first extension aims to reduce the variance in gradient estimates by introducing a baseline term into the loss function, further detailed in Subsection 4.6.1. The second extension draws inspiration from Proximal Policy Optimization (PPO) (Schulman et al., 2017), aiming to enhance computational efficiency and training stability by employing multiple updates per sample, as later discussed in Subsection 4.6.2.

4.6.1. Baseline Post-Hoc Regret

The gradients computed using SFGE are Monte Carlo estimates rather than exact values, leading to potentially high variance and instability in the learning process. When applying SFGE in reinforcement learning, specifically the REINFORCE algorithm, it is common practice to introduce a baseline term to reduce this variance (Sutton and Barto, 2018). The baseline should ideally be a reliable estimate of the expected return for a given example. By subtracting this baseline from the loss, the variance in gradient estimates can be reduced, resulting in more stable and efficient learning. This technique helps to focus updates on the discrepancy between observed and expected returns, rather than the observed returns alone. In reinforcement learning, this concept is often referred to as advantage.

The baseline is a value that does not depend on the current prediction and is subtracted from the loss prior to gradient computation. In reinforcement learning, the state-value function often serves as the baseline. However, as there is no state-value function in our context, we propose an alternative approach that follows a similar principle. This approach involves recording the loss for each example in the training set over past epochs to build a loss history. The mean of this loss history serves as the baseline. This method aims to determine if the performance on the example is improving relative to its past performance, i.e., whether the advantage is positive, indicating better predictions. This method introduces a hyperparameter, which determines the number of historical steps to consider. For instance, setting this parameter to 10 means that only the last 10 loss values are used in the baseline calculation. Another variation could involve using a weighted average, where more recent samples have greater influence as they more accurately reflect the current model state.

This extension is incorporated into the initial part of the revised update step, as depicted in Figure 4.4. Instead of using the raw post-hoc regret, we use its baseline-adjusted version. The baseline post-hoc regret (BPHR) is calculated by subtracting the baseline value from the computed post-hoc regret. It is

defined as:

$$\text{Baseline Post-Hoc Regret}(y, \hat{y}) = \text{Post-Hoc Regret}(y, \hat{y}) - \text{mean}(\text{Post-Hoc Regret History}(x_{\text{true}}))$$

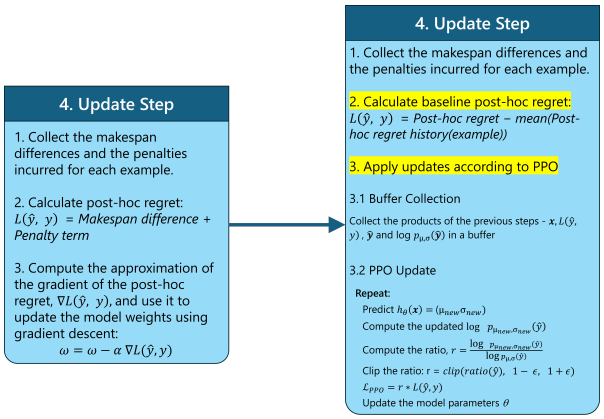


Figure 4.4: Extended version of DFL SFGE. The chart highlights the modifications necessary to implement the two proposed extensions.

4.6.2. Proximal Policy Optimization Updates

The second extension also involves a modification in the update step. Rather than employing the conventional gradient descent approach, we opt for Proximal Policy Optimization (PPO) updates. PPO (Schulman et al., 2017) is a well-known reinforcement learning algorithm designed to mitigate some drawbacks of traditional policy gradient methods, such as slow convergence, high variance, and instability. The fundamental idea behind PPO is to collect multiple batches of experiences and perform several optimization epochs on each batch. This approach is particularly relevant for our DFL framework, where solving each optimization problem constitutes a significant computational burden (as shown in Figure 4.3), especially for complex problems like scheduling. By leveraging PPO, we can execute multiple model updates per training example while only requiring a single solver call and correction step, potentially reducing training time and improving convergence speed.

One challenge that arises from performing multiple updates on a single sample is the risk of large, destabilizing updates to the model. PPO addresses this by employing a clipped surrogate objective function that bounds the update magnitude within a controlled range. The clipped objective is defined as:

$$L^{PPO}(\hat{y}) = \hat{\mathbb{E}}\left[\min\left(ratio(\hat{y}), \text{clip}\left(ratio(\hat{y}), 1-\epsilon, 1+\epsilon\right)\right) \cdot BPHR(y, \hat{y})\right]$$

where:

$$ratio(\hat{y}) = \frac{\log p_{\theta_{\text{new}}}(\hat{y})}{\log p_{\theta_{\text{old}}}(\hat{y})}$$

Here, $p_{\theta_{\text{new}}}(\hat{y})$ and $p_{\theta_{\text{old}}}(\hat{y})$ denote the likelihoods of the example given the new and old sampling distributions predicted by the updated and initial model states, respectively. The ratio $ratio(\hat{y})$ measures the change in policy likelihoods and regulates a smooth update to the policy parameters. The clipping operation ensures that the ratio is kept within the interval $[1-\epsilon, 1+\epsilon]$, so that the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective, thus maintaining stability during training.

The implementation of the PPO update is integrated into our algorithm as depicted in Algorithm 2. Initially, all necessary components ($x_{\text{true}}, \hat{y}, \log p_{\mu,\sigma}(\hat{y}), BPHR$) are computed as in the standard SFGE algorithm. These components are then collected into a buffer $\mathcal{B}$, which is subsequently used to perform multiple epochs of PPO updates. During each buffer epoch, the updated model predicts new distributional parameters μ_{new} and σ_{new}. The updated log probability $\log p_{\mu_{\text{new}},\sigma_{\text{new}}}(\hat{y})$ is computed and then used to calculate the ratio r. Then, the ratio and baseline-adjusted post-hoc regret $BPHR$ are used to obtain the PPO loss $\mathcal{L}$. Finally, the model parameters are updated using the gradient of $\mathcal{L}$.

Algorithm 2 DFL SFGE with PPO Updates and Baseline Post-Hoc Regret

for epoch $= 1, 2, \ldots$ **do**
  for training example $= 1, 2, \ldots, N$ **do**
    Predict distributional parameters μ and σ from true labels ▷ Prediction step
    Sample predicted processing times $\hat{y} \sim N(\mu, \sigma)$
    Compute log probability of $\hat{y}$, $\log p_{\mu,\sigma}(\hat{y})$
    Determine $x^*(\hat{y})$, optimal solution using $\hat{y}$ ▷ Solving step
    Correct infeasibilities in $x^*(\hat{y})$ ▷ Correction step
    Compute baseline post-hoc regret $BPHR$ ▷ Update step - Extended version
    Collect $x_{\text{true}}, \hat{y}, \log p_{\mu,\sigma}(\hat{y}), BPHR$ into buffer $\mathcal{B}$
  end for
  for buffer epoch $= 1, 2, \ldots$ **do**
    for batch $\in \mathcal{B}$ **do**
      Predict $\mu_{\text{new}}, \sigma_{\text{new}}$ from x_{true}
      Compute **updated** log probability of $\hat{y}$, $\log p_{\mu_{\text{new}},\sigma_{\text{new}}}(\hat{y})$
      Compute ratio $r = \frac{\log p_{\mu_{\text{new}},\sigma_{\text{new}}}(\hat{y})}{\log p_{\mu,\sigma}(\hat{y})}$
      Clip ratio r
      Standardize $BPHR$ of batch
      $\mathcal{L} = r \cdot BPHR$
      Update model parameters using $\nabla\mathcal{L}$
    end for
  end for
end for

5

Evaluation

The goal of this chapter is to describe the evaluation carried out to answer the research questions specified in Chapter 3. It begins with Section 5.1, which delves into the process of generating evaluation instances. This involves combining the instances from PSPLIB (Kolisch and Sprecher, 1997) with artificially created contextual features (Tang and Khalil, 2022). Firstly, it describes the structure of each PSPLIB scheduling instance and then introduces in detail the process of generating artificial contextual data, including the hyperparameters involved and how the data is adapted to suit our context. Subsequently, the next four sections address each specific research question. Section 5.2 focuses on the basic score function implementation by evaluating different sampling distributions. Section 5.3 is dedicated to the two reinforcement learning-based extensions: baseline post-hoc regret and PPO updates. In Section 5.4, different problem settings are explored, examining the impact of instance and penalty settings. Finally, Section 5.5 investigates the influence of uncertainty on the method. These sections follow a consistent structure, beginning with an explanation of the experimental approach adopted to address each research question. Subsequently, the results of the experiments are presented, followed by a synthesized conclusion summarizing the outcomes.

5.1. Data and Instance Generation

The benchmark dataset utilized in this study is PSPLIB, which contains instances of the RCPSP problem. Each data instance follows a specific structure. All instances have a predetermined number of tasks. For the purposes of this research, only instances with 30 and 90 tasks are utilized. Subsequently, the data about each task is described. This includes the processing time, the list of successors, and the resource usage of each type. Notably, there are four different types of resources available. Apart from the data about the tasks, the dataset provides information on the total renewable resource availability of each type.

The settings this work aims to explore involve training a predictive model to estimate the processing times of the tasks based on a set of input contextual features. However, the PSPLIB dataset does not include these necessary input features. To address this limitation, it has been decided to disregard the provided task durations and instead generate new durations that incorporate contextual features. This approach allows for the exploration of the desired setting within the constraints of the available dataset.

5.1.1. Data Generation Process

The need for artificial contextual features is a common challenge within the domain of DFL, primarily because many of the commonly used problem instances lack such features. The data generation process mimics the one used in the works of Elmachtoub and Grigas (2022) and Silvestri et al. (2023). For each scheduling instance, a set of p input features, denoted by the vector $\mathbf{x}_i$, is sampled from a multivariate Gaussian distribution with a zero mean and unit variance, i.e., $\mathbf{x}_i \sim \mathcal{N}(0, I_p)$. Subsequently, the durations of the q tasks in the problem instance, denoted as $\mathbf{d}_i$, are computed (where q could be either 30 or 90 tasks depending the configuration investigated by the experiment). The duration of the j^{th} task in the i^{th} scheduling instance, d_i^j, is calculated using the formula:

$$d_i^j = [(\frac{1}{\sqrt{p}}(\mathcal{B}\mathbf{x}_i)_j + 3)^{deg} + 1] \cdot \epsilon_i^j$$

Here, $\mathcal{B} \in \mathbf{R}^{m \times p}$ is a random matrix where each entry is a Bernoulli random variable that equals 1 with a probability of 0.5. The *deg* parameter is a positive integer that determines the degree of the polynomial relationship between the input features and the output durations. The term ϵ_i^j introduces multiplicative noise into the computation.

This data generation process was chosen for two main reasons. First, it allows control over the amount of uncertainty introduced through its hyperparameters, which facilitates its evaluation. Second, it is the most common method used to generate artificial features for DFL tasks (Tang and Khalil, 2022; Elmachtoub and Grigas, 2022; Silvestri et al., 2023).

5.1.2. Hyperparameters of the Data Generation

As mentioned earlier, the data generation procedure has two main hyperparameters - *deg*, and ϵ_i^j. Their impact will be experimentally evaluated in Section 5.5. *deg* increases the complexity of the input-output relation, making it more challenging for the underlying linear predictive model to capture. Thus, the *deg* parameter controls the extent of model misspecification. ϵ_i^j is sampled from a uniform distribution in the interval $[1 - \nu, 1 + \nu]$, where $\nu \in [0, 1]$ is a parameter controlling the width of the distribution of multiplicative noise.

5.1.3. Scaling

The data generation process, as described in Subsection 5.1.1, produces values that might be negative or fractions. However, the aim is to adhere to the format of durations in the original PSPLIB, where they can only take integer values ranging from 1 to 10. Therefore, the values generated by the data process are also rescaled. The rescaling works as follows: initially, the smallest value is mapped to 1, and the largest to 10. Subsequently, the remaining $N - 2$ values are mapped to the range $[1, 10]$ while maintaining their relative distances. Finally, they are rounded to the nearest integer.

Important Note: In the original PSPLIB instances, the number of occurrences of the 10 possible values for duration is uniformly distributed, i.e., the number of tasks in the dataset with a duration of 1 is approximately equal to the number of tasks with durations of 2, 3, 4, and so on. However, when using synthetic data generation with scaled durations, the processing times no longer exhibit a uniform distribution. Instead, they seem to follow a distribution similar to Poisson. For smaller values of *deg*, the distribution is fairly balanced. However, as the value of *deg* increases, the distribution becomes highly imbalanced with a long tail. For example, when $deg = 8$, approximately 60% of the values are either 1 or 2, 20% are 3 or 4, and only 20% are greater or equal to 5.

In Section 5.2, Section 5.3, and Section 5.4, the evaluation primarily focuses on the model and the

framework rather than the data itself. Therefore, the parameters deg and ϵ_i^j will remain fixed. Specifically, we have set $deg = 4$ and $\epsilon_i^j = 0.5$. These values have been deliberately chosen to retain a level of uncertainty in the data generation process without overwhelming the evaluation. Higher values could potentially introduce bias towards a specific type of uncertainty, thereby affecting the overall assessment. The impact of uncertainty is then evaluated in Section 5.5.

5.2. Research Question 1: Sampling Distribution and Warm-starting

Formulation: How does the choice of sampling distribution in the Score Function Gradient Estimation method impact the training performance of the overall algorithm?

As discussed in Section 4.2, rather than generating point predictions, SFGE operates under the assumption that the predicted processing times adhere to specific probability distributions. In this study, four distinct distributions were selected in order to evaluate the impact of the sampling distribution on SFGE.

The first distribution chosen is Normal distribution, widely adopted in SFGE applications (van den Houten et al., 2024; Silvestri et al., 2023). Two different settings of Normal distribution have been chosen. Both of them have an N-dimensional mean vector μ, where N refers to the number of tasks in the scheduling instance. However, in one of them, all tasks share a single scalar standard deviation (σ), and another where each task has its own σ, forming an N-dimensional vector.

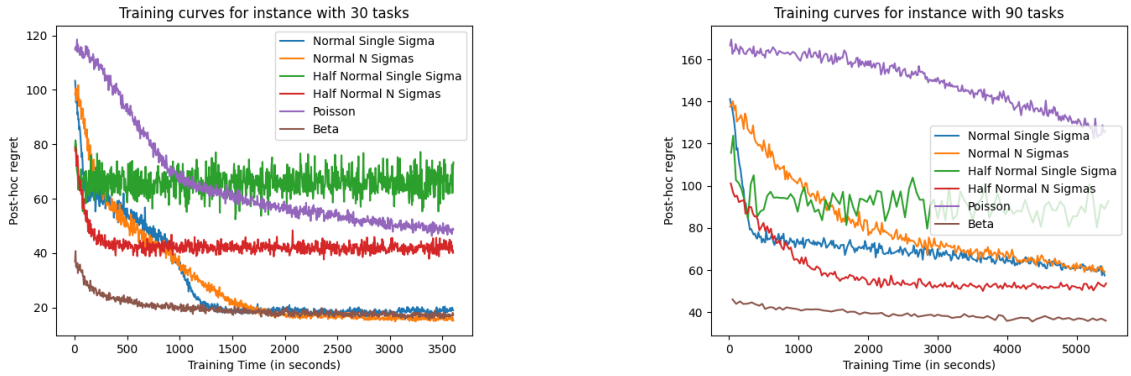
The second selected distribution is Half-Normal. The motivation behind its choice is that DFL-trained models generally predict more conservatively because they are trained using post-hoc regret, and in order to minimize it, they try to avoid accumulating penalty. As a consequence, they tend to predict higher values compared to PFL methods. As samples from Normal distribution may have a value lower than its mean, it is possible that Half-Normal converges faster as it always predicts on the "correct" side of the mean to avoid penalty. Similarly to Normal distribution, Half-Normal is used in two settings, with σ as a scalar and a vector.

The third distribution used is Beta. It is another common choice when applying SFGE. It is defined using two parameters α and β, which we have chosen to be N-dimensional vectors. Generally speaking, Beta distributions come in more diverse shapes compared to the Normal. The values of the samples from Beta distribution are always between 0 and 1. Thus, they need to be rescaled to fit the general convention that we decided to adhere to, that the values for the processing times range between 1 and 10. We do that by multiplying the value of the samples by nine and adding one.

The fourth one is Poisson. Typically, SFGE is applied with continuous distributions, but if the underlying data follows a discrete distribution, it is also feasible to use such. As described in Subsection 5.1.3, the shape of the distribution of the rescaled processing times closely resembles that of Poisson; thus, it is also sensible to try to apply SFGE using it.

To evaluate the efficacy of each distribution, we train the model while using each of the distributions on two instances: one with 30 tasks and one with 90 tasks. We set a time limit of one hour for the 30-task instance and of one hour and a half for the 90-task instance. The penalty factors of the post-hoc regret are set to $\frac{1}{3}$ for the instance with 30 tasks and $\frac{1}{9}$ for the instance with 90. The results are shown in Figure 5.1.

Examining Figure 5.1a, it is evident that Beta distribution achieves the quickest convergence to a post-hoc regret value of under 20. Both versions of Normal distribution also reach this threshold but at a slower pace. Meanwhile, both variants of Half-Normal distribution converge rapidly but to a higher regret value, whereas Poisson distribution fails to converge within the time limit. This trend appears consistent when analyzing the instance with 90 tasks in Figure 5.1b. Once more, Beta distribution



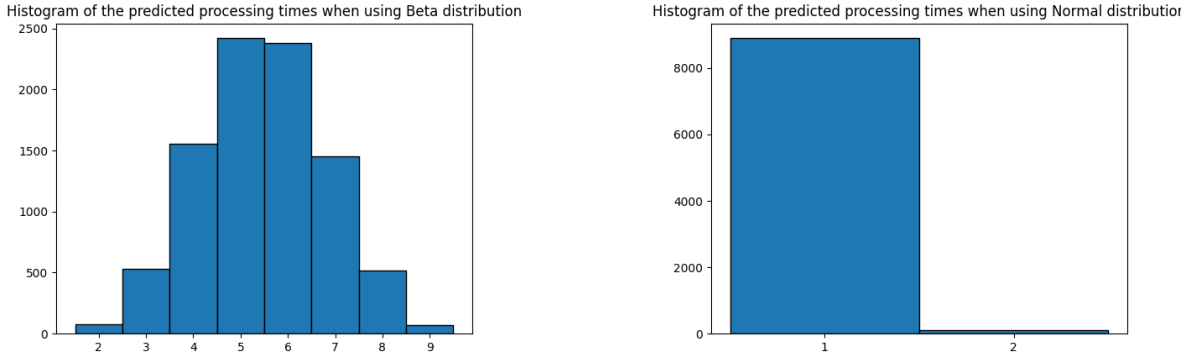
(a) Instance with 30 tasks

(b) Instance with 90 tasks

Figure 5.1: Training curves for the different distributions. The results reveal distinct convergence behaviors. In the 30-task instance, Beta distribution converges most rapidly, followed closely by the two Normal distributions. Conversely, other distributions lag behind in convergence speed. In the 90-task instance, Beta distribution demonstrates superior performance from the outset, maintaining the lowest regret throughout the training process.

exhibits the fastest convergence to the lowest regret value, with a notably wider gap compared to the other distributions.

A noteworthy observation is that even the initial value for Beta distribution outperforms the final values of the other distributions in Figure 5.1b. This observation indicates the significance of the predictive model's initialization. Due to the rescaling needed by Beta distribution, its initial predictions, as depicted in Figure 5.2a, encompass a broader range of values for the processing times. In contrast, the distribution of initial predictions made using Normal distribution, as illustrated in Figure 5.2b, predominantly centers around ones and rarely extends to twos.



(a) Beta Distribution

(b) Normal Distribution with scalar σ

Figure 5.2: Distributions of the predictions made during the first epoch of training. The graphs reveal notable differences among the initial distributions. Figure 5.2a clearly shows that Beta distribution starts with promising initial predictions. In contrast, Figure 5.2b demonstrates that the initialization of the Normal distribution yields initial predictions mainly centered around 1, which inadequately models the underlying problem.

To mitigate this disparity and ensure a more equitable comparison, one could improve the initialization of the other models. This involves employing warm-starting of the predictive models when utilizing Normal and Half-Normal distributions. A suitable starting point for these models is to use the model checkpoint obtained when performing PFL, which not only can be quickly obtained but also generates predictions close to the true values. Subsequently, the experiment for the 90-task instance was repeated with warm-starting. The results, depicted in Figure 5.3, reveal a significant shift in the outcomes when warm-starting is employed. Under these conditions, Beta distribution performs the poorest among all. In contrast, the Half-Normal distributions emerge as the top performers. Normal distribution with a

scalar σ also achieves a similar regret level, but it requires a longer time to do so.

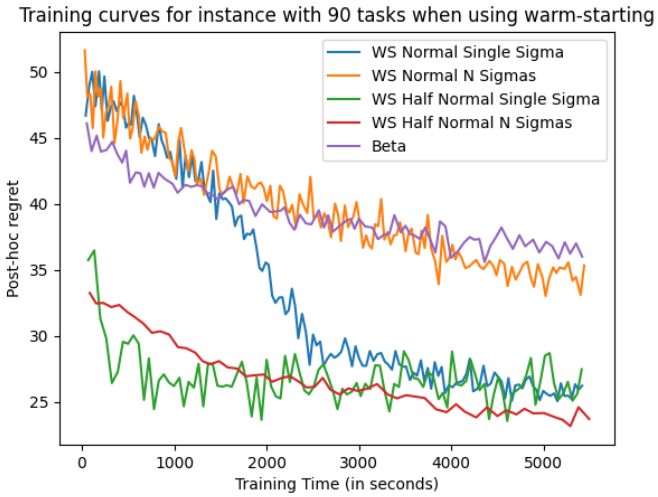


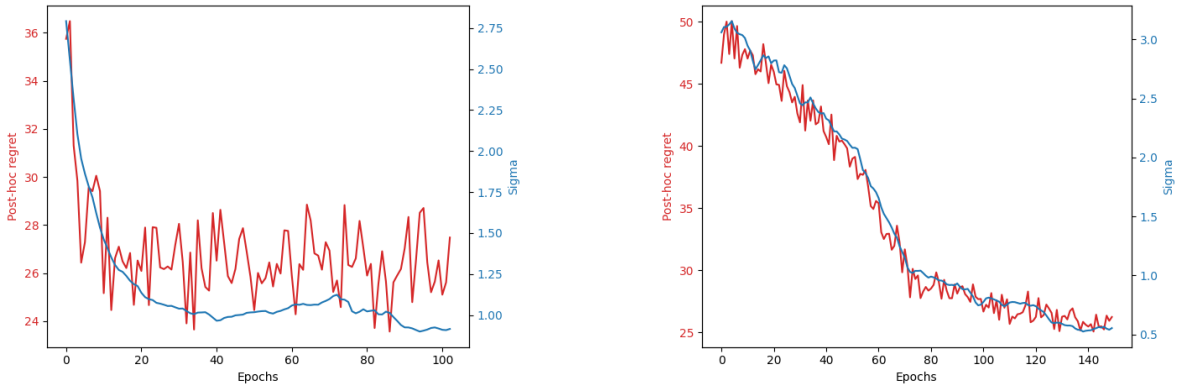
Figure 5.3: Training curves when using warm-starting for the Normal distributions on the instance with 90 tasks. The graph depicts a reversed scenario compared to the previous experiment. Following warm-starting, Normal now exhibits significantly improved performance compared to Beta. Additionally, Half-Normal demonstrates the quickest convergence due to it being able to better exploit its favorable initialization.

The performance changes observed can be attributed to two factors: post-hoc regret and exploration of the solution space by SFGE using different distributions. Post-hoc regret is comprised of two main components: the difference between predicted and true makespan ($f(x^*(\hat{y})) - f(x^*(y))$) and the penalty term ($Pen(x^*(\hat{y}))$). Increasing the predicted durations reduces the likelihood of incurring penalties but leads to a larger discrepancy in makespan. Thus, DFL aims to find a balance in the trade-off between these two components. Penalizing corrections more strictly amplifies the influence of the penalty term, prompting the algorithm to predict larger values.

Understanding that DFL tends to predict larger processing times, we can examine how Normal and Half-Normal distributions explore the solution space. The warm-start initialization gives both versions a good starting point. Half-Normal always samples values greater than the mean, efficiently exploring the direction favorable for DFL. Consequently, it converges quickly to a favorable value. Conversely, the Normal distribution samples both larger and smaller values, taking more time to converge. This observation aligns with the movement of the value of σ throughout epochs in Figure 5.4. In the case of Half-Normal distribution (Figure 5.4a), σ quickly declines as sampling values a lot bigger than the PFL could quickly have an opposite effect on the makespan difference component. When the σ is low, the exploration is limited, and the post-hoc regret stays within the 24 to 30 range. Conversely, the smoother decline in σ for Normal distribution (Figure 5.4b) leads to a slower decrease in regret.

While the limited exploration benefits the Half-Normal in this scenario, it may hinder convergence to an optimal value in environments with higher uncertainty. In such cases, the PFL warm-start would not be as beneficial, and Half-Normal would have suffered. Conversely, despite taking more time to explore, Normal distribution provides more reliable results, even without warm-start initialization, enabling it to reach a relatively good final value.

Research Question 1 Conclusion: The initialization of the sampling distribution is pivotal for SFGE, and warm-starting can severely enhance its performance. Notably, Half-Normal distribution exhibits commendable performance when the warm-start is favorable. Conversely, Normal distribution may be slower but offers greater reliability.



(a) Half-Normal Distribution with scalar σ

(b) Normal Distribution with scalar σ

Figure 5.4: Movement of σ (in blue) and post-hoc regret (in red) over the epochs. Figure 5.4a shows that the value of σ quickly decreases, indicating that Half-Normal relies more on exploitation. On the flipside, Figure 5.4b illustrates that Normal distribution spends more time exploring the solution space.

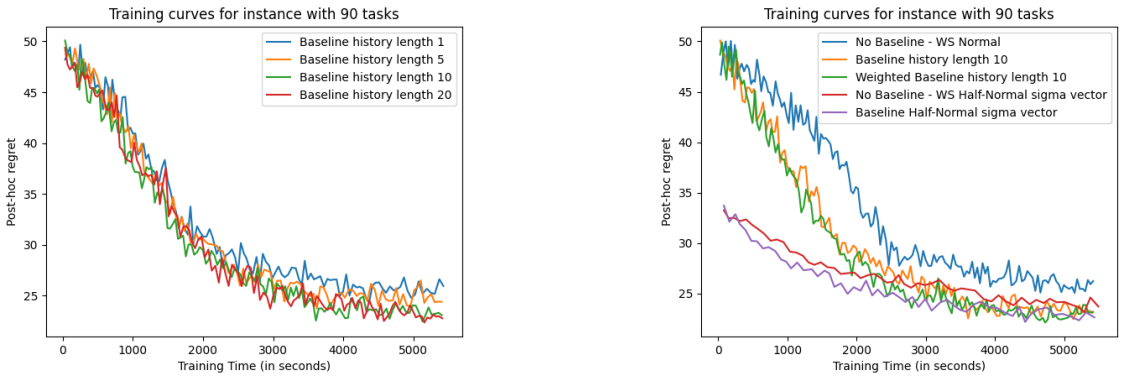
5.3. Research Question 2: RL-Inspired Extensions

Formulation: Would the addition of reinforcement learning-based techniques improve the convergence speed of the Score Function Gradient Estimation method when applied to DFL?

5.3.1. Sub-question 1: Baseline Post-Hoc Regret

This section is dedicated to evaluating the baseline post-hoc regret extension introduced in Subsection 4.6.1. Initially, we optimize its hyperparameters, followed by a comparative analysis against other methods.

The baseline extension has one hyperparameter – the number of historical steps considered (hereafter referred to as history length). To assess its efficacy, we conduct an experiment examining algorithms differing solely by their history lengths. The selected values for evaluation are: 1, 5, 10, and 20, encompassing a spectrum from the smallest possible value to a longer history length. This experiment is conducted exclusively on the instance with 90 tasks, because of its greater complexity compared to the 30-task instance. The penalty factor remains at $\frac{1}{9}$.



(a) History length of the baseline extension

(b) Comparison between the different methods

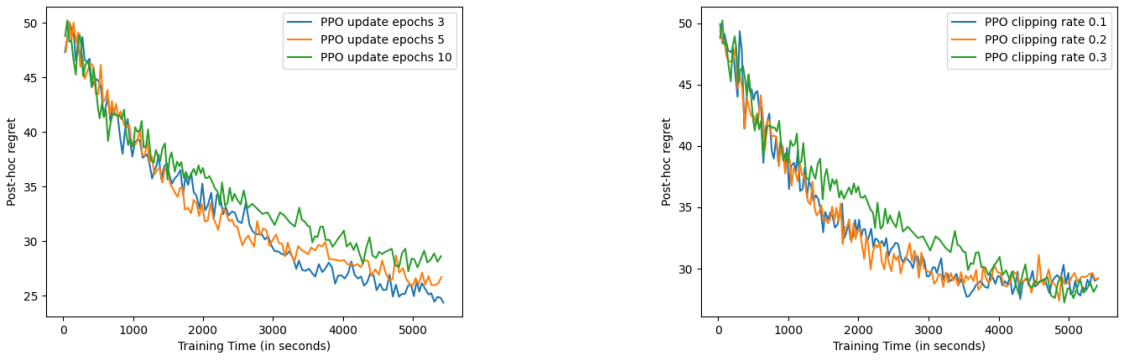
Figure 5.5: Evaluation of the baseline extension. Figure 5.5a shows that the length of the baseline history is not that impactful on the training process, with a length of 10 performing the best. Figure 5.5b illustrates the benefits of the baseline extension – there is no non-baseline version that outperforms its baseline counterpart.

In Figure 5.5a, training curves for varying history lengths are plotted. It is evident that longer history lengths result in slightly better performance, with the optimal length identified as 10. With this determined, we proceed to compare different versions of the algorithm, as depicted in Figure 5.5. It is noteworthy that the baseline version of regret outperforms the standard non-baseline version across both Normal and Half-Normal distributions. Specifically, the non-baseline SFGE with Normal distribution achieves a training post-hoc regret value of 25.06, whereas the baseline version reduces it to 22.47. Similarly, in the case of Half-Normal distribution, the achieved values are 23.18 and 22.18, respectively. Additionally, we evaluate a weighted version of the baseline. In this form, rather than treating each element in the baseline history with equal weight, we prioritize the more recent observations. Remarkably, this enhancement yields even better results, reducing the achieved post-hoc regret to 22.13.

5.3.2. Sub-question 2: PPO Updates

Proximal Policy Optimization represents a novel approach yet to be explored within the context of DFL. Mirroring the previous section, the evaluation begins by optimizing the hyperparameters before conducting comparative analyses with the other DFL algorithms. For this experiment, we maintain consistency with previous settings, focusing solely on the instance with 90 tasks and a penalty factor of $\frac{1}{9}$.

The implementation of PPO in DFL proposed in Subsection 4.6.2 incorporates two primary hyperparameters: the clipping rate and the number of epochs in the PPO update loop. The first parameter to be evaluated is the number of epochs. The values that we are going to test are 3, 5, and 10. In order to do this, we are going to fix the value of the clipping rate to 0.2, which is a rule of thumb value for the clipping value in the PPO literature (Schulman et al., 2017). Results presented in Figure 5.6a reveal that lower epoch values perform better, with 3 epochs achieving a training post-hoc regret of 24.38. With the optimal epoch number determined, attention turns to assessing clipping rates, with values of 0.1, 0.2, and 0.3 being under consideration. Notably, larger values are uncommon due to their tendency to diminish update sizes. Although differences are marginal, a recommended clipping rate of 0.2 demonstrates slightly better performance overall (refer to Figure 5.6b).



(a) Number of epochs of the PPO update loop

(b) Clipping rate of PPO

Figure 5.6: PPO hyperparameter evaluation. The figures show that the best hyperparameters for PPO are to use 3 epochs and a clipping rate of 0.2.

Subsequently, our focus shifts to evaluating whether the optimized PPO configuration translates to tangible gains in the training process relative to non-PPO methods. For this, we compare 3 methods. The first one does not use PPO – it is the weighted baseline described in the previous section. The other two are PPOs – the first one is without a baseline, and the other uses a weighted baseline. As depicted in Figure 5.7, the non-PPO method exhibits the fastest convergence and attains the most favorable regret value. While the discrepancy with PPO methods is relatively small (less than 10%), this experiment suggests that PPO fails to provide notable benefits to DFL training. Nonetheless, given the novelty of the method and the minor differences from the others, PPO should be considered further

as future work.

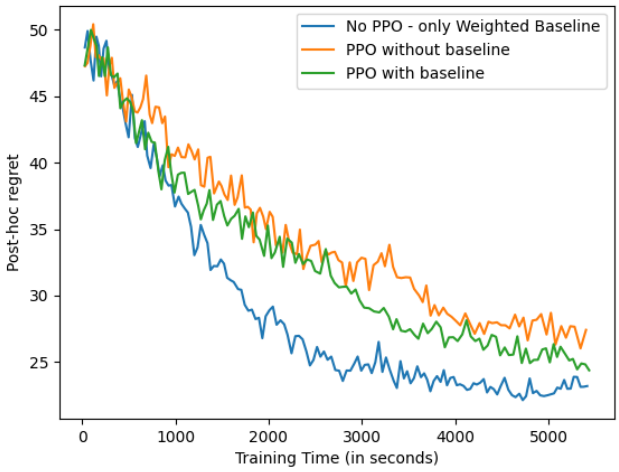


Figure 5.7: Comparison between the different versions of PPO and baseline. The plot shows that the non-PPO version of the algorithm outperforms both PPO versions, rendering the PPO extension not beneficial.

Research Question 2 Conclusion: The baseline post-hoc regret extension boosts the convergence speed and reduces the achieved post-hoc regret of the algorithm. Optimal settings for the baseline include a history length of 10 and prioritization of the most recent observations.
Unfortunately, PPO did not improve the convergence of the Decision-Focused Learning pipeline. Still, its performance remains respectable, and the method deserves further consideration in the future.

5.4. Research Question 3: Impact of the Added Penalty

Formulation: What is the impact of the penalty incurred due to constraint violations on the Decision-Focused Learning algorithm?

The penalty factor could be interpreted as part of the problem. It plays a crucial role in the DFL framework, reflecting the decision-maker's perspective on the significance of each correction. In real-world applications, this penalty factor should ideally mirror the true impact of corrections. Thus far, our experiments have utilized a moderate penalty factor of $\frac{1}{9}$, striking a balance in the significance of the penalizing. In this section, we delve into the performance implications of varying penalty factors.

We assess three penalty factors: a smaller penalty of $\frac{1}{90}$, labeled as "small," the existing penalty of $\frac{1}{9}$ referred to as "medium", and a larger penalty of 1, denoted as "large." Considering the penalty as an integral aspect of the problem, we evaluate its effects across three distinct 90-task instances using both DFL and PFL. The DFL method under evaluation is the most effective approach identified thus far: the weighted baseline SFGE with warm-started Normal distribution using a scalar σ. For this evaluation, nine distinct versions of this model have been trained, each corresponding to a specific combination of instance and penalty factor. Conversely, the PFL method entails a linear regression minimizing mean squared error trained on the same dataset as the DFL methods. Our evaluation entails a test set comprised of 100 scenarios per instance, where each scenario could be looked at as a pair of input contextual features and 90 processing times. The main metric used is the mean post-hoc regret across the 100 scenarios. Additionally, the two components of the post-hoc regret, makespan difference and penalty, are also presented separately.

The experimental results, detailed in Table 5.1, reveal notable trends. Firstly, a consistent finding across all three instances is the significant influence of the penalty factor used during training on the

Penalty		large		medium		small	
Instances	Metric	DFL	PFL	DFL	PFL	DFL	PFL
Instance 1	Post-hoc Regret	**45.84**	325.89	**25.11**	46.23	**13.88**	14.77
	Makespan	23.74	11.27	14.15	11.27	10.65	11.27
	Penalty	22.10	314.62	10.96	34.96	3.23	3.50
	MSE	5.74	1.88	3.84	1.88	3.20	1.88
Instance 2	Post-hoc Regret	**47.35**	355.12	**26.29**	49.37	15.32	**14.97**
	Makespan diff	27.99	11.15	14.56	11.15	11.53	11.15
	Penalty	19.36	343.97	11.73	38.22	3.79	3.82
	MSE	9.96	1.88	7.42	1.88	6.96	1.88
Instance 3	Post-hoc Regret	**46.09**	274.76	**24.94**	39.31	12.98	**12.82**
	Makespan diff	22.18	9.88	14.07	9.88	10.12	9.88
	Penalty	23.91	264.88	10.87	29.43	2.86	2.94
	MSE	6.16	1.88	4.44	1.88	3.30	1.88

Table 5.1: Comparison between DFL and PFL for the three different penalty factors across three different instances. The table illustrates that DFL is able to outperform PFL for all three instances when the penalty factor is large or medium. The main reason for this difference is that DFL manages to avoid accumulating large penalties.

convergence of the DFL model. Higher penalties lead to larger MSE, indicating a greater deviation from the warm-starting initialization compared to instances with smaller penalties. This observation is further supported by examining the makespan difference component of the post-hoc regret, which registers the highest values for the large penalty factor. This occurs because the model prioritizes reducing the impact of the penalty on the post-hoc regret. Conversely, instances trained using a small penalty factor exhibit lower makespan differences, close to that of PFL, as the penalty's influence is less pronounced under these settings. With this in mind, let's proceed to compare the results between DFL and PFL. The disparity in post-hoc regret between DFL and PFL is most pronounced when the penalty is large. Across all three instances, DFL demonstrates a post-hoc regret at least six times smaller than PFL. Similarly, with a medium-sized penalty factor, DFL still outperforms PFL significantly, albeit with a relative difference of under two-fold. This discrepancy arises from PFL's tendency to maintain a small MSE and make predictions closer to the true values, thus incurring a high penalty. However, the dynamics shift when considering the small penalty factor, where the strict penalization is not present, emphasizing the significance of makespan differences. Consequently, PFL manages to gain a slight advantage over DFL in instances 2 and 3.

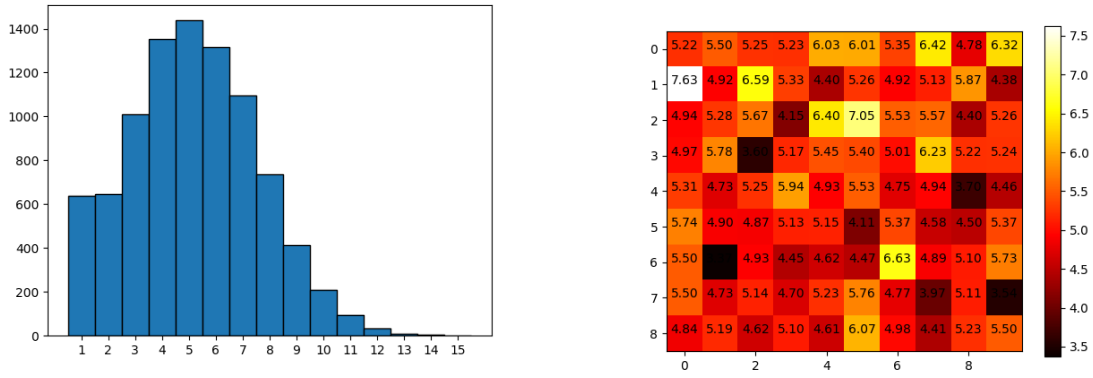
Method	Added Constant lag					
Penalty	large			medium		
Amount of lag	1	2	3	1	2	3
Post-hoc Regret	89.12	**38.03**	39.16	**21.72**	24.55	35.67
Makespan diff	13.29	22.87	35.23	13.29	22.87	35.23
Penalty	75.83	15.16	3.93	8.43	1.68	0.44

Table 5.2: Evaluation of the constant lag PFL method. The table shows the efficacy of the extension, which is able to outperform DFL for both large and medium penalties.

Inspired by the trade-off between the makespan difference and the penalty components of the post-hoc regret, we introduce an extension to the basic PFL method. Referred to as constant lag PFL, this variant adds a constant lag to the processing times for each task. The motivation behind this extension is to explore the possibility of sacrificing a portion of the makespan difference to alleviate some of the penalty incurred, particularly in cases with large and medium penalty factors. The performance of the constant lag PFL method on Instance 1 is illustrated in Table 5.2. Notably, this extension surpasses DFL for both large and medium penalty factors. With an added lag of 1, the post-hoc regret for the

medium penalty factor decreases from 46.23 to 21.72, outperforming DFL by approximately 3.5 regret. Similarly, for the large penalty, a constant lag of 2 significantly reduces the penalty component of the post-hoc regret from 314.62 to 15.16, resulting in an overall post-hoc regret of 38.03, also superior to the DFL counterpart.

Despite the fact that DFL performed worse than constant lag PFL, it still outperforms the general PFL for instances with a large penalty by a long shot. An interesting observation regarding DFL is its high MSE. This prompted an examination of the quality of DFL predictions. Figure 5.8a presents a histogram of predicted values by DFL, revealing predictions exceeding the feasible range, i.e., predicted processing times with values greater than 10. Moreover, Figure 5.8b demonstrates that certain tasks exhibit higher average values than others, suggesting that DFL not only predicts higher values on average, but that it also makes those high predictions for some specific targeted tasks and this way learning a complex underlying structure for the problem.



(a) Distribution of the predicted processing times when using DFL with large penalty factor

(b) Heat map of the mean predicted processing time per task

Figure 5.8: Predictions made by DFL when the penalty factor is large. Figure 5.8a shows that the predictions made by DFL are higher than the actual true values. Figure 5.8b displays that DFL makes these high predictions for specific tasks, indicating that it learns an underlying structure within the problem.

Motivated by DFL's incorporation of complex structures, we explore a hybrid approach combining structural insights with constant lag PFL. This method, termed progressive lag PFL, leverages the precedence constraint present in the RCPSP problem. It does so by using the following observation: if a task that has numerous successors is corrected, then its immediate successors might also need to be corrected. If the immediate successors get corrected, then their successors might also get corrected and so on. Therefore, it is logical to give a higher lag to the tasks that have a lot of layers of successors in order to dodge a penalty avalanche and lower lags to those with no successors as they could not lead to penalty. As the tasks that are early in the predecessor tree are the tasks that have low index, the method divides the 90 tasks into three groups of 30 tasks. Then, each task within a group receives the same lag, with differing lags assigned to each group. The results of this method are shown in Table 5.3. Progressive lag PFL achieves the lowest post-hoc regret of methods for instances with large penalty factors by applying a lag of 3 to the first 60 tasks and a lag of 2 to the remaining 30. Still, its performance for instances with medium penalty factors is inferior to that of a constant lag of 1.

Overall, constant and progressive lag PFL offer promising alternatives to DFL, demonstrating quicker training times and easy explainability. However, their applicability may be challenged in more complex contexts. Furthermore, caution is advised when adjusting the lag, as it may lead to overfitting to the specific instance. More about the impact of the PFL extensions and how they affect the general DFL horizon is present in Section 7.1.

Method	Progressive lag							
Penalty	large				medium			
Amount of lag	2-1-0	3-2-1	3-2-2	3-3-2	2-1-0	3-2-1	3-2-2	3-3-2
Post-hoc Regret	91.13	41.61	37.89	**36.82**	24.09	25.16	28.27	30.94
Makespan diff	15.71	23.10	27.07	30.20	15.71	23.10	27.07	30.20
Penalty	75.42	18.51	10.82	6.62	8.38	2.06	1.20	0.74

Table 5.3: Evaluation of the progressive lag PFL method. In the "Amount of lag" row, "2-1-0" indicates that the first 30 tasks will receive an increase of 2, the second 30 an increase of 1, and the last 30 an increase of 0. The "3-3-2" configuration performs the best among all the models evaluated for large penalties. Its performance for medium penalties is good but not better than that of the constant lag PFL.

Research Question 3 Conclusion: The penalty factor plays an important role in the DFL pipeline, shaping the characteristics of the resulting DFL models. Notably, DFL demonstrates superior performance compared to PFL under scenarios with medium and large penalties. However, incorporating lag into the conventional PFL framework could substantially enhance its performance, surpassing that of DFL. Additionally, observations suggest that DFL exploits an inherent structure in its predictions.

5.5. Research Question 4: Effect of Uncertainty

Formulation: How does uncertainty affect the performance of the Decision-Focused Learning method for scheduling with uncertain processing times?

Next to the penalty factor, the amount of uncertainty is another critical component of the problem, which models the environment. As discussed in Section 2.3, there are two types of uncertainty: aleatory uncertainty, which arises from inherent randomness in the processes, and epistemic uncertainty, which conveys incomplete knowledge about the process. The amount of uncertainty can be controlled through the hyperparameters of data generation. Epistemic uncertainty can be varied by changing the polynomial degree of the relationship (deg). Similarly, aleatory uncertainty is influenced by the noise parameter (ϵ).

To evaluate the impact of each type of uncertainty in isolation, we fix the value of one hyperparameter and vary the other. In the first experiment, we assess the effect of aleatory uncertainty by fixing the polynomial degree to 4 and testing on five different datasets with noise levels: $\epsilon = \{0, 0.1, 0.3, 0.5, 0.7\}$. In this part, we compare the performance of four methods: DFL (weighted baseline SFGE with warm-started Normal distribution), PFL, Added Lag 1 PFL, and Added Lag 2 PFL (Added Lag 1/2 PFL refers to the added constant lag extension of PFL where a constant lag of 1/2 is added to each prediction). We keep the penalty factor fixed at the previously used moderate value of $\frac{1}{9}$.

The results, presented in Table 5.4, show that increasing noise levels correspondingly increase the post-hoc regret across all methods. The performance of the first three methods (DFL, PFL, and Added Lag 1 PFL) is similar. Each of them exhibits a similar relative increase in regret with higher noise levels, resulting in an approximate 50% rise when comparing no noise to the highest noise level of 0.7. This increase is primarily driven by changes in the penalty term of the post-hoc regret, while the makespan term remains relatively constant. Among these, Added Lag 1 PFL performs the best, closely followed by DFL, with a consistent difference of about one post-hoc regret unit at each noise level. PFL lags behind due to a higher penalty accumulation. The fourth method, Added Lag 2 PFL, shows minimal variation and appears more robust. However, this does not necessarily mean that it always performs the best. In fact, its performance is poor at low noise levels because it tends to be over-conservative. Based on these results, we conclude that for low noise levels, either Added Lag 1 PFL or DFL is preferable, as they maintain a balance between the penalty and makespan terms of the regret. For noise levels above 0.5, the more robust Added Lag 2 PFL is likely the better choice, as it minimizes accumulated penalties

Noise	Metric	DFL	PFL	Added 1	Added 2
0	PHR	18.62	30.47	17.23	24.51
	Makespan	11.56	8.55	13.08	23.95
	Penalty	7.06	21.92	4.15	0.56
0.1	PHR	19.41	31.16	18.70	24.10
	Makespan	12.25	8.52	12.87	23.55
	Penalty	7.16	22.64	3.83	0.55
0.3	PHR	22.22	38.98	19.38	24.14
	Makespan	13.39	9.92	12.95	23.12
	Penalty	8.83	29.06	6.43	1.02
0.5	PHR	25.11	46.23	21.72	24.55
	Makespan	14.15	11.27	13.29	22.87
	Penalty	10.96	34.96	8.43	1.68
0.7	PHR	26.65	45.77	25.55	25.53
	Makespan	15.39	10.36	14.60	22.97
	Penalty	11.26	35.41	10.95	2.56

Table 5.4: Post-hoc regret (PHR) for the different noise levels. The table shows that increasing the noise results in higher regret. DFL and Added Lag 1 maintain similar levels, while PFL performs worse. Added Lag 2 PFL demonstrates robust performance, maintaining consistency across noise levels.

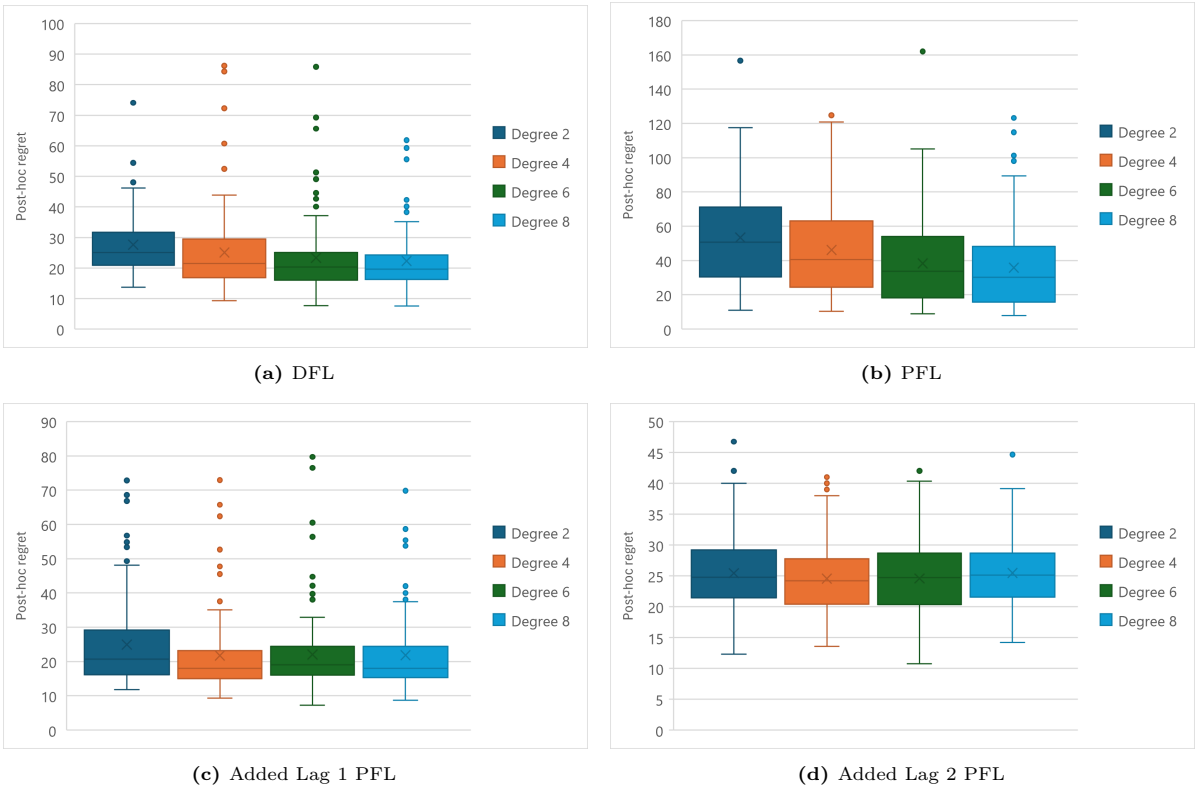
and manages to keep the regret at a controlled level.

The second experiment focuses on epistemic uncertainty. In this experiment, we fix the noise value at 0.5 and evaluate four different polynomial degrees $deg = \{2, 4, 6, 8\}$. We use the same four methods as in the previous experiment: DFL (weighted baseline SFGE with warm-started Normal distribution), PFL, Added Lag 1 PFL, and Added Lag 2 PFL.

Degree	Metric	DFL	PFL	Added 1	Added 2
2	PHR	27.60	53.34	24.91	25.42
	Makespan	15.44	13.15	14.48	23.25
	Penalty	12.16	40.19	10.43	2.17
4	PHR	25.11	46.19	21.71	24.55
	Makespan	14.15	11.27	13.29	22.87
	Penalty	10.96	34.92	8.42	1.68
6	PHR	23.32	38.29	22.03	24.56
	Makespan	13.28	9.03	13.39	22.82
	Penalty	10.04	29.26	8.64	1.74
8	PHR	22.28	35.75	21.83	25.46
	Makespan	12.83	8.47	13.25	23.01
	Penalty	9.45	27.28	8.58	2.45

Table 5.5: Post-hoc regret (PHR) for the different polynomial degrees. The table indicates that higher polynomial degrees result in lower regret. DFL and Added Lag 1 consistently perform better across various degrees. Added Lag 2 shows stable but slightly higher regret.

The results, presented in Table 5.5, show trends similar to the previous experiment. Added Lag 2 again displays robust performance, with post-hoc regret consistently between 24 and 26, regardless of the degree. In contrast, the other methods exhibit more fluctuation. PFL is the most impacted method



(a) DFL **(b)** PFL

(c) Added Lag 1 PFL **(d)** Added Lag 2 PFL

Figure 5.9: Boxplots illustrating the distributions of the post-hoc regret for the different models. All distributions exhibit positive skewness, which becomes more pronounced with increasing polynomial degree. The distribution for Added Lag 2 PFL stands out as less skewed compared to other models, reflecting its conservative approach.

and shows the biggest change, performing best at degree 8 and worst at degree 2, with an increase in post-hoc regret from degree 8 to degree 2 of about 50%. DFL and Added Lag 1 also experience increases, but the fluctuations are smaller, at approximately 23% and 15%, respectively. These differences when evaluating the degree are smaller than those observed when evaluating the noise, suggesting that DFL and Added Lag 1 adapt more readily to epistemic uncertainty compared to aleatory uncertainty.

An important observation from this experiment is that the post-hoc regret decreases with increasing polynomial degree, contrary to our initial expectations. We anticipated that higher polynomial degrees would make the problem more challenging; however, this does not appear to be the case. Nonetheless, it would be incorrect to directly conclude that increasing the polynomial degree inherently makes the problem easier without performing further experiments. To investigate this further, we plotted the distribution of the post-hoc regret for the different methods and degrees. Focusing on Figure 5.9, we observe that all methods exhibit an increase in their positive skewness as the degree increases, i.e., the mean increases more relative to the median. This increase in skewness can be attributed to two factors: a lower mean and a heavier tail. As discussed in Subsection 5.1.3, scaling causes an imbalance in the distribution of processing times for higher polynomial degrees. Consequently, the average processing time of the tasks decreases from 4.43 for degree 2 to 2.98 for degree 8. Thus, it is normal that the mean post-hoc regret is also higher for the instance with degree 2. Regarding the tail, as the data for the higher degrees contains more tasks with lower processing times, the predictor learns to predict smaller values on average. The problem arises when an example with high processing times is evaluated. Due to its inability to predict these high values, the resulting decision will be significantly worse than the average case in terms of regret. These instances with high-valued processing times are the reason for the heavy tail. This is most evident in Figure 5.9a, where the mean for degree 2 is notably lower than that for degree 8, but degree 8 has more outliers and higher skewness.

From these observations, we can conclude that increasing epistemic uncertainty makes the problem instances easier on average but more complicated if optimizing for the worst-case scenario because of

the model's inability to predict the outliers. The conclusion on when to use each method is similar to that for aleatory uncertainty: if the goal is to achieve good performance on average, Added Lag 1 PFL and DFL are preferable. However, if the environment is characterized by high uncertainty and worst-case performance is crucial, a more conservative method such as Added Lag 2 PFL is recommended.

Research Question 4 Conclusion: Uncertainty plays a significant role in the problem. Increased aleatory uncertainty results in higher penalties, leading to worse performance for the models. On the other hand, epistemic uncertainty simplifies the problem in the average case but complicates it in the worst-case scenario. Mitigating the impact of high-uncertainty environments for both types can be achieved by designing a more conservative and robust predictive model. However, if the uncertainty is negligible, such conservatism is unnecessary and could lead to worse performance.

6

Influence of the Solving Technology

The interaction between the solving technology that computes the optimal solution and the repair mechanism used to fix the infeasible schedule posed a new challenge for training and evaluating the DFL framework. Specifically, the existence of multiple optimal schedules for each scenario means that the solving technology and the specific solution it produces significantly affect decision quality. This discovery may limit the effectiveness of the training process in this study, potentially impacting the evaluation and achieved results negatively. The objective of this chapter is to highlight the necessity for greater attention to be directed towards the solving technology within the DFL domain and to explain the decisions made to navigate the resulting trade-offs. Firstly, Section 6.1 describes why and how the solver and the correction mechanism impact the resulting post-hoc regret. Subsequently, Section 6.2 outlines three critical properties that the solver should possess, assessing various solving configurations to determine their alignment with these criteria. Lastly, Section 6.3 elaborates on the chosen solver configuration, its potential implications on the study's outcomes, and how it limits the research.

6.1. Solving Technology and Its Effects on the Pipeline

While conducting the experiments in this study, it was discovered that the solver is far from being just a black box that outputs solutions, but instead, it has an impact on the computed regret for the solution. This stems from the realization that not every scheduling problem possesses a singular optimal solution. Instead, there exist multiple, often numerous, alternative schedules, each achieving the optimal makespan. However, these optimal schedules may entail varying repair (penalty) costs. In fact, after conducting the experiments, it became evident that the different optimal solutions have very different costs. Consequently, the value of the post-hoc regret, defined as $phr(y, \hat{y}) = f(x^*(\hat{y})) - f(x^*(y)) + Pen(x^*(\hat{y}) \rightarrow x^*_{corr}(\hat{y}, y))$, particularly the penalty term, is intricately linked to the optimal schedule produced by the solver. If we were to change the solver or some of its settings, such as the search strategy, the solver would arrive at a different optimal schedule, and the post-hoc regret value would also change, despite using the same predicted processing times. Alternatively, one might characterize the objective function as a deterministic mapping — given the same processing times, all solvers would arrive at a solution with the same optimal objective value. In contrast, the post-hoc regret was found to be non-deterministic: with the same processing times, different solvers produced distinct optimal solutions based on their internal mechanisms, resulting in different post-hoc regret values.

Having this in mind, the selection of a suitable solver technology emerges as a crucial consideration. Three key characteristics define an ideal solver for DFL training: speed, accuracy, and consistency. Firstly, speed pertains to the time required for the solver to reach the optimal solution. Secondly,

accuracy refers to how closely the regret of the predicted solution aligns with the smallest attainable regret for a solution under identical weights (e.g., processing times in scheduling). Lastly, consistency denotes the tendency for minor changes in input durations not to result in significant alterations in the outputted schedule or the regret associated with it.

6.2. Evaluation of the Solver

To understand whether the solving technology had an impact on the training and the evaluation conducted in Chapter 5 and whether it limits the results, we carried out additional experiments related to the three properties highlighted earlier. The experiments included the previously used IBM ILOG CPLEX solver and evaluated all of the possible search configurations within the solver.

6.2.1. Speed and Accuracy

The first experiment focuses on evaluating both speed and accuracy across four distinct solver settings, each employing different search strategies. Evaluation is conducted on five instances, each consisting of 90 tasks, with ten scenarios per instance. Speed is assessed by counting the number of timeouts within a fixed time limit of 60 seconds; lower timeouts indicate superior performance. Regarding accuracy, obtaining the actual smallest regret necessitates computing all optimal solutions, rendering it impractical. Therefore, comparisons are made relative to the default strategy utilized when computing the results presented in Chapter 5, aiming to identify potential enhancements through alternative solving configurations. The results from this experiment are illustrated in Table 6.1. When it comes to the speed property, the substantial number of timeouts renders the three alternative search strategies (excluding Default) practically infeasible for most instances. As for accuracy, no single solver configuration completely dominates the others. What is crucial for this study is the performance of the Default solver configuration, which outperforms others in the second and third instances and exhibits no more than an 8% deviation in the remaining instances. The main conclusion from the experiment is that given the speed advantage of Default, it appears to be the sole viable option among those tested.

Metric	Instance	Default Solver	MultiPoint	IterativeDiving	DepthFirst
Timeouts	1	0	9	10	9
	2	0	2	4	1
	3	0	9	10	8
	4	0	8	10	8
	5	0	6	9	7
Accuracy	1	-	4.79%	2.77%	4.13%
	2	-	−14.95%	−5.91%	−1.42%
	3	-	−8.41%	−12.71%	0.06%
	4	-	6.95%	1.69%	7.98%
	5	-	0.50%	4.89%	1.97%

Table 6.1: Evaluation of Speed and Accuracy. The maximum number of timeouts is 10. Accuracy is measured relative to the Default Solver. Interpretation: The numerous timeouts make the three alternative search strategies practically infeasible for training. In terms of accuracy, no solver outperforms all other solvers for all instances.

6.2.2. Consistency

Assessing consistency presents a more challenging task compared to evaluating speed and accuracy, necessitating a separate experiment. When evaluating it, the objective is to compare the regret values of solutions generated by the solver using two different sets of durations: y_{true} and $y_{modified}$. If the

disparity is minimal, it implies that the solver is consistent, remaining indifferent to minor alterations in the input. $y_{modified}$ is created by initially using y_{true} and then altering the duration of a single task with no successors. Subsequently, two outcomes emerge for the optimal solution of $y_{modified}$: it either retains the same makespan as before or sees an increase by one. Instances where the former scenario occurs are disregarded, focusing solely on the latter. This choice is dictated by the notion that in the latter scenario, the earliest starts of the optimal schedule computed using y_{true} also serve as an optimal solution when solving for $y_{modified}$. Consequently, we can compare the repair costs of the two resulting optimal solutions, derived from y_{true} and $y_{modified}$, to assess whether the slight modification to obtain $y_{modified}$ impacts the penalty term.

For this evaluation, three instances, each comprising 90 tasks, were utilized. Three distinct baseline models were employed to predict the durations: DFL, warmstarted DFL (which completed its training), and DFL after 100 epochs, to investigate whether the impact varies throughout different stages of the training process. Table 6.2 shows that the penalty difference is somewhat significant – mostly varying between 10% and 20%, while the makespan difference is comparatively smaller. Moreover, it appears that earlier epochs are more susceptible to inconsistent behavior.

Method	Instance	MS diff	Actual MS	% diff	Penalty diff	Actual penalty	% diff
DFL	1	1.00	49.05	2%	8.55	46.23	18%
	2	2.44	47.84	5%	20.22	115.74	17%
	3	2.44	43.61	6%	23.00	104.06	22%
DFL after 100 epochs	1	3.89	25.26	15%	45.64	259.56	18%
	2	2.76	18.62	15%	51.81	449.10	12%
	3	2.75	16.65	**17**%	42.68	366.77	12%
WS DFL	1	1.08	40.12	3%	6.56	36.23	18%
	2	0.57	37.35	2%	1.26	36.74	3%
	3	3.83	29.41	13%	10.02	35.37	**28**%

Table 6.2: Evaluation of Consistency. In the table MS is short for makespan. The penalty difference varies between 10% and 20% which could be impactful, while the makespan difference is negligible.

6.3. Aftermath

The results from these experiments emphasize how difficult the nature of solver selection in decision-focused optimization could be. Despite the unsatisfactory results obtained from assessing the consistency of the solver, practical constraints related to speed dictate that the Default solver remains the only viable option for evaluating the results. Imperfect accuracy and consistency may lead to certain predictions being erroneously deemed poor, and thus slowing the training of the DFL algorithm. Despite these shortcomings, the algorithm still managed to converge to an acceptable outcome when answering the research questions.

This observation illustrates the importance of evaluating the solver in any DFL research involving uncertainty in constraints and employing CP solvers. Looking ahead, an intriguing opportunity for future research could prioritize the development of solver configurations that enhance consistency while maintaining competitive performance in speed and accuracy.

7

Conclusion and Future Work

Combinatorial optimization encompasses a wide class of problems with broad practical applications. However, their practical use is often hindered by the inherent uncertainties present in real-world scenarios. To address these challenges, Decision-Focused Learning (DFL) emerges as a promising framework. Previous research on DFL has explored various approaches to modify the framework to handle problems where uncertainty emerges within the constraints. Yet, a notable limitation of existing research is the restriction to either simple combinatorial problems (Silvestri et al., 2023; van Steijn, 2022) or those adhering to specific structures (Hu et al., 2022, 2023). This study aimed to take the next step by assessing the applicability of the DFL framework to more complex problems, specifically the Resource-Constrained Project Scheduling Problem. By doing so, it looked to enhance our understanding of how DFL can be effectively leveraged to tackle complex combinatorial optimization problems, and what are its strengths and weaknesses. The contributions made in this thesis can be grouped into two groups: firstly, the study of the application of DFL to the scheduling problem with uncertain processing times, and secondly, the analysis and extensions related to the SFGE methodology. Section 7.1 summarizes the first contribution and suggests potential directions for future research that emerged from it. Analogously, Section 7.2 does the same but for the second contribution.

7.1. DFL in Scheduling

The proposed DFL method demonstrated superior performance compared to the conventional ”two-stage” approach, as illustrated in Table 5.1. This difference is most clearly pronounced for the high penalty factor, indicating that DFL is especially effective in instances where making poor decisions incurs substantial costs. The reason why DFL manages to excel is that it makes the processing times of the tasks longer, thereby reducing the penalties it accumulates, which constitute the majority of the post-hoc regret. This increase is particularly noticeable for certain tasks (Figure 5.8b), suggesting that DFL effectively manages to learn a structure about which processing times should be extended. However, the PFL-based extension, which adds constant factors to the PFL-predicted processing times in a predefined structure based on the task’s position, i.e., the progressive lag PFL extension, achieved better post-hoc regret than DFL (Table 5.3). These results indicate that combinatorial optimization problems with complex underlying structures, such as the scheduling problem explored in this study, might be too complicated for the DFL method used in this work. Specifically, the structure learned by the DFL method performed worse than using a predefined structure based on problem-specific heuristics on top of PFL. One potential explanation could be that, despite predicting distributional parameters, DFL relies solely on the mean of the distribution, effectively making it act as a point predictor. This mean is then input into a simple deterministic solver, causing the loss of information related to the

specific scheduling instance.

An interesting avenue for future research would be to explore the possibility of integrating predefined heuristic structures into the training process of data-driven approaches. One potential topic could be to examine whether DFL performs better when it is enforced to adhere to a specific heuristic structure rather than learning one independently. To achieve this, during training, the model could be guided to comply with this structure and be penalized for diverging from it, thereby driving the DFL algorithm to make predictions that align with the heuristic while still learning the optimal setup for it. This investigation could provide insights into the potential benefits of integrating problem-specific heuristics within the DFL framework for various combinatorial optimization problems. If successful, this approach could demonstrate how leveraging domain knowledge and heuristic structures can enhance the effectiveness of DFL across different applications. This would suggest that a hybrid method, combining DFL with established heuristics, might be particularly powerful for tackling complex problems with intricate dependencies and constraints, leading to more efficient and accurate solutions in practical settings. A different promising area for research could do the opposite and would involve initially providing the algorithm with a predefined structure based on heuristics, but now allowing it to deviate during training. By closely examining the changes, valuable insights could be gained into how the algorithm modifies the structure, which could be used to refine and improve the initial setup, potentially enhancing performance in handling complex structures.

Another reason why DFL might not be the most suitable method for solving the scheduling problem is that, to handle uncertainty in the constraints, DFL requires a complete repair of the first-stage solution (the solution computed using the predicted costs) to determine the penalty term of the post-hoc regret. DFL assumes that the entire solution needs to be produced in one go and that the true values of the costs are revealed all at once. However, in scheduling and other complex practical scenarios, these reveals can happen independently and at different times, allowing for the solution to be computed in parts. For instance, in scheduling, if the uncertain processing time is caused by a complex biochemical reaction and multiple such processes are running throughout the day, the true processing time for each process is revealed upon its completion, not all at once as DFL assumes. Therefore, a potentially more suitable framework for solving real-world scheduling problems with uncertain processing times could be multi-stage optimization, where each stage computes only part of the solution, necessitating smaller recourse actions relevant only to a subset of all tasks. To illustrate the difference between this alternative approach and traditional DFL, consider a project scheduling problem where a PhD student needs to select projects to complete during their degree. Initially, it is challenging to precisely determine how much time each project will take and its value. Based on preliminary insights, the student can craft an initial plan. Applying DFL to this situation would yield a plan with reasonable time estimates that minimizes the risk of overlapping unfinished projects. However, in reality, midway through a project, the student might realize that it is not as valuable as initially thought and decide to stop working on it, or they might create a new project extending an existing one that was not considered in the preliminary planning. DFL struggles with such reconsiderations because its goal is to make predictions that obviate the need for changes. This is impractical in scenarios where changes are almost inevitable. Therefore, project selection problems are often viewed as sequential decision-making problems. With this in mind, a possible better fit for DFL when applying it in the real world might be to integrate it as a predictive part of the sequential decision-making process. The vision of having DFL as a multi-stage approach aligns with some of the recent work in the field. Hu et al. (2024) suggests that DFL should be implemented as a two-stage optimization problem, even for simpler problems. Their approach also highlights the weakness caused by the need for repair and thus envisions DFL functioning as a framework without any repairs, enhancing its applicability to scenarios with uncertain constraints. Pervsak and Anjos (2024) introduce a new Decision-Focused Forecasting framework. It is motivated by a similar argument about the fact that many real-world decision problems are recurrent and information about these problems unfolds over time, and that modeling them as multi-stage problems, where each set of decisions considers the most current contextual information, is often more effective. This framework represents the first application of DFL to multi-stage optimization problems. It achieves this by employing a recurrent predictive model. The evaluation of this framework demonstrates that it improves on conventional DFL in multi-stage settings, though it also reveals that the multi-stage approach incurs a considerable computational burden.

7.2. SFGE Methodology

The evaluation results in Section 5.4 revealed that DFL is generally more conservative in its predictions, potentially leading to better handling of uncertainty by avoiding penalties. This justified the use of Half-Normal distribution as a sampling distribution for the score function. An alternative approach to achieving this could involve employing quantile regression. By focusing on an upper quantile of the distribution rather than its mean, DFL could be tailored to predict higher values with greater precision. Quantile regression could be particularly useful in scenarios where the upper bounds of processing times significantly impact the overall penalty costs. Thus, future research could delve into the application of quantile regression within the DFL framework, comparing its effectiveness against traditional mean-based predictions.

Another limitation of the SFGE method is its potential sample inefficiency. This inefficiency can lead to slower convergence and suboptimal performance, especially in complex scheduling problems where the solution space is large. A prominent research direction, as outlined in Section 3.1, is to explore how to optimize the choice and initialization of the sampling distribution. Proper initialization can significantly impact the quality of the solutions found and the speed at which the algorithm converges. Additionally, investigating the impact of the number of samples could be highly beneficial. Currently, only a single sample is used when calculating the loss, which theoretically suffices for differentiation. However, Monte Carlo sampling with multiple samples could provide better statistical guarantees and more reliable gradient estimates. This approach could lead to more accurate and stable learning, as the variability inherent in single-sample estimates would be mitigated. Furthermore, adaptive sampling strategies, where the number of samples is dynamically adjusted based on the current state of the learning process, could be explored to balance computational cost and learning efficiency.

While storing and reusing examples in a replay buffer and performing PPO-like updates did not improve the overall performance or speed up the convergence of the method, the general idea of performing multiple model updates on a single data example could be revisited. This is motivated by the fact that the solving step is by far the most expensive one in the training loop, taking significantly more time than the updates. Inspiration for future ideas on how to reuse samples could come from the field of reinforcement learning, where replay buffers are a foundational component in many algorithms due to their ability to enhance data efficiency, stabilize training, and make efficient use of computational resources.

What is the future of DFL?

As a relatively new field, Decision-Focused Learning research faces a myriad of intriguing open questions. For instance, which problems are most effectively addressed by DFL, and is there a universal DFL approach that can be applied across all domains? While this thesis does not aim to definitively answer these questions, it provides valuable insights into DFL, the situations where it may face challenges, and how it could be improved. A tentative conclusion from this research is that DFL excels particularly well in optimization problems with simpler constraints and straightforward repair mechanisms. This simplicity allows DFL to function more efficiently and accurately. However, the true potential of DFL extends beyond these current applications. Exploring new avenues such as integrating DFL into multi-stage pipelines, developing hybrid approaches that combine DFL with structured optimization, and refining the current methodologies like the Score Function Gradient Estimation signal a promising evolution. However, these enhancements present new significant hurdles including increased computational burden, ensuring robustness in diverse scenarios, and effectively leveraging structural insights without sacrificing generality. If these obstacles are overcome, the advancements could revolutionize the landscape of decision-making frameworks, offering solutions to increasingly complex optimization problems and paving the way for broader, more impactful applications across diverse domains.

Bibliography

R. Arora, S. Arora, A.J. Kulkarni, and P. Siarry. *Combinatorial Optimization Under Uncertainty: Real-life Scenarios in Allocation Problems*. Advances in metaheuristics. CRC Press, Taylor & Francis Group, 2023. ISBN 9781003329039. URL https://books.google.nl/books?id=4gCrzwEACAAJ.

Aharon Ben-Tal, Laurent Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. 08 2009. ISBN 9781400831050. doi: 10.1515/9781400831050.

IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46 (53):157, 2009.

Emir Demirović, Peter J. Stuckey, James Bailey, Jeffrey Chan, Chris Leckie, Kotagiri Ramamohanarao, and Tias Guns. An investigation into prediction + optimisation for the knapsack problem. In Louis-Martin Rousseau and Kostas Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Lecture Notes in Computer Science, pages 241–257. Springer, 2019. ISBN 9783030192112. doi: 10.1007/978-3-030-19212-9_16. URL https://cpaior2019.uowm.gr/,https://link.springer.com/book/10.1007/978-3-030-19212-9.

Adam N. Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Manage. Sci.*, 68(1):9–26, jan 2022. ISSN 0025-1909. doi: 10.1287/mnsc.2020.3922. URL https://doi.org/10.1287/mnsc.2020.3922.

Xinyi Hu, Jasper C. H. Lee, and Jimmy Ho man Lee. Predict+optimize for packing and covering lps with unknown parameters in constraints. *ArXiv*, abs/2209.03668, 2022. URL https://api.semanticscholar.org/CorpusID:252118951.

Xinyi Hu, Jasper C. H. Lee, and Jimmy H. M. Lee. Branch & learn with post-hoc correction for predict+optimize with unknown parameters in constraints. In Andre A. Cire, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 264–280, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-33271-5.

Xinyi Hu, Jasper C.H. Lee, and Jimmy H.M. Lee. Two-stage predict+optimize for mixed integer linear programs with unknown parameters in constraints. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

Robert Klein. *Scheduling of resource constrained projects*. Kluwer Academic Publishers, USA, 2001. ISBN 079238637x.

Rainer Kolisch and Arno Sprecher. Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program. *European Journal of Operational Research*, 96(1): 205–216, 1997. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(96)00170-1. URL https://www.sciencedirect.com/science/article/pii/S0377221796001701.

Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2012. ISBN 3642244874.

Pierre L'Ecuyer. On the interchange of derivative and expectation for likelihood ratio derivative estimators. *Management Science*, 41(4):738–748, 1995. ISSN 00251909, 15265501. URL http://www.jstor.org/stable/2632893.

Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Víctor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *ArXiv*, abs/2307.13565, 2023. URL https://api.semanticscholar.org/CorpusID:260155160.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21:132:1–132:62, 2019. URL https://api.semanticscholar.org/CorpusID:195584180.

G.L. Nemhauser, A.H.G. Rinnooy Kan, and M.J. Todd. Preface. In *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*, pages v–ix. Elsevier, 1989. doi: https://doi.org/10.1016/S0927-0507(89)01001-7. URL https://www.sciencedirect.com/science/article/pii/S0927050789010017.

Egon Pervsak and Miguel F. Anjos. Decision-focused forecasting: Decision losses for multistage optimisation. 2024. URL https://api.semanticscholar.org/CorpusID:269982527.

Utsav Sadana, Abhilash Reddy Chenreddy, Erick Delage, Alexandre Forel, Emma Frejinger, and Thibaut Vidal. A survey of contextual optimization methods for decision making under uncertainty. *ArXiv*, abs/2306.10374, 2023. URL https://api.semanticscholar.org/CorpusID:259203793.

S.C. Sarin, B. Nagarajan, and L. Liao. *Stochastic Scheduling: Expectation-Variance Analysis of a Schedule*. Cambridge University Press, 2010. ISBN 9781139486385. URL https://books.google.nl/books?id=Mfz-V-tKnrEC.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

Mattia Silvestri, Senne Berden, Jayanta Mandi, Ali .Irfan Mahmutougullari, Maxime Mulamba, Allegra De Filippo, Tias Guns, and M. Lombardi. Score function gradient estimation to widen the applicability of decision-focused learning. *ArXiv*, abs/2307.05213, 2023. URL https://api.semanticscholar.org/CorpusID:259766730.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Bo Tang and Elias Boutros Khalil. Pyepo: A pytorch-based end-to-end predict-then-optimize library for linear and integer programming. *ArXiv*, abs/2206.14234, 2022. URL https://api.semanticscholar.org/CorpusID:250113469.

Kim van den Houten, David M. J. Tax, Esteban Freydell, and Mathijs de Weerdt. Learning from scenarios for repairable stochastic scheduling. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 234–242, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-60599-4.

Jeroen van Steijn. Predict+optimize for combinatorial optimization with uncertainty in the constraints. Master's thesis, November 2022. URL http://resolver.tudelft.nl/uuid:2a61bdf4-2899-420e-89fd-cf0f18ebdb25.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: decision-focused learning for combinatorial optimization. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33011658. URL https://doi.org/10.1609/aaai.v33i01.33011658.