Learning heuristics for a supply planner

Kevin Chong

Delft University of Technology

a REMIRA company

Learning heuristics for a supply planner

by

Kevin Chong

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday May 30, 2022 at 3:00 PM.

Student number:	4450116	
Project duration:	July 1, 2021 – May 30, 2022	
Thesis committee:	Dr. M. M. de Weerdt,	Delft University of Technology
	Dr. K. S. Postek,	Delft University of Technology
	Dr. C. Hemig,	Outperform
	E. A. T. Julien MSc	Delft University of Technology

Preface

This work is the last chapter in my journey as a MSc student at TU Delft. I would like to thank everyone that has made this experience so amazing to me.

I would first like to thank my supervisors, Professor Mathijs de Weerdt, Professor Krzysztof Postek, and Esther Julien. Thank you for providing me with this opportunity in the first place. Furthermore, thank you for the constant guidance and excellent feedback.

To my supervisor at Outperform, Claas Hemig, I would like to thank you for all your guidance. You have made me feel right at home when working at Outperform and I will be looking forward to continue working with you in the future. This gratefulness is extended to all the colleagues at Outperform, you made this journey really comfortable for me.

Lastly, I would like to thank my family and friends. I would not have been able to complete this thesis without their support. Thank you for being the rubber duck during my rants when something was not quite working as intended.

Kevin Chong
Delft, May 2022

Contents

1 Introduction

Supply chain management is a large and ever growing business, in 2020 its market value was already at $18.7 billion and it is estimated to increase to $52.6 billion by 2030 [30]. It is a large field that encompasses many components, ranging from forecasting sales at the supplier, to detailed planning of production and transportation of goods to the end customer. Many of these services have moved some of the manual labour from experts to automated processes. In this thesis we will focus on the component of supply planning, a crucial part of the supply chain management that ensures that we get our products to where the demand lies. Proper planning is crucial for a manufacturing company, lack of this can lead to increase in production cost, operational cost, and customer dissatisfaction [1]. The available inventory might not be enough to meet the demand of the customer, leading to empty shelves in the shop or last minute production at a higher production cost.

In situations where there is only a single product, a single location, and a single resource it can be relatively easy to find an optimal planning. However, when an additional product is added, conflicts can arise. Planning one product can lead to the other product missing out on a perfect opportunity for production. When an additional location is added, products could be shared between different locations to meet some demand and transportation needs to be considered. When different resources are added, choices need to be made between which resource to use for production as they may conflict. Especially if these resources have different properties such as the speed at which they produce or the efficiency they have. These problems in reality often have even more layers of complexity: multiple production methods, constraints on time, uncertainties in the input, and much more. Because of all of this supply planning is a difficult combinatorial problem.

Outperform has a framework to solve this problem with many constraints using an algorithm called the `Heuristic Planner` (HP). HP takes the products and sorts them based on their priority defined by the customer. Each product is then planned one at a time, by trying different possibilities. This algorithm uses heuristics to both improve the quality of the planning for the customer, as well as the performance of the algorithm. These heuristics are hand-tuned. HP is run often by the customers of *Outperform* to get planning for the upcoming duration. In each of the runs the input does not vary drastically. The demand and the available inventory will change, but other properties such as the products, warehouses, resources, and production methods usually do not see large changes.

Because of this similarity of the past and potential future planning problems, a question arises whether we can improve these heuristics, using information obtained from previous runs. More formally, the research question can be phrased as: can the execution time of the `Heuristic Planner` be decreased by learning new heuristics using ML, without decreasing the quality of the planning?

To answer this question, we will design and develop two different machine learning models that learn from past executions of HP. These models will be integrated into HP and evaluated on several data sets provided by *Outperform*.

This thesis combines the fields of **Combinatorial Optimization** (CO) and **Machine Learning** (ML). Combinatorial Optimization is a field in mathematical optimization that is interested in finding an optimal object within a finite set of possible objects. These sets are often relatively large, such that it is not feasible to exhaustively enumerate the objects within them. The purpose of the work in CO is therefore to find algorithms that can more efficiently find or approximate the optimal object. These solutions often require some hand-crafted heuristics and/or models to solve the problem, where the actual computation would be too expensive or too difficult to define mathematically. Finding a good heuristic is difficult and requires extensive knowledge of the problem at hand.

On the other hand Machine Learning is the field that studies algorithms that are trained on past data to make quick decisions in the future. The algorithms observe patterns within the data and uses those patterns to create new rules for its decisions. This process of creating the rules is automated and can result in complex models that have a great performance for its task.

The combination of these two fields has seen a lot of interest in the recent years, especially with the advancement in the ML field. There are two ways in which the techniques from ML can be used to improve the work in CO [4]. On one hand when expert knowledge is lacking or not existent, the work in ML can be used to fill in the gap by learning from experience. On the other hand even if expert knowledge is available, some difficult or expensive computations can be replaced with a fast approximation. A broad overview of recent works in using ML to gain advantages in CO can be found in [4].

1.1. Context

This thesis is done in collaboration with Outperform, a company that specializes in creating software to optimize the processes within a supply chain. The supply planning algorithm that will be used is called `Heuristic Planner`. The algorithm uses heuristics in several locations, out of which two were identified as possible candidates for improvement using machine learning.

1. The order in which the products are planned. Once the production for a product is decided, this is set in stone and the latter products will have to work around the decisions made for the previous products. This is the order of the products at the top of Figure 1.1.

2. For a given product, the order in which possible production or transportation methods are attempted. This determines the order of the possible solutions in the branch and bound tree of a single product. If a good solution is found earlier, then more parts of this branch and bound tree can be skipped. This is bottom part of Figure 1.1.

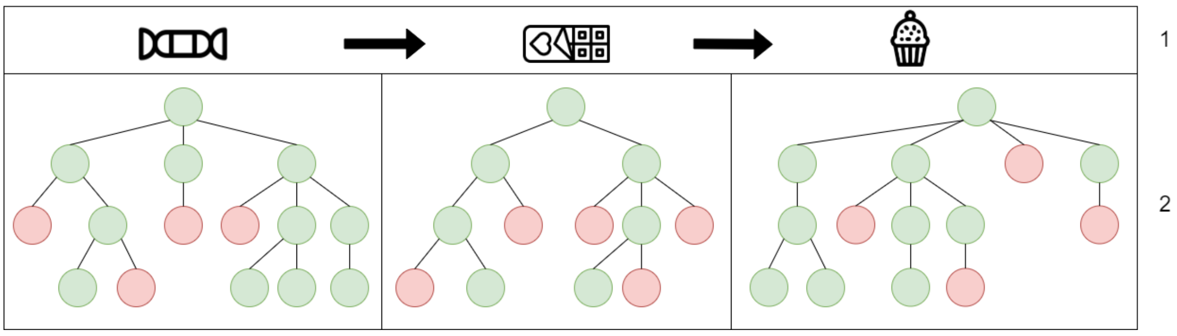


Figure 1.1: Heuristics in HP. At the top (1), the products are put in some order in which they will be planned. Below each product (2), each product will be planned using a branch and bound approach.

1.2. Problem Statement

We will focus on the sequence in which the products are planned. The reason for choosing this component over the alternative is because it was noted that the possible production or transportation methods for each product was often very limited. In which case ordering these methods will not lead to significant improvements in the performance of the algorithm.

The main objective of this work is to create a model that learns from previous runs of the Heuristic Planner to improve the performance of future runs. In this work we will only change the order in which the products are planned, no further changes will be made to the algorithm. This is because we want to investigate the possibility of learning a new heuristic. Making changes to the other parts of the algorithm would require additional experiments to evaluate the impact of these changes.

The most importance metric in this work will be the execution time of the algorithm, a lower execution time is desired. Additionally, the resulting plannings after our changes should have a similar or better quality compared to the plannings obtained with the current heuristic. The quality will be measured in the form of units of lost sales, as well as the amount of products that are below target on each day.

1.3. Contribution

In this thesis we evaluate two different models for learning a heuristic that decides which products should be planned first. The datasets used for these models are obtained from supply chains provided by `Outperform`. Slight alterations to the data are applied to a supply chain to create more varied instances of it. Additionally, a framework is made to enable more efficient data collection of prior problem solving experiences.

1.4. Structure

The structure of the thesis is as follows: in Chapter 2 relevant background information is provided and the problem is explained in detail. To which Chapter 3 presents the proposed models to find an improvement on the sequence for the products. The performance of the proposed models is then evaluated in Chapter 4. In Chapter 5 some directions for future work is highlighted. Finally, the results are summarized in Chapter 6.

2 Problem Background

In this chapter we will discuss the problem at hand, starting with defining the supply planning problem, then explaining the solution that *Outperform* provides for supply planning, and ending with what machine learning can change in that solution.

2.1. Supply Planning

> "A detailed formulation of a sequence of plan-tasks and actions associated with the plan-tasks. These plan-tasks and actions can be executed by agents in order to transform an initial-world-state into a goal-world-state. The output of the planning task is a plan specified in accordance with application-specific solution criteria." [26]

This is the definition that Rajpathak and Motta have given for planning, solving a planning problem entails finding the sequence of actions that brings one from the initial state to the goal state all while adhering to some constraints set by the problem and optimizing on some cost-function.

Planning problems have applications in many different fields, such as the medical field where planning is used to plan operations within a hospital [19, 34], the field of transportation where an example usage is to determine when and where trains should be on a network to deliver their cargo with minimal costs [12]. In this thesis we will focus on the field of supply planning, where planning is used to determine the optimal moment in time to produce certain items at a supplier to meet the demand at a customer. This field, similarly, has a lot of real world use cases, such as production at a vehicle manufacturer [29], food industry [28], or more recently the distribution of vaccines [14]. Interest in this topic came up as early as 1955 in the work of Modligiani and Hohn [22], where production planning is tackled using an approach that divides the horizon into successive periods of time that get solved to optimality.

Outperform works on the problem of supply planning. In their framework the planning problem is formulated as one that answers the following question:

> " How do we, given demands from customers on specific days in the planning horizon, create a plan that uses the available inventory at the start of the horizon and the available resources during the horizon to maximize the amount of fulfilled demand of the customers? "

The planning problem is about making sure that the inventory is kept at a level where incoming demand can be resolved. If the available inventory is always enough to meet the demand, then there is no need for planning. Unfortunately, this is not the case in reality, inventory does not appear magically out of thin air. So we need to plan for production or transportation to increase our inventory before the demand hits us.

There are a few questions that can be used to help in finding such a planning:

- **What** do we need to produce/transport?
- **How** do we produce/transport it?
- **When** do we produce/transport it?
- **How much** do we produce/transport?

The “what” in this question consists of two parts, the product and the location. This is to differentiate between the same product at different warehouse or factories, where the demand at each location can differ. The *combination of product and location* will be referred to as PL in the rest of this thesis.

Products	$P = \{p_1, p_2, ..., p_m\}$, this is a finite set of products that are included in the Heuristic planner.
Locations	$L = \{l_1, l_2, ..., l_n\}$, this is a finite set of locations that are included in the Heuristic planner.

“How” is answered by selecting a combination of a resource and a bill of materials (BoM) in the case of production. The resource can be seen as the person or machine that will spend time on the production, the BoM specifies what materials are needed and in what quantity. For transportation there is only a BoM that specifies the source and destination location.

Resources	$R_i = \{r_1, r_2, ..., r_k\}$, this is a finite set of resources that are included in the Heuristic planner, where i is the index of a location in L. A resource can be seen as a machine or tool used to produce a product. The resources each have their own capacity limits. Each resource is associated with a single location, whereas a location can have multiple resources. `resource`$(l_i) = R_i \subseteq R$, $R_1 \cup R_2 \cup ... \cup R_n = R$, $\forall 1 \leq i, j \leq n(R_i \cap R_j = \emptyset)$
Bill of Materials (BoM)	$B_p \in B_p^*$, where $B_p^* = B_p^{production} \cup B_p^{transport}$. $B_p^{production} \subseteq \mathcal{P}(P \times \mathbb{N}^+)$, i.e. set of combinations of products to produce product p with an associated quantity indicating how much of that material is required. $B_p^{transport} \subseteq \{(l_1, l_2) \mid \forall l_1, l_2 \in L, l_1 \neq l_2\}$, i.e. set of pairs of locations to transport product p between. A BoM B_p is either a production, defined as a list of products, or a transportation, defined as a tuple of locations.

The day on which the PL is produced or transported is the next question, the available days depend on the planning horizon.

Planning horizon	$T = \{t_1, t_2, ..., t_j\}$, this is a finite set of days that are included in the Heuristic planner.

The last question is related to the quantity that will be produced or transported. This quantity is given as a positive integer using stock keeping units (SKU).

Quantity	$\mathbb{N}^+$, the set of natural numbers without 0.

The combined answer of these questions provide an allocation, that is an assignment of production or transportation by defining clearly the product, the method, the time, and the quantity.

Allocations	$A = \{a \mid a \in P \times L \times R \times B_p^{production} \times T \times \mathbb{N}^+ \vee a \in P \times L \times B_p^{transport} \times T \times \mathbb{N}^+\}$. Each allocation is a 5-tuple or 6-tuple, depending on whether it is an allocation for transportation or production respectively. The common elements in the two types of tuples are a product, a location, a day, and a quantity. In the case of production two more elements are added to the tuple, indicating which resource is used together with which BoM. For transportation only the BoM of the transportation type is needed.

The solution to the planning problem is then a set of allocations that we want to include, describing all the productions and transportations needed during the planning horizon. This planning is one that should aim to minimize the amount of demand that can't be resolved, which in turn would lead to potential revenue being lost.

Plan	$\pi \subset A$, a plan is a set of allocations that have been included in the planning. Note that this is a strict subset of A as the cardinality of $Plan$ has to be finite.

A naive approach would be to simply include all allocations, ensuring that a maximum amount is produced for each PL. However, not all allocations are feasible at all times, so this idea would obviously not work. In the case of production, the allocations require materials and will occupy some resource for a certain amount of time. This leads to two constraints for an allocation to be included in the planning: 1) The required materials are available at the day of production and 2) the resource used for this allocation is available for the duration required. The required materials by themselves can be readily available in inventory or they could require additional production or transportation to be made available. Other than allocations being infeasible by themselves, the combinations of allocations can also be infeasible. The inclusion of one allocation in the planning can lead to changes in availability of materials as well as resources, when dealing with multiple products this can lead to allocations for other products being made infeasible.

Furthermore, out of all the feasible allocations, not all allocations are considered useful. Allocations are used to solve issues with the demand that can't be solved entirely by the available inventory. Skipping out on some beneficial allocations can lead to a higher amount of unmet demand, whereas including too many allocations can lead to overproduction. Overproduction is not ideal due to the additional costs of materials, production, and storage, as well as the limitation when dealing with perishable products. Instead, the idea is to only produce as much as is necessary and following the philosophy of just in time, only produce it right when it is needed. Only the allocations that contribute to this purpose are considered useful for the planning.

Lastly, the feasible and useful allocations can have a different degree of utility. If we require 800 units of a product and we are able to produce this amount in any way we would like, we would ideally produce it in a single batch of 800, rather than splitting it into several smaller batches. Similarly, if we are allowed to pick any day to produce it, we would like to be just in time, so not too early but definitely not too late. To be able to compare the utility of two allocations, a scoring mechanism is used. The score of a single allocation is computed as follows:

The score for a single allocation is computed by taking the an initial score of 100 and adjusting it using the following bonuses and penalties:

- Bonus if this allocation extends a previous one, i.e. it increases the quantity to be produced for an allocation that has not hit the capacity yet. This leads to less changeovers and a higher acceptance from the planners and staff on the work floor.

$$Be(a) = \begin{cases} 6 & \text{if allocation } a \text{ is extended} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

- Penalty for producing too early or too late.

$$P(t', t^*) = \begin{cases} 14 + (t' - t^*) & \text{if } t' > t^* \\ t^* - t', & \text{otherwise} \end{cases} \tag{2.2}$$

  - t' is the day currently being evaluated.
  - $t^* = max(\{t \mid t + c_t \leq t_{last}, t \in \{0, ..., t_{last}\}\})$. This is the optimal day to supply also called "Just in time" (JIT).
  - $t_{last} = max(\{t | t + h_t + 1 \leq d, t \in \{0, ..., d - 1\}\})$. This is the last day where supply could be created to meet the demand.

  - d is the day with the unmet demand.
  - h_t is the holding days on day t, i.e. products produced/transported on day t will require h_t days before they are usable.
  - c_t is the number of days to cover at day t, a cover of N days implies that the inventory should cover the demand for d as well as $N-1$ days before d. Essentially making the demand required $N-1$ days earlier than d. However, this is not a hard constraint, solving the unmet demand without complying to this cover is better than missing demand.

- Penalty for producing a different amount than the desired quantity, i.e. the amount of unmet demand.

$$P(q', q^*) = \begin{cases} \lfloor 5 * (1 - \frac{q'}{q^*}) \rfloor & \text{if } q' < q^* \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

  - q' is the quantity currently being evaluated.
  - q^* is the desired quantity.

- Penalty for using an allocation that is marked as “should be avoided”.

$$Pa(a) = \begin{cases} 10 & \text{if allocation } a \text{ should be avoided} \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

The final score of a single allocation is then:

$$S(a) = 100 + Be(a) - P(t', t^*) - P(q', q^*) - Pa(a) \tag{2.5}$$

2.2. Heuristic Planner

In the previous section we have seen what the supply planning problem looks like, to solve it there are many ways to do that. Guzman et al. [15] conducted a review of the existing literature on the topic of production planning, scheduling, and sequencing problems. They have identified four larger categories of approaches to solve the planning problems used in literature:

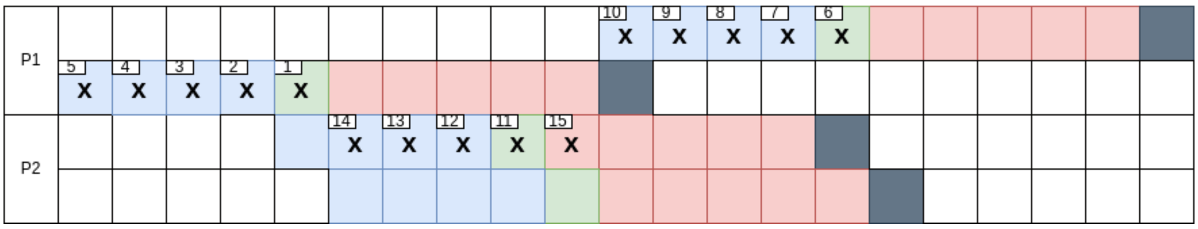
Optimiser Algorithms include the techniques that guarantee to return solutions that are optimal. An example of such an algorithm is one that uses branch and bound.

Heuristic Algorithms do not have the guarantee for an optimal solution, instead it uses heuristics that are often problem specific to find a relatively good solution.

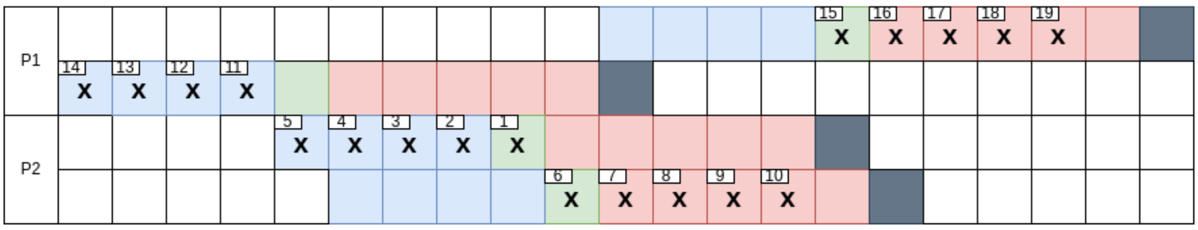
Metaheuristic Algorithms similarly does not guarantee that an optimal solution is returned. Compared to heuristic algorithms metaheuristic algorithms are problem-independent and provide a framework that can be applied to solve multiple problems. Genetic algorithms are a popular example of metaheuristic algorithms.

Matheuristic Algorithms combine the approaches from metaheuristics with mathematical programming, where a mathematical model is created that can be solved using techniques such mixed-integer programming (MIP). Similar to metaheuristic algorithms, the techniques found in this category are often frameworks that can be applied to different problems. An introduction to these algorithms can be found in [10].

The planning algorithm from *Outperform* is called the `Heuristic Planner` (HP) and falls under the category of Heuristic Algorithms. Several heuristics are used in this algorithm to find plannings that resolve as much demand as possible, while keeping the execution time of the algorithm feasible. To better understand the areas in this where heuristics play a role, we first need to walk through the idea of the algorithm. HP takes a single PL and finishes the planning for it before moving on to the next one. The order of these PLs is important, as the earlier PLs will have more feasible allocations available, whereas the latter PLs are getting more and more restricted. An example of this can be seen in Figure 2.1. Each column represents a single day in the planning horizon and each row represents a case of unmet demand. The grey shaded squares indicate the day of the demand, the green squares indicate the JIT days for the different deadlines, the blue squares are the early production days, whereas the red squares are the late production days. The numbers in the corner of the squares indicate the order in which HP includes the allocations. In this example we have two products to consider and there is a single resource, the demands are all of equal size and require 5 days to resolve.



(a) Product 1 first, 5 units of production are missing



(b) Product 2 first, 1 unit of production is missing

Figure 2.1: Planning of two products

This is the heuristic that we will focus on in this thesis, to be able to find sequences that result in a faster execution time of HP. In the rest of this section we will continue with explaining how HP and the other heuristic work, in section 2.3 we will discuss the PL sequence in more detail.

Once the sequence of PLs is decided, the first PL can be planned. Planning a single PL is done by finding the days on which the demand is strictly higher than the available inventory, starting with the earliest day with such a problem. A search is made for an allocation that can resolve this demand fully or partially, the allocation with the highest score is selected first. If the demand is resolved fully, the algorithm moves on to the next day with unmet demand. If the demand is only resolved partially, another search is made for the next allocation but with the updated state induced by the previously selected allocation.

Out of the 4 questions that need to be answered for an allocation, the question related to what we need to produce is already set in stone. It is the current PL with some unmet demand, this leaves the “how”, “when”, and “how much” left to be answered. The naive idea is to try all possible combinations and then pick the successful allocation with the highest score, see Algorithm 1. By repeating this procedure for the entire planning horizon of this PL, we can find the allocations used in the planning.

The naive approach, however, is overly time consuming if the search is done over all possibilities. Additionally, the success of an allocation depends on the feasibility of it in the current situation. Two conditions were defined previously for an allocation to be feasible, 1) the required materials should be available, and 2) the selected resource shouldn’t have its capacity exceeded. Checking the second condition is relatively straightforward, the desired quantity in this allocation combined with the existing load on the resource should not exceed the capacity. The first condition, on the other hand, requires more work. Simply checking whether the available inventory for each material is not sufficient, production or transportation can increase the available inventory for these materials as well. So, if the available inventory for this material is not sufficient for the allocation, we can treat the deficiency as a new demand for that material that needs to be resolved in a similar manner as the usual PLs. If the deficiency can be resolved and all the other materials are in the same situation, that there is either enough inventory or it can be resolved, this allocation is feasible. If a single material is not resolvable, the allocation is infeasible. This process can be be repeated many times for larger supply chains where

Algorithm 1 Finding one allocation - Naive

```
    Input: $t'$ - the day of unmet demand, $q'$ - the amount of unmet demand
    Output: $a$ - the chosen allocation
1:  procedure FindOneAllocation
2:      $bestScore \leftarrow -\infty$
3:      $bestAllocation$
4:      for $t = 0$ to $t' - 1$ do
5:          for $q = q'$ to $0$ do
6:              $allocations \leftarrow$ all allocations that match the current product, location, day, and quantity
7:              $isPossibleQuantity \leftarrow false$
8:              for $a$ in $allocations$ do
9:                  $success \leftarrow TryAllocation(a)$
10:                 if $success \wedge Score(a) > bestScore$ then
11:                     $bestScore \leftarrow Score(a)$
12:                     $bestAllocation \leftarrow a$
        return $bestAllocation$
```

a lot of products and their materials are entirely produced in house.

As checking a single allocation can already be time consuming, we want to minimize how often this needs to be done by skipping unnecessary allocations where possible. For this purpose we can use the score of the allocations. The search process for an allocation of a single PL can be visualized as a tree with a maximum height of three.

Example:
In Figure 2.2 the first layer of the tree branches on the day (“when”), in the second layer the quantity (“how much”), and in the last layer the resource and BoM (“how”). The leaves are the allocations using the values found on the path from the root. This tree can be further expanded above this maximum height when dealing with products that have missing materials, in which case the leaf nodes would be the roots of subtrees for the missing materials.

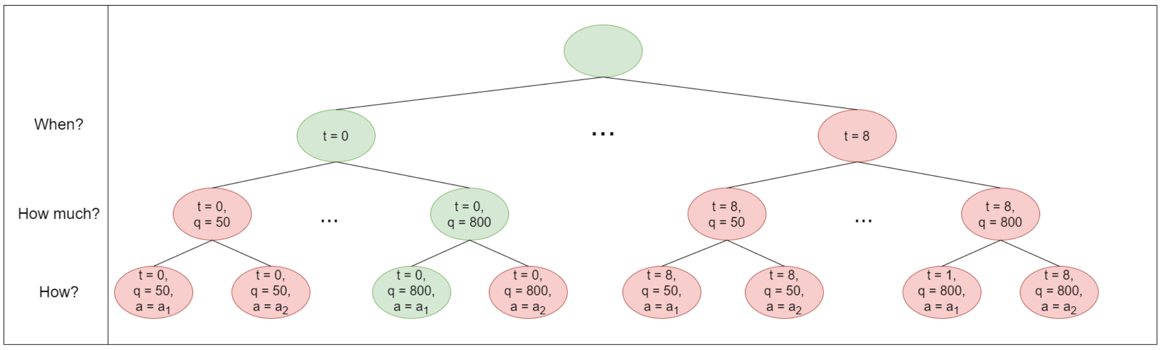


Figure 2.2: Search Tree for HP, the green path leads to an allocation that is selected. The red nodes do not need to be considered after the green node, due to the score of the green being higher.

Using a similar approach to branch and bound algorithms, an incumbent solution or in this case an allocation needs to be found first. This incumbent will then be used to fathom branches where the most optimistic computation of the score would lead to a strictly lower score than the incumbent. This can be computed at any layer of the tree, even when branching on the day for example. The highest possible score for that branch can be simply computed by taking the actual penalty for the day, but no penalties for the other components. With this, once we have found a good incumbent, the rest of the search can be completed relatively fast.

To speed up finding the first incumbent HP orders the branches in each layer based on the potential score that the decision made in that layer can lead to. For example, when branching on the day, the best day would be the day considered JIT, which has no penalty. Whereas the other days induce a penalty based on how early or how late they are from this JIT day. Similarly, the quantities are attempted by checking the desired quantity first. When none of the allocations with quantity are feasible, a binary search like approach is used to find the best quantity possible. Lastly, the allocations are ordered by the production rate of the resource, with the idea that those allocations can finish the quantity in less time. The expanded algorithm can be found in Algorithm 2.

Algorithm 2 Finding one allocation

```
    **Input:** $t'$ - the day of unmet demand, $q'$ - the amount of unmet demand
    **Output:** $a$ - the chosen allocation
 1: procedure FindOneAllocation
 2:     $bestScore \leftarrow -\infty$
 3:     $maxScore \leftarrow 106$
 4:     $ordinals \leftarrow [0, ..., t' - 1]$
 5:     Sort $ordinals$ based on penalties (Equation 2.2) in ascending order
 6:     for $t$ in $ordinals$ do
 7:         $maxScoreOrdinal \leftarrow maxScore$ – penalty for this day
 8:         if $bestScore \geq maxScoreOrdinal$ then
 9:             continue with next day
10:         $minQ \leftarrow 0$
11:         $maxQ \leftarrow q'$
12:         $q \leftarrow q'$
13:         while $q > minQ$ do
14:             $maxScoreQuantity \leftarrow maxScoreOrdinal$ – penalty for this quantity (Equation 2.3)
15:             $quantitySuccess \leftarrow false$
16:             if $bestScore < maxScoreQuantity$ then
17:                 $allocations \leftarrow$ all allocations that match the current product, location, day, and quantity
18:                 Sort $allocations$ based on fastest production rate
19:                 $isPossibleQuantity \leftarrow false$
20:                 for $a$ in $allocations$ do
21:                     $success \leftarrow TryAllocation(a)$
22:                     if $success$ then
23:                         $quantitySuccess \leftarrow true$
24:                         if $Score(a) > bestScore$ then
25:                             $bestScore \leftarrow Score(a)$
26:                             $bestAllocation \leftarrow a$
27:             if quantitySuccess then
28:                 $minQuantity \leftarrow quantity$
29:                 $q \leftarrow (maxQuantity + minQuantity)/2$
30:             else
31:                 $maxQuantity \leftarrow quantity$
32:                 $q \leftarrow (minQuantity + minQuantity)/2$
        return $bestAllocation$
```

While the time and quantity are ordered by using the scoring mechanism as a guideline, the allocations use the rate of production which is not included in the scoring mechanism. A different heuristic for ordering these allocations can guide the algorithm to allocations that are successful earlier on, whereas those that would likely be unsuccessful would be attempted last.

2.3. Product Location Sequence

In the rest of this thesis we will focus on the PL sequence as the heuristic that we try to improve using machine learning.

Each PL is assigned to a priority group, products in a higher priority group will always be planned first. This makes it more likely for higher priority PL to be able to find satisfiable allocations, whereas the lower priority ones would have more demand that can't be met. These assigned priority groups are obtained from user input and can't be changed by the algorithm, even if swapping two PLs between priority groups would have resulted in a strictly better planning. The order for PLs with the same priority is defined using the *length of a PL's supply chain* called the network level, an example of this is given in Figure 2.3.

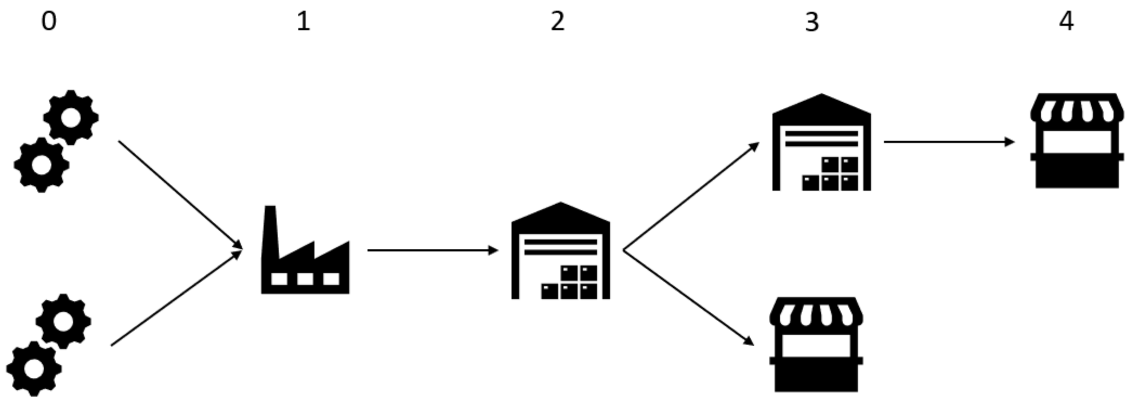


Figure 2.3: Example supply chain with network level of each stage on top. Materials at the start of the chain have the lowest network level, whereas the stores at the end of the chain have the highest.

The possible sequences can be represented in a multi-valued decision diagram (see Figure 2.4), where each path represents a possible product location sequence for HP. Starting at the "root" node r, a first choice needs to be made between the PLs with a priority of 10. Each branch is a different choice for the first PL to be planned. Depending on the first choice, the next branches only contains the remaining PLs. After the three PLs with priority 10, we move on to the PLs with priority 0, where the same process is repeated. Multi-valued decision diagrams are used in the literature to find ways to simplify the problem by merging possible branches into one using some generalization. For example, ...

In this case, nodes can't be merged because of the dependency between them. Each node represents the action of planning one PL, the allocations resulting from this decision will change the state of environment with respect to availability of resources and materials. The action of planning $(p_1, p_2, p_3, p_4, p_5)$ (green) differs from $(p_1, p_3, p_2, p_4, p_5)$ (blue), such that the nodes for p_4 and p_5 are not equal in both paths and can't be merged. Despite the order in that priority group not changing at all.

However, this is only the case if two PLs are dependent on each other, if none of the allocations chosen for the two products conflict with one another then the products can be considered independent of each other. In which case, the order of the two PLs will not matter and the two paths could be merged.

To check whether two PLs are independent of each other, we can use an undirected graph. Each PL is a node and there is an edge between two PLs if they share at least one resource or at least one material for production, see Figure 2.5 for an example. If there is an edge between two PLs they are related, this relation is transitive, which means that if item a is related to item b, and item b is related to item c, item a must also be related to item c. The connected components within this graph show the groups of related items, which means that the order for these PLs can be important, whereas the order between two PLs in different connected components isn't.

This notion of related PLs can be extended further by only creating edges when the shared resource or material can be a bottleneck. If a material has a sufficiently high or infinite supply, planning one PL that uses this material will not affect the other PLs. Similarly, if the demand for the PLs that use this resource is smaller than the maximum amount that can be produced with the capacity available on this resource, then there is again no issues that can result from planning one PL. However, to find out whether this is the case for each PL is more time consuming and in some cases a definite conclusion can only be drawn by solving the planning problem partially.

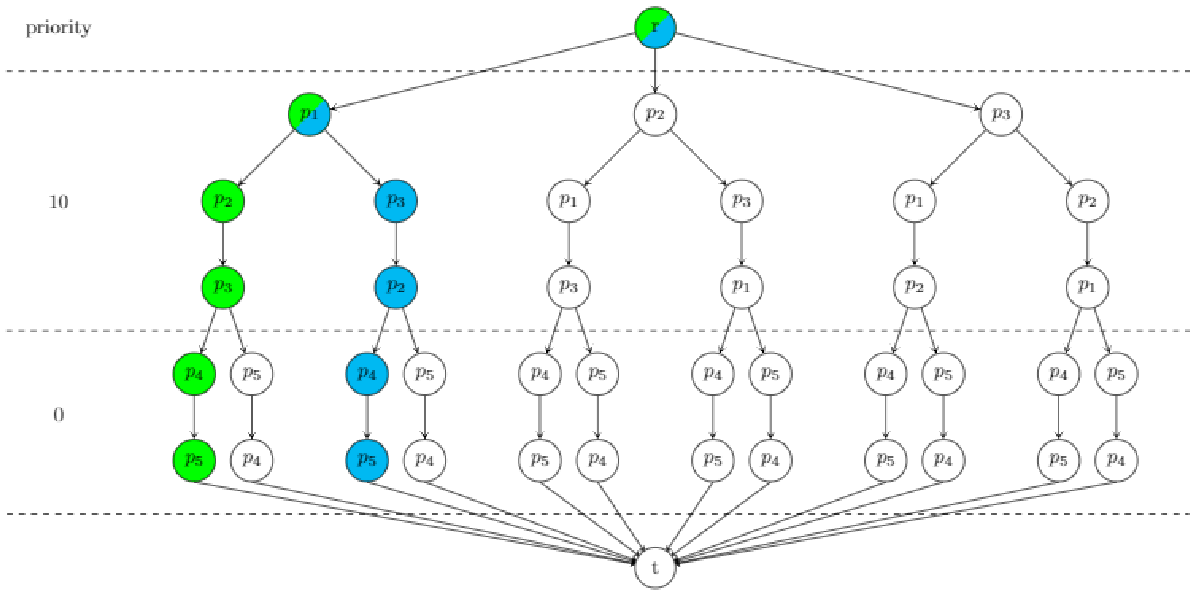


Figure 2.4: Decision Diagram, each level represents a decision where the node states which product is planned. The dashed lines divide the priority groups, as well as the starting (r) and ending (t) nodes.

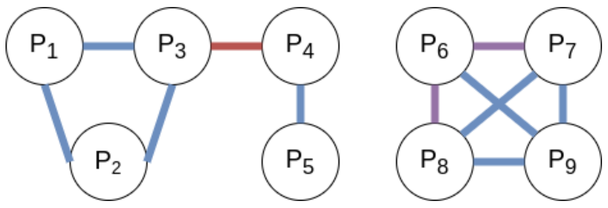


Figure 2.5: Connected Component, each node is a single item in the planning. The blue edges show a connection between two items that share a resource. The red edge shows a relation between two items that require a common product as material. The purple edges indicate that both relations hold.

2.4. Machine Learning in Planning

Different combinations of planning problems with learning approaches are discussed in an early survey of Zimmerman and Kambhampati [35] back in 2003. In their work they describe several dimensions along which an automated planning system augmented with learning capabilities can be categorized.

The first dimension defined is the type of planning problem, where the distinction is made between classical planning and full-scope planning. In classical planning the state of the environment only changes when an action is done, the state is fully observable, and actions are deterministic and instantaneous. In full-scope planning we have more uncertainties in the form of stochastic actions, but also a dynamic and only partially observable environment.

The second dimension that is defined is the approach in which the planning problem is solved, Zimmerman and Kambhampati split these approaches into two larger categories, search processes or model checking. For the search processes both state-space search and plan-space search approaches are considered. In state-space search, an initial state is assumed after which actions or more generally operators are applied to reach new states, with a final objective of reaching a goal state [2]. In plan-space search instead of searching through these states, the algorithms would search through partial plans. On each step the current partial plan is refined, until all conditions are met [25].

The third dimension relates to the goal of the learning component, whether this is to improve the quality of the plan, to speed up the planning, or to learn more about the environment.

The fourth dimension describes in which phase the learning takes place, before the planning, during the planning, or during the execution of the plan.

The last dimension defines the type of learning approach used, the distinction is made here between analytic learning techniques and inductive learning techniques. In analytic learning techniques the domain theory plays a crucial role in the learning process, it consists of prior knowledge that can be used during learning. This includes information such as what effects certain actions will have, as well as which actions are valid. Analytic learning techniques such as explanation-based learning can learn a model using a single instance. This is done by inferring rules that align with both the domain theory and the instance used for training. The domain theory in this case needs to be correct and expansive to ensure that the rules are correct for the other instances as well.

Inductive learning techniques are the widespread modern machine learning techniques, such as decision trees, neural networks, and reinforcement learning. These techniques learn from examples and do not necessarily require the domain theory to function.

Using the dimensions defined by Zimmerman and Kambhampati, we can describe the combination of machine learning and planning problem that this thesis deals with. It is a classical planning problem, where the solver uses a state space search, and the goal is to speed up the planning. This leaves two dimensions left, the learning phase and the type of learning approach, see Figure 2.6. For the learning phase, the option of learning before the planning starts implies only using information available about all PLs that are to be planned, as well as the current availability of the resources and materials over the planning horizon. Whereas the other two options allow for changes to the information provided after each product is planned. Those two options are in the case of HP the same, as HP creates the planning and executes it at the same time. HP inherently does not allow for backtracking after one PL is planned, allocations chosen for the previous PLs are final.

For the type of learning, we have chosen for inductive learning. Analytic learning can be used when data is scarce, the domain theory is then used to “substitute” for the lack of the data. Analytic learning requires more prior knowledge, knowledge about how the order of PLs affect the resulting planning. This knowledge is not available.

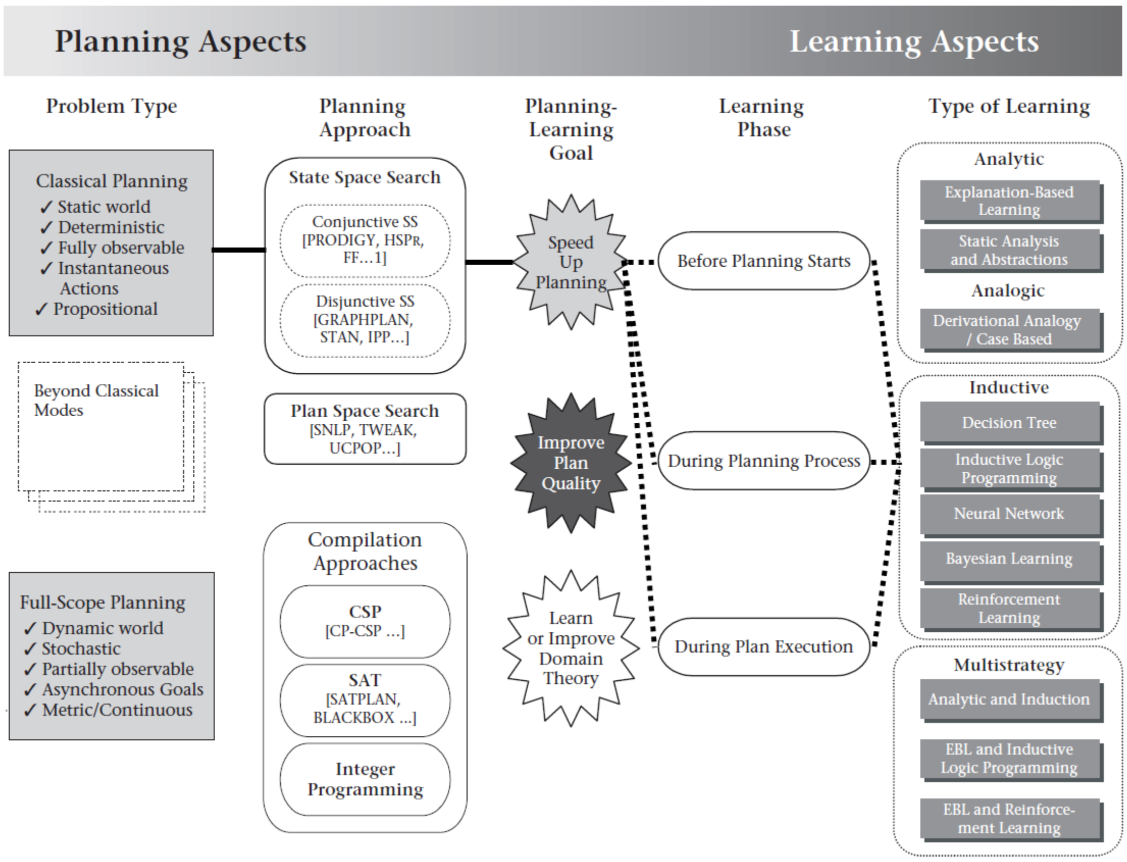


Figure 2.6: Aspects of learning in planning from [35], the black line between dimensions are the options that are set in stone by the current problem, the dashed lines represent possible options for the solution.

2.5. Speedup Learning

Learning to speedup algorithms, or better known in the literature as speedup learning is the field where machine learning is used to improve the performance of problem solvers.

This idea is not new, the concept dates back to Michie in 1968 [21], Natarajan and Tadepalli introduced a formalized framework in 1996 [31]. Fern [8] gives a broader overview of the field of speedup learning, he describes it as a problem where we take observations of prior experiences to train a learner to output knowledge that a solver can exploit to find solutions more quickly. In his work he makes a distinction between several dimensions of speedup learning.

A distinction can be made between offline and online learning. In offline learning the model is trained on data collected across multiple instances of the problem to learn a generalized concept that can be used for problem instances it hasn't seen yet, which is why this type of speedup learner is also known as inter-problem speedup learner. In online learning, on the other hand, the learning takes place at the same time as the solving happens. It uses the information obtained from the current instance to learn specific actions it should take as a next step while solving the instance. The learned model is generally only used for the instance it was trained on, for this reason this approach is also known as intra-problem speedup learner.

Another distinction made is in the type of learning process. In deductive learning new rules are learned through the use of previously known facts, these rules can always be shown to be correct. This can be seen as taking existing rules and refining them. In inductive learning on the other hand, the learner tries to find new rules by looking at provided data and finding patterns within them. These rules do not have the guarantee to always be correct, it is only an approximation that tries to fit the provided data as much as possible.

Lastly, the difference in the type of knowledge learned is also important. Fern notes that most problem solvers in this domain can be viewed as search procedures, for which four types of commonly used knowledge that can be learned are:

Pruning Constraints	inform a solver which parts of the search space can be ignored. Similarly to the process of fathoming branches in a branch and bound algorithm, these constraints can speed up the search process by ignoring larger parts of the search space using the knowledge it has obtained during the seach process.
Macros Operators	are common sequences of search operators that are used, where an operator is used to dive deeper down the search space. An example of such an operator in the planning problem would be planning a single product. In some cases certain operators are commonly used together in a specific order. Having such a sequence of operators as a single operator can reduce the amount of time spent in the search process by reducing the action of choosing the specific operators to a single step. Essentially, the knowledge learned here would a certain generic patterns used in the search procedure to reduce the amount of steps required in the process.
Search-Control	dictates how the search process should be continued at each step. It can for example decide to change the approach from a depth-first search to preferring specific nodes in the search tree.
Heuristic Evaluation Functions	measure the quality of search nodes. The quality can then be used to steer the search process using the learned heuristics. An example of such an evaluation function is the scoring mechanism in HP.

In this work we will focus on offline learning or inter problem learning rather than online learning. Online learning has the disadvantage that the early steps while solving can be poor, in hope that once the model has learned more the decisions in the latter steps will compensate for this. With HP this is not a viable option, as a choice will need to be made for which product to plan first, making a poor choice here could already lead to unresolvable issues for the remaining products.

For the learning process we will use inductive learning, where we take previous examples as input data and learn a rule that acts as a heuristic for ordering the PL sequence.

Lastly, the type of knowledge we want to learn falls under the category of heuristic evaluation functions. The current order of the PL sequences can be seen as a ranking based on the priority group and network level. The objective is to improve this ranking, the resulting model would act as an evaluation function for the PLs.

In this work we will use imitation learning, by taking PL sequences that resulted in a relatively fast execution time and using those as training data for our models. Imitation learning is the idea of teaching a robot to mimic human behaviour, this is done by learning a mapping between observations and actions [17]. This field is gaining more and more traction because it allows for teaching without providing deeper knowledge of the problem or a sophisticated model to be made.

An example of imitation learning being used for speedup learning can be found in the work of Pan and Srikumar [24]. They learn a speedup classifier that acts as a heuristic to replace a more expensive model. In their work they consider two models to work side by side while training, one model is an already trained black-box classifier that is expensive, whereas the other model is the inexpensive speedup classifier. The speedup classifier is trained to mimic the black-box classifier by using the black-box classifiers past decisions as its training data. This approach is related to the idea of imitation learning, where a model is trained to mimic a reference policy, with as goal to achieve similar results with a lower cost or possibly better policies using information learnt over multiple scenarios. The expensive model serves as an oracle that provides accurate answers as training data. Namaki et al. [23] similarly apply imitation learning to improve the query plans for a state of the art graph query reasoner. In their work an oracle is also included, this oracle is able to compute query plans of high quality that can be used as training data for the imitation learning. The DAgger algorithm [27] is included in the process to further improve the model's performance past the initial training setup. DAgger includes the predictions made by the trained model as training data for the next prediction, this way the model is able to learn from its own past mistakes. In both works the oracles play an important role, it provides the input data required for training. As the models try to imitate the actions made by these oracles, if the actions result in a bad solution, the trained model will likely suffer from this as well. Out of these two works, the second one is more aligned with the circumstances of our problem, as we similarly do not have access to a perfect oracle.

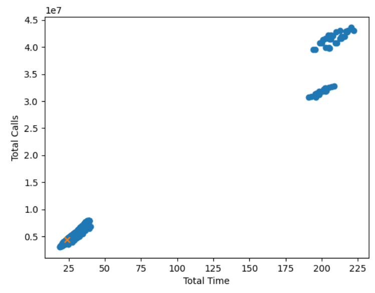
A different example of a speedup learning problem solved in literature can be found in the works of Domshlak et al. [7]. In this work imitation learning is not used, but they deal with the problem of deciding which heuristic to use. They speed up a heuristic state-space search algorithm by using machine learning to choose the optimal heuristic to compute at each state during the planning process. When using algorithms such as A*, the heuristics play a crucial role in determining the performance of the algorithm. These heuristics are problem dependent and there is no single heuristic that is optimal in all situations, picking the correct heuristic is important. Combining multiple heuristics could alleviate this problem with the downside that at each step more computations need to be done, which could be detrimental. Domshlak et al. solve this by learning a model that is used as a classifier to predict whether it is worth to compute a more expensive heuristic at that step or not.

A different field that could help in speeding up algorithms is through the use of surrogate models. With surrogate models a complex computation can be replaced by a trained model that can accurately approximate the resulting value from the complex computation in a relatively fast time, the speedups in this field are often in the orders of magnitude [9, 18, 6]. This approach is especially relevant if there is a single bottleneck in the process that you would like to speed up. This, for example, could be applied to the problem at hand to approximate the resulting execution time of HP with a certain PL sequence. This way you could attempt to find better sequences through this surrogate model.

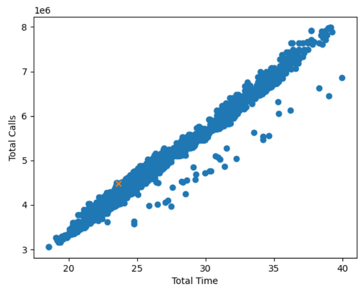
In this work we will follow the approach of Namaki et al. [23] to use imitation learning with an oracle to fill in the gap of the expert. By finding better PL sequences and using that as the training data for our model, we should be able to imitate these PL sequences and obtain better results than the baseline of using the current heuristic.

2.6. Planning Evaluation

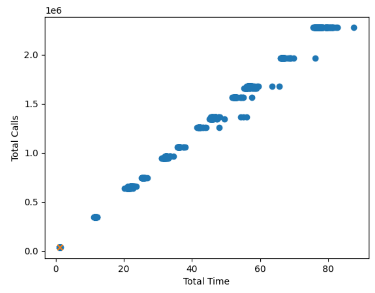
The resulting plannings can be evaluated in two different dimensions, quality vs time. Quality of a planning is measured by two metrics, the number of lost sales and the number of times the inventory goes below a target. Time can be measured as the amount of time required to run the algorithm. However, an approximate will be used instead to make the measurements less prone to noise. This approximate is the number of calls to a function in the last stages of checking an allocation. This function will be called for all possible allocations that haven't been pruned beforehand. In Figure 2.7 it can be seen that this indicator has a roughly linear relation to the time spent running the algorithm. More formally, the correlation coefficient between the time spent running the algorithm and the number of function calls is visualized in Figure 2.8. The histogram contains data of 80 different instances, where each instance has 250-300 samples. In most instances it can be seen that these two values are highly correlated with one another.



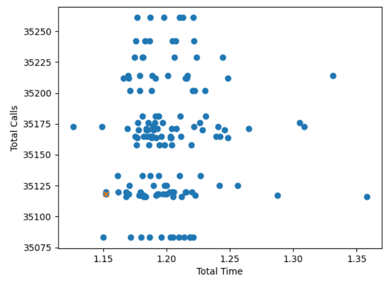
(a) Instance A: All sequences



(b) Instance A: Zoomed in on the first cluster



(c) Instance B: All sequences



(d) Instane B: Zoomed in on the first cluster

Figure 2.7: Total Time (Performance) vs Total Calls (Performance), each data point is a a run of HP with a different product location sequence

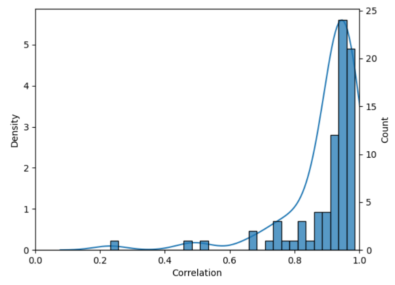


Figure 2.8: Correlation between Total Time and Total Calls

3 Methodology

In ML the process usually follows the same procedure:

1. Collect data
2. Clean/prepare data
3. Train a model
4. Evaluate the model
5. Use the model

In this section we will discuss the first three steps in this procedure. We will discuss the two models designed to learn which PL sequence leads to faster execution times, as well as an initial model created while studying the problem.

First we will start with discussing process of collecting data. Depending on the problem, historic data might be available that can be used for training after some preparation. If this isn't the case then data will need to be collected first, during which we need to think about what features we want to use for our models. But before talking about the features, we need to know what the aim is for our ML models. The models all follow the idea of imitation learning. Imitation learning is not an algorithm or model by itself, it is rather a collection of different techniques that follow a similar approach. The goal is to learn and mimic the behaviour of an expert that is demonstrating their actions on a certain task. Examples of imitation learning in practice can be found in [17], the use cases include fields such as self-driving vehicles and assistive robots, as well as use cases in the field of combining ML and CO [16, 33].

Attia et al. [3] have given an overview of some of the most used approaches for imitation learning. One of them being supervised learning, where the expert's actions create labels for the training data that the model can then use. Often this data consists of a sequence of observations as well as a sequence of actions that the expert has taken. The trained model will then need to pick the same action as the expert when given the same observation. Attia et al. point out that this approach lacks the ability to recover from failures, that is once an incorrect action has been picked during the sequence, the model will not be able to adapt to this. Other algorithms have been mentioned in their work that aim to solve this issue. In this work we will focus on the simpler approach of supervised learning. The reason for choosing this is because in HP once a poor choice is made for a PL, this decision is set in stone. HP itself has no way to recover from this, unlike the more common use cases of imitation learning such as self-driving vehicles.

With this approach of using supervised learning to do imitation learning, we aim to train the models such that they are able to give optimal PL sequences with respect to the execution time of HP. The training data will, thus, need to contain the sequences that improve upon the current heuristic, but ideally it would contain the optimal sequences for the planning problem at hand. This leads to the first step in the process, data collection.

3.1. Data Collection

Finding an optimal sequence is certainly non-trivial, as otherwise, one could simply use that approach before running HP. The search space for these sequences depends on two factors, the number of PLs and the number of priority groups. The number of possible sequences in the search space is equal to $\prod_{k \in P'} |k|!$, where P' is the set of priority groups, each priority group in itself is a set of PLs. In the worst case, if there is only a singly priority group, the number of sequences is simply $n!$, where n is the total number of PLs.

Example:
Given an instance with 3 priority groups and 5, 3,and 4 PLs in each priority group respectively, the size of the input space will be equal to $5*4*3*2*3*2*4*3*2 = 17280$ possible sequences in total.

Evaluating all sequences is generally not advisable, for the example given if each sequence requires 10 seconds for its evaluation, the time required would be exactly a full day. The instances used in practice are often much larger and thus have a larger search space but also require more time to evaluate a single sequence.

To give a better idea of what this search space looks like, all possible sequences for different instances are visualized in Figure 2.7. The circles represent different (groups of) PL sequences, whereas the orange x is the PL sequence obtained from the current heuristic. It can be seen that the current heuristic is often much better than a lot of other random sequences, but there is definitely still room for improvement especially in cases such as instance A.

3.1.1. Equal Sequences

There are some ways to reduce the possible number of sequences that need to be evaluated, one such way is to use the notion of related PLs introduced in section 2.3. When two PLs are not related, the order between these two does not matter. Only the order of the PLs that are related to each other matters. As an example we take the the PLs shown in Figure 2.5, with a more detailed version in Table 3.1. Using those PLs, 4 example sequences are provided that are equal. The orders for each PL within a group of related PLs stays consistent, thus the results of running HP with any of of the 4 sequences will be the same for the others. Because each group of related PLs does not have any relations with the others, it is possible to treat each of those groups as a separate sequence. For the example given, you could run HP twice, first with the sequence containing the PLs of group A and then a second run with the PLs of group B. Combining the results will have no conflicts and will give a complete planning. The alternative is to simply group up the related PLs within a priority group, avoiding the situations in sequence 2 and sequence 4 of the example, where unrelated PLs are intertwined within a priority group. Additionally, adhering to a strict order in which group of related PLs to plan first in a priority group will reduce these equal sequences to a single sequence to be evaluated.

Depending on the supply chain, this can drastically reduce the number of possible sequences. To give an example, one of the datasets used in this work contains 52 PLs spread over 7 priority groups. The total number of possible sequences is roughly 3.6 million, from which only 1440 sequences remain after removing all sequences that are equal. However, in the worst case if all PLs are related, no sequences can be eliminated with this approach.

3.1.2. Local Search

Despite reducing the total amount of sequences that need to be evaluated, it is still not feasible to enumerate the remaining sequences to find our candidates for the training data. To speed up the process of finding such candidates in a reasonable amount of time, we use the observation that the current sequence obtained from the heuristic is relatively good as can be seen in Figure 2.7, but in the neighbourhood of this sequence there are still better options. To find better sequences we use a local search approach. The idea is to make small changes to the current sequence to create a new candidate, this candidate will be evaluated and depending on the result we will continue with this candidate as the current sequence or discard it, see Figure 3.1. By repeatedly taking the best sequences found and continuing with those, the process should take us to better sequences in the search space.

The process for local search can be found in Algorithm 3. The idea is to take the PL sequence obtained from the current heuristic as the first sequence evaluated in the search space. This sequence

PL	Priority Group	Related Group
P1	10	A
P2	10	A
P3	0	A
P4	0	A
P5	0	A
P6	10	B
P7	0	B
P8	0	B
P9	0	B

	1	2	3	4	5	6	7	8	9
Sequence 1	P1	P2	P6	P3	P4	P5	P7	P8	P9
Sequence 2	P1	P6	P2	P3	P4	P5	P7	P8	P9
Sequence 3	P6	P1	P2	P3	P4	P5	P7	P8	P9
Sequence 4	P1	P2	P6	P3	P7	P4	P8	P5	P9

Table 3.1: Left: PLs with their associated priority group and group of related PLs. Right: 4 sequences (read from left to right) that are equal, because the order of the PLs within a group of related PLs stays the same. Note that the PLs with a priority of 10 are always planned before those with a priority of 0.

is then modified to create new sequences. The new sequences each need to be evaluated and we keep the sequences with the lowest number of function calls for the next iteration. In the next iteration we do the same modification of sequences for each sequence that we have kept. This process is then repeated d times, with each time modifying k sequences per priority group, and keeping m sequences after evaluating all new sequences. Which means that the algorithm for local search only terminates when a certain number of sequences have been evaluated. Another stopping criterion can be used such as time. This way the local search can be run during the weekends when the system is idle. However, it has the disadvantage that the experiments will become dependent on the system used, but more importantly comparisons between different supply chains become more complex. Some supply chains finish a single run of HP within a minute, whereas others could require more than an hour.

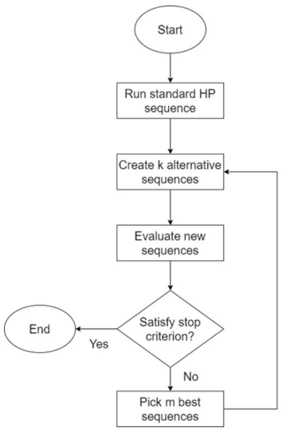


Figure 3.1: Local Search Process

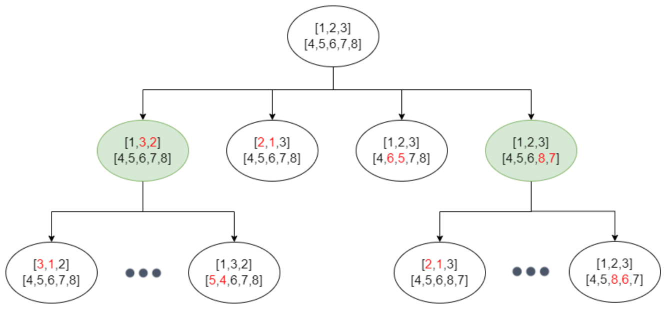


Figure 3.2: Local Search Example

New PL sequences are created by swapping two adjacent PLs in a priority group. The simple idea is to pick a random PL in the sequence and swapping its place with the next related PL within the priority group. However, a single swap will not be sufficient on larger sequences, it will take too many iterations before the first and last PL in a priority group will be swapped. Instead, we allow for multiple swaps at once. This is done by taking a sequence of indices where the swap starts instead of only a single PL. Then by going over the indices we make the swaps between adjacent PLs one at a time, resulting in the new candidate PL sequence, see Figure 3.3. By further keeping track of sequences that have already been evaluated we can avoid running HP when it's not needed. Pseudocode for the algorithm can be found in Algorithm 4.

Applying local search to a test instance shows that it is able to steadily find better sequences, the result of each successive sequence is visualized in Figure 3.4. It is important to note that this approach

Algorithm 3 Local Search

Input: k - the number of new sequences per priority group, m - the number of sequences to keep after each iteration, d - the number of iterations, seq - the first sequence where local search starts

```
1: procedure LocalSearch
2:     $S \leftarrow \{seq\}$
3:     for $i = 0$ to $d$ do
4:         for $s \in S$ do
5:             for $p \in$ Priority Groups do
6:                 for $j = 0$ to $k$ do
7:                     Create new sequence from $s$ and evaluate it in HP
8:         Store $m$ best sequences in $S$
```

Algorithm 4 Swapping adjacent PLs

Input: pg - the priority group where the swap needs to take place, S - the set of sequences already evaluated, q - the number of swaps to make

Output: seq - the new sequence for this priority group

```
1: procedure CreateNewSequence
2:     $n \leftarrow |pg|$
3:     $l \leftarrow [0, ..., n]$
4:     $r \leftarrow 0$
5:     while $r < 5$ and $seq \in S$ do
6:         Shuffle $l$
7:         $seq \leftarrow pg$
8:         for $i = 0$ to $k$ do
9:             $p1 \leftarrow pg[i]$
10:            $p2 \leftarrow$ first PL in pg between $i$ and $n$ that $p1$ is related to
11:            $seq \leftarrow seq$ with $p1$ and $p2$ swapped
12:        $r \leftarrow r + 1$
       return $seq$
```

is prone to getting stuck in a local optimum, especially if the number of swaps done in a single iteration is limited. If a large change in the PL sequence is required to get to a better neighbourhood, then increasing the number of swaps could help out. However, this will depend on the supply chain in question. Creating larger changes in the sequences could also lead to spending more time exploring sequences with worse performance.

3.1.3. Synthetic instances

The previous approaches support the data collection process in obtaining good training samples for the current state of the supply chain, a single instance. However, the supply chain is not a static object, over time changes are to be expected. These changes can range from ones that happen over time, such as the demands for products fluctuating and the degradation of resources, to more instantaneous changes such as the inclusion or removal of items, as well as locations. If these changes were never to happen then simply reproducing the best solution found by the local search process, would solve the problem. This means that we need to make the model robust against the changes that we can expect to happen, only exploring the current instance and using that as training data is not sufficient. We need to collect data from different instances of this supply chain, where we can observe the behaviour of the PL sequences when the supply chain is changed.

While it’s possible to combine the training data obtained from different supply chains to create one large set of training data, the performance of this is unknown. Supply chains from different companies can vary drastically, for example, some might. The heuristics that are beneficial for one company, might not have the same effect for the other. Additionally, even with combining the training data from the companies available, the amount of data might not be sufficient. We will need to prepare a different

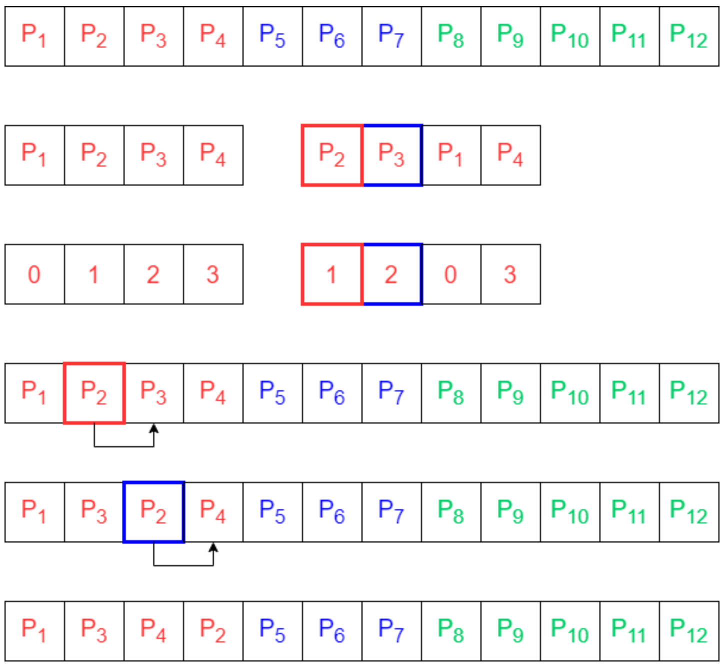

P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12
P1 P2 P3 P4
P2 P3 P1 P4
0 1 2 3
1 2 0 3
P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12
P1 P3 P2 P4 P5 P6 P7 P8 P9 P10 P11 P12
P1 P3 P4 P2 P5 P6 P7 P8 P9 P10 P11 P12

Figure 3.3: Adjacent Swap in the red priority group. A random order for the indices is created and based on that the red adjacent swap on index 1 is performed first, followed by the blue adjacent swap on index 2. If more swaps would have taken place, the next swap would be on index 0.

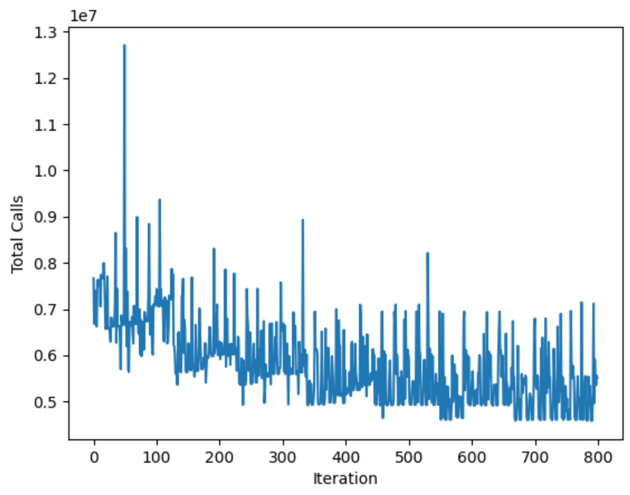


Figure 3.4: Total calls over each successive sequence in the local search process. The instance contains 52 PLs spread over 7 priority groups. The parameters used for the local search process are: $d = 6$, $m = 4$, $k = 2$, and for the adjacent swap: $q = 4$

approach that can provide us with more training data.

The approach in mind is to alter the existing instance to reflect changes that we would like the model to be robust against. The changes chosen for this work are the daily demand of each PL, as well as the production rate of the different resources. Other changes such as varying the spread of the demand and changing the existing supplies for the PLs are not included in this work, but should be considered for future work. The daily demand of each PL is varied by picking a factor in the range $[0.85, 1.15]$ for each day and multiplying it with the current demand. For the production rate of the machines we pick a factor in the range $[0.95, 1.05]$. These changes can cause new PLs to be introduced to the problem, because their demand is now higher than the available inventory. The reverse holds true as well, where previously included PLs do not need to be included in the planning anymore. Furthermore, production that could be solved in one batch, might now require to be split due to a lower production rate, or the other way where planning the production for a PL might become easier with a higher production rate.

Some changes are not considered for this work, such as varying the PLs that need to be planned, essentially changing the structure of the supply chain. This change does not happen often and could be an indicator to train a new model instead. However, some initial experiments were run with a few PLs left out in the training data. These experiments showed a possibility for learning from smaller subsets of PLs. Similar experiments were run with PLs included in the training data, but not in the validation set. Collecting training data from smaller instances could be an approach to further improve the data collection process, by reducing the search space of a single instance and allowing for more instances to be explored.

3.2. Stepwise Expert Model

The first model created in this work is the stepwise expert model, it revolves around the idea of imitation learning, where we attempt to mimic an expert when deciding which PL to plan next. At the start of HP we predict what the best starting PL is out of all available options, we take this PL and solve the planning problem for it. After each PL we create a new observation for the current state, recording features such as:

- The remaining demand of each PL
- The remaining demand in total
- The number Function calls made
- The number of lost sales
- The current inventory of each PL
- The number of PLs left

This information is then used to update all the PLs that haven't been planned yet. Once the state is updated a new prediction is made for the next PL. Using this approach we create a new PL sequence step by step.

3.2.1. Features

In supervised learning the training data will consist of two types of information: the features and the label(s). The goal is to use the features to predict what the label will be. Each sample in the training data will be a single PL.

The training features used in this model can be split into two groups: PL features, and state features. The PL features are the features that encode the current PL and it generally stays consistent at each step. The state features, on the other hand, are those that encode information on the current state and will be altered at each step by the previous PL that was included in the planning. The idea is to find out which PL with its accompanying PL features suit the current state features the best.

Aside from the input features an important aspect of the training data will be the label. The objective is to take the candidate PLs and mark them as either a good or a poor choice when considering them as the next PL to be included in the planning. One approach is to use a binary classification, where a sample is marked as expert if planning this sequence with the current state would be part of a good sequence. There are several options when it comes to choosing which sequences are good:

- Take top k sequences as expert
- Take top l percentage of the sequences as expert
- Combine the previous two

Feature	Explanation
Priority	The priority of this PL
Network Level	Network Level as defined in section 2.3
Rank	The rank of this PL within its priority group, first PL to be planned has a value of 1, incrementing with each successive PL
Priority Group Size	The size of the priority group this sample belongs to
Number of Violations	Total number of PLs that requires planning
Violation Progress	The current number of planned PLs
Days	The number of days in this planning
First Violation Ordinal	The first day with a violation for this PL
Max Violation Ordinal	The (currently known) last day with a violation for this PL
Demand	The demand of this PL in SKU
Total Demand	The sum of demands for all PLs
Fraction of Total Demand Solved	The percentage of total demand solved so far, between 0 and 1.
Starting Inventory	Starting inventory of this PL in SKU
Lost Sales Priority	Amount of sales lost for this priority group in SKU
Lost Sales Total	Amount of sales lost over all priority groups so far

Table 3.2: Features for step-wise expert model

To take the top k sequences you simply need to sort all the sequences found during the local search process. Finding the optimal value for k, however, is a lot trickier. Keeping it low means that the training data will likely contain a higher fraction of sequences that are relatively fast, but the amount of sequences will then be small. Whereas increasing the value for k makes more data available, but with the added consequence that the quality of the data could see a drop. There is another argument for making this value for k larger, which is that it can lead to a more robust model as it has seen more varied sequences.

A similar approach is to label a percentage of the found sequences as expert. This has the benefits that it is easier to balance the two classes of expert and non-expert regardless of the data.

However, both approaches are prone to incorrectly classifying the sequences. In the scenario that the local search process finds a large amount of sequences of a high quality, the k or top l% best sequences might not include all these sequences. Which means that some of them will be labeled as not an expert despite having similar quality to those that have been labeled as experts. Similarly, if the sequences found show no sign of improvement, the labels assigned using these approaches could cause more harm. This can be partially solved by taking all top k sequences, but also those that are really close in performance to the top. Essentially, taking all sequences that have a number of function calls below the function call of the k-th sequences with some margin.

Example:
For example, if the k-th sequence has 1000 function calls and the margin is set to 5%, then we will label all sequences with a number of function calls below 1050 as an expert.

3.2.2. Prediction

A PL sequence is created using a trained model by predicting one step at a time, at each step we take all the remaining PLs in the current priority group and compute the features for them. These are all given as input to the model, while the output will be a label for each PL as well as the probability of it being in that class. Out of all PLs that are labelled as experts we take the one with the highest probability as our next PL. This process is then repeated for all the remaining PLs in this group, the algorithm moves on to the next priority group once the current one is empty.

3.2.3. Discussion

The stepwise expert model was a first iteration at experimenting with using ML to predict a new PL sequence. The results for this approach were not too promising due to several factors, the first being a lack of good features.

In the process of experimenting with this approach it was noted that the difference between a fast PL sequence and a slow one usually comes down to a single or a few PLs taking up a majority of the time. This is because they run into the issue where they are unable to find a successful allocation in a relatively short time, which is required to be able to eliminate large parts of the search space when solving one violation. This is further made worse when these PLs require several materials that are scarce, where an allocation is only shown to be infeasible when trying to resolve the materials required. These scenarios can happen when two PLs are contesting for limited resources, if one of these PLs has very limited possibilities then prioritising that PL could be beneficial as it will be easier for the other PLs to find other resources to use. The current features, however, do not capture these contentions between PLs. Other issues with the features show up in the demand and inventory. These features can be useful to give an idea of how much work needs to be done to create a planning for a certain PL. However, a single SKU for one PL is not equal to a SKU for another, in the case of production this will also depend on how fast the production for this PL can be, so we will need to use that to scale these features. Lastly, a high demand for one product does not necessarily mean that it will be an immediate issue. If one PL only has a high demand in a limited time frame where no other PLs have demands, then this would not cause any issues at all. So we will need to look at how often and how much demand between different PLs overlap with each other.

Another issue lies with the fact that the data does not capture the dependencies between the PLs it has already planned and those that are coming up next. Currently, if a sequence is labelled as a good sequence, all the steps in that sequence are considered as an example of an expert's action, but this also works the other way around. If a sample is marked as bad, all the actions are similarly considered as poor choices. This is, however, not practical. A single PL in the sequence could have caused the sequence to be relatively slow, whereas the majority of the actions taken in that sequence were optimal. While it's easy to find out which PL runs into such issues by simply looking at the time they require, it is not trivial to find the steps that caused the issue. It could be a single step that by itself did not have any issues while running, but only caused it for another PL later down the line, or it could even be some combination of steps that caused the issues. A good sequence must contain only (relatively) good actions, whereas a bad sequence could only contain a single bad action. In this case it is easier to learn from good examples, than to learn from bad examples.

The last issue lies with computation time required for this approach. To create a single sequence we now need to evaluate a lot of steps one at a time, at each step we need to update the features for the remaining PLs and predict the label for these new samples.

3.2.4. Implementation

The last implementation for this model was made in Python and uses a random forest implementation from the library `scikit-learn`[1]. Alternative implementations have been experimented with, using neural networks from the library `AutoKeras`[2].

3.3. Learning to Rank Model

The second approach we try uses learning to rank (LTR). LTR refers to the broad domain of ML models that are used for ranking tasks, the applications of these can be found in Information Retrieval, Natural Language Processing, and Data Mining [20]. The idea behind LTR is often explained in the setting of information retrieval, for example, when searching for websites in a search engine. The query specifies what you are searching for, whereas the documents are the results that match this query. The task for LTR is to sort these documents based on the relevance to the query. A simple example would be, when searching for "chocolate recipes". If the three documents retrieved are:

- "Making chocolate 101"

[1]https://scikit-learn.org/stable/
[2]https://autokeras.com/

- “Charlie and the chocolate factory”
- “Chocolate truffles recipe”

The document for “Charlie and the chocolate factory” will likely have a lower relevance for our query, whereas the other two documents would be more relevant and thus needs to be ranked higher. The models need to learn which document is the most relevant for the query that is provided, for this each document will have their own features that can provide more information for the model. These features in the example of information retrieval could be: the page rank, number of clicks, the number of matching words to the query, etc.

Garret et al. [13] have applied LTR in the field of planning problems, where they frame the problem of learning heuristics for large planning problems as an LTR problem. In their work they consider the STRIPS planning problem that can be solved by a forward state-space greedy heuristic search, where the heuristics are used to approximate how far a state is from the goal. The states with a higher score are then more likely to be expanded first. While the use of heuristics differs from the one used in this thesis, we will similarly apply LTR to train a new heuristic.

3.3.1. Features

In the example of information retrieval the documents with the highest relevance scores are the first to be shown to the user. In our case the PLs with the highest relevance score will be planned first. In both cases the training data should act as a demonstration of the correct order between the different items to be ranked. Which means that we will need to provide good PL sequences as our training data. The actual relevance score for each PL does not matter, as long as the order for the PL sequence will remain when sorted on this relevance in descending order. This means that the first PL to be planned in this sequence needs to have the highest relevance, while the last PL has the lowest relevance score as its label. However, as indicated previously the order between two unrelated PLs does not matter. If we train the model with the false knowledge that this order does matter it could lead to degradation of the performance due to this noise. For this reason, we split each group of related PLs as a separate sequence that is used as training data. So each group of related PLs would have their own ranking and relevance scores.

The trained model will be used to predict the entire PL sequence in one go, unlike the previous model. Which means that many of the previous features are not feasible with this model, as we can only use features that are already available before running HP. So nothing changes to features such as the demand, but features that are related to the progress of HP, such as the amount of lost sales so far, would not be usable with this approach. The features used in this model can be found in Table 3.4. The issues mentioned in subsection 3.2.3 have been taken into account while choosing these features. These changes can be divided into three categories:

- *The demand is now scaled by the production rate of the PL*. For each PL the minimum production rate out of all usable resources is taken as an estimate of the effort required to resolve the demand in the worst case. This is not always an accurate estimate and should be used with caution. A counterexample for the minimum rate is when an old machine is still kept available but is in fact rarely used. This could lead to the PLs having a much lower minimum rate. In the case of a PL that only has transportation this scaling is not done because transportation does not impose a limit on the quantity, only on the number of days required.
- *The spread of the demand over the planning horizon is considered.* The demand of a single PL over the planning horizon can be seen as a n-dimensional vector (see Table 3.3), where n is the number of days in the planning horizon. The sparsity (or lack of it) in this vector can be used to distinguish PLs that could require more effort in planning.
- *The overlap between the demands of PLs is considered*. The overlap for two PLs can be computed by taking the daily demand of a PL and turning it into a vector, see Table 3.3 for an example of such demands. The dot product between two of such demand vectors will be an indicator of

how much overlap there is between two PLs. This can be computed for each pair of related PLs and the sum of these dot products will be used as a metric of how much overlap one PL has with the other PLs that need to be planned. For example, for p_1 the dot products with p_2 and p_3 are 18 and 42 respectively, the sum is then 60. The same computations can be done for p_2 and p_3 to get 20 and 44 as the sum of dot products. A higher value means that this PL can cause more issues for other PLs, whereas a value of 0 would imply that there is no overlap between the demands. In this case p_3 has the highest sum, due to the high demand it has and the overlap that exists for these high demands. This will more likely cause issues during planning as the resources will run out of capacity earlier. On the other hand, p_2 has a lower sum than p_1 despite having more demand. This is because the demand of p_2 has less overlap with the other PLs. This simple feature currently only uses the day of the demand. While in practice a demand on day t means that any day from 0 to $t-1$ could be used to resolve this demand, with days closer to t having a higher likelihood. So one way to improve this feature is by using a sliding window for computing the overlap between two PLs, instead of only a dot product. For example, if using a sliding window of 2 days, then the demand of day t is its own demand plus the demand of $t-1$ and $t+1$ if they are within the planning horizon.

- *The contention for resources between PLs is considered*. This feature that is related to how often a PL is competing for resources, as well as what fraction of the desired capacity it would take. For the count we take the number of days where the PL has demand while at least one other PL that shares the same resources does as well. For each day we can then compute what fraction of the total demand belongs to this PL. For example, using the demands in Table 3.3 we can compute both features for p_2. The count of tight days would be 4, despite there being demand for p_2 on day 4, it is not included in the count because no other related PL has demand on that day. The sum of the fractions would then be roughly 2.06 $(0.6 + 0.5 + 0.71 + 0.25)$. For p_1 and p_3 it would be roughly 1.52 and 2.42 respectively.

Time		0	1	2	3	4	5	6
Demand	p_1	2	0	2	2	0	2	2
	p_2	3	1	0	5	3	1	0
	p_3	0	1	10	0	0	1	10

Table 3.3: Example of demands for three PLs with a planning horizon of 7 days.

3.3.2. Prediction

Similar to the training data we will split the PLs by their group of related PLs. Each group is given as input to the training data separately, this results in several disjoint sequences which need to be merged back together. The order in which this happens does not matter, we do however adhere to the priority groups when merging the sequences, despite it not having an impact on the results. This is to stick to the rule imposed that PLs of different priorities should not be swapped. While the model does not guarantee this, it is possible that the last PL of a higher priority and the first PL of a lower priority get relevance scores that are close to one another, such that the PL with a higher priority is ranked lower. This needs to be checked when creating the final PL sequence, which can be done by separating the PLs on their priority first, before sorting the PLs in a priority group on the predicted relevance.

3.3.3. Implementation

Initial experiments for the LTR model used an implementation for LTR called LambdaMART [5], the implementation was in Python using the code found in an online repository[3]. Whereas the final LTR model uses the algorithm called `FastTreeRanking` from the library `ML.NET`[4], a library made by Microsoft for C#. `FastTreeRanking` is similarly based on Multiple Additive Regression Trees (MART).

[3]https://github.com/lezzago/LambdaMart
[4]https://dotnet.microsoft.com/en-us/apps/machinelearning-ai/ml-dotnet

Feature	Explanation
Priority	The priority of this PL
Network Level	Network Level as defined in section 2.3
Directly Related PLs	The number of PLs that are directly related to this PL, that is those that share resources or materials
Resources	The number of resources that this PL has access to
Related Resources	The number of PLs that (in)directly share resources with this PL
Directly Related Resources	The number of PLs that directly share resources with this PL
Minimum Production Rate	If the PL has production: The minimum production rate of this PL, considering all resources + BoM combinations , this is essentially a worst case scenario Else: Always 0
Starting Inventory	Starting inventory of this PL in SKU
Demand	The total demand of this PL in SKU
Scaled Demand	If the PL has production: The total demand of this PL is scaled by dividing by the minimum rate Else: Same as demand
Relative Demand	The PL's own scaled demand divided by the total scaled demand of the entire group of related PLs
Average Demand	Average of the demands of this PL in the entire planning horizon
Standard Deviation Demand	Standard deviation of the demands of this PL in the entire planning horizon
Coefficient of Variation Demand	Coefficient of variation of the demands of this PL in the entire planning horizon, average demand divided by standard deviation demand
SumDots	Sum of dot products between the demands of this PL with its related PLs
Count of Tight Days	The number of days where it has demand that overlaps with at least one related PL that shares a resource
Share of Tight Days	The sum of the fractions of demand that it has with respect to the other PLs that it competes against

Table 3.4: Features for LTR model

MARTs use multiple regression trees and train in a sequential order. At each step we take the result from the previous trees and we fit a new function on our data, such that the sum of the previous result and the result from the new function minimizes some loss [11]. Essentially, each additional tree tries to improve the previous result further.

This library has a severe limitation for the LTR model as the labels needs to be an integer from 0 to 4. This means that we will not be able to cleanly separate all related PLs when we have more than 5 PLs, we will only be able to separate them into 5 different groups, where the model will not be able to distinguish the order between PLs within one such group.

3.4. Pairwise Model

The idea behind the pairwise model is to take a step back from learning an entire sequence immediately, instead we consider only pairs of PLs. For each pair of PLs p_1 and p_2 we create two samples as training data. One sample will be created for the option of planning p_1 before p_2 and one sample will be created for the reverse, planning p_2 before p_1. We can create a binary classification problem with this data, the samples with the correct order will get a label of 1, whereas the other will get the label 0.

The idea for this approach comes from the fact that the current heuristic, where the products are first ordered based on the priority group and then ordered based on the network level, can be modelled using a simple decision tree. This decision tree only uses two features to classify each sample: priority group and network level. A comparison is made between each PL based on these two features resulting in a PL sequence. In Figure 3.5, such a decision tree is visualized with some test data.

The previous decision tree can be expanded with these features, as well as more layers with the idea that the current heuristic used in HP is still in there as a fallback if the other features end up not contributing much. In Figure 3.7, the decision tree is expanded with an additional feature. In the first decision tree we can see that the priority group creates clean splits in the data, resulting in leaf nodes with a single class. This is because it is a hard constraint on the sequence and any sequence in the training data satisfies this constraint. The network level, however, can provide some split in the data. When the feature is less than or equal to -0.5 according to the splitting rule in the node, 76% of the samples have a label of 1. This rule can be interpreted as if a PL p_1 has a higher network level than a PL p_2 then p_1 should be planned before p_2, which is exactly what the current heuristic uses. However, in the other cases, where the network level of the two PLs are equal or the network levels of the two are swapped, the distinction between the classes is less clear. In Figure 3.7 an additional level is added with which it can be seen that in most leaf nodes the split in the classes is quite clear, except for one leaf, where it is roughly 50% split for both classes. This is the case when the network level of two PLs are exactly the same. Additional information will be required to create a better split in these situations, an example of such can be found in Figure 3.7. An additional feature is added in this decision tree, that uses the number of directly related PLs. The idea is to use these additional features to further improve the accuracy of a simple model.

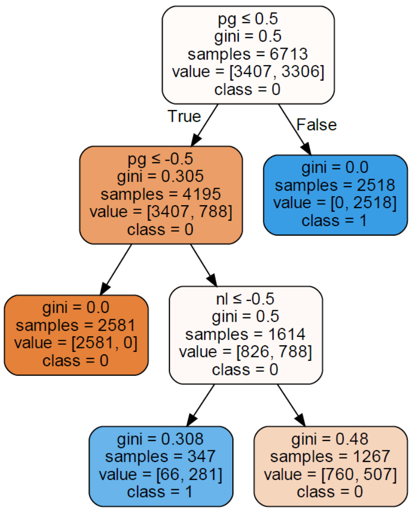


Figure 3.5: Decision Tree with priority and network level as features, imitating the current heuristic

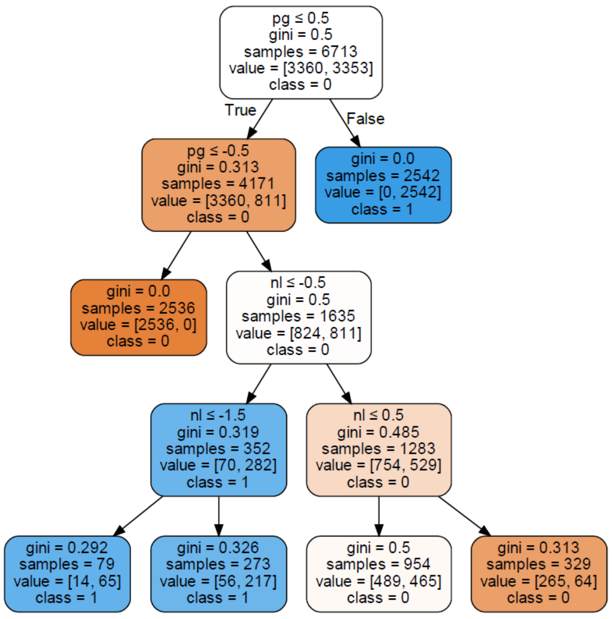


Figure 3.6: An additional level added to the decision tree

3.4.1. Features

The pairwise model uses similar features to the learning to rank model. The differences between the two lies in the fact that pairwise combines the values of both PLs for its own features. For example, instead of only the demand of one PL, we take the difference of demand between the two PLs. A different option is to simply create binary features that indicate whether the demand of the first PL is higher than the second one. However, this would remove some information that the model could use to learn. The difference between the two values could provide useful information, especially in cases where the difference is so little that it is insignificant.

3.4.2. Prediction

Similar to the previous model of learning to rank, only related PLs are considered. In this case that means we only take pairs of related PLs as the samples that need to be predicted.

The trained model predicts whether the two PLs in a sample are in the correct order or not, this means that if the model outputs a 1 for a sample that puts p_1 in front of p_2, then the resulting PL

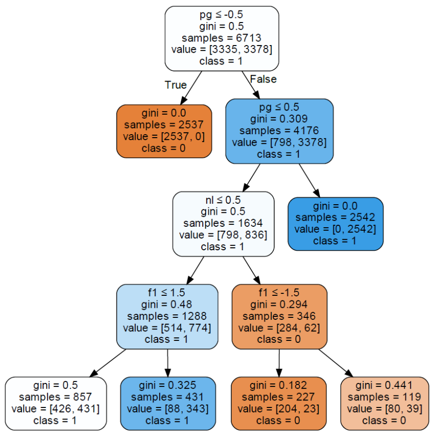


Figure 3.7: Decision Tree with an additional feature using the number of directly related PLs

sequence should have p_1 planned before p_2. One way to get a PL sequence out of these predictions is by summing up how often a PL is predicted to be planned ahead of another PL. For example, when looking at a p_1, we have all the samples that include p_1 as the first PL in the partial order, we count how many of these samples result in a label of 1. This is done for every single PL and then we sort these PLs in descending order.

This simple idea can be improved by using the confidence score of a prediction. A score closer to 1 means that the model is more confident of the label being 1, and a score closer to 0 means it is confident of the label being 0. We can sum up these confidence scores instead of simply the number of predicted labels being 1. A PL that has high scores in its predictions is then more likely to be planned ahead of one that has an equal amount of predicted labels being 1, but with lower confidence scores. This reduces the influence from predictions that hover around 0.5, where the model is uncertain of which class it belongs to. This can be further altered such that the predictions of which the model is uncertain is not taken into account at all. Only the scores above a certain threshold are summed up, those below it are ignored.

3.4.3. Implementation

The first iteration of this model used a simple decision tree in Python from the library `scikit-learn`. The final pairwise model uses the same library as the LTR model and the algorithm has a similar name, `FastTreeBinary`. It uses the same approach with MARTs, but now used for binary classification rather than ranking.

4 Experiments

In this section we will evaluate the performance of the two heuristics obtained from using the ML approaches. First we will take a look at the performance of the current heuristic and compare it to the alternative sequences. Then we will create an experimental setup for evaluating the ML approaches, which we will use on several datasets. The first dataset will be a synthetic dataset used by *Outperform*, the other datasets will be real supply chains provided by *Outperform*. The real supply chains will be referred to as “Supply chain A”, “Supply chain B”, etc.

4.1. Datasets

4.1.1. Synthetic Supply Chain

This is a supply chain that is created by `Outperform` to showcase their products. The supply chain is relatively small with only three locations and a limited amount of products per location. The experiments were run with 52 PLs that need to be ordered for the sequence, these are split into 7 priority groups, the distribution can be found in Table 4.1. The execution time of HP for this input is roughly 50 seconds.

Priority Group	Number of PLs	Groups of Related PLs
7	10	1
6	7	3
4	10	3
3	10	4
2	8	4
1	5	3
0	2	1

Table 4.1: Properties of the synthetic supply chain

In this setting the three locations are largely disjoint, but within one location most PLs are related to each other. The dataset has been further modified to decrease the available resources, such that there is less capacity for production and PLs run into more unsolvable demand. This change is done because the order for the PL sequence does not matter if there is more capacity than is needed.

4.1.2. Supply Chain A

This is a real supply chain with a much larger size than the synthetic one, but the execution time of HP is much faster at about 10 seconds per run. It has 410 PLs that need to be planned with HP, not including any materials that these PLs can depend on. These 410 PLs are separated into 15 locations and 4 priority groups, the distributions for the priority groups can be found in Table 4.2. This supply chain is a more complex one compared to the synthetic supply chain. It has a larger mix between production and transportation, due to several locations acting as warehouses or stores.

Priority Group	Number of PLs	Groups of Related PLs
3	1	1
2	109	10
1	13	3
0	287	23

Table 4.2: Properties of supply chain A

4.1.3. Supply Chain B

Supply chain B comes from the same customer as supply chain A, but with a year between the two and some changes in the supply chain. This dataset now contains 444 PLs that need to be planned. They are separated into 18 locations and 4 priority groups. The new distributions for the priority groups can be found in Table 4.3. The changes in this supply chain are quite varied, from the inclusion/removal of PLs to changes in the resources. With these changes the time required to run HP measured on my machine changed from roughly 10 seconds to about 14 minutes.

Priority Group	Number of PLs	Groups of Related PLs
3	1	1
2	109	8
1	15	4
0	319	21

Table 4.3: Properties of supply chain B

4.1.4. Supply Chain C

Supply chain C is another large supply chain, but one where the time required to run HP is similar to supply chain B. It requires roughly 10 minutes on the same machine to finish the planning. The planning includes 346 PLs, separated into 4 locations and 10 priority groups. However, out of the 10 priority groups only 2 are mainly used, as can be seen in Table 4.4.

Priority Group	Number of PLs	Groups of Related PLs
9	28	3
8	1	1
7	1	1
6	1	1
5	1	1
4	1	1
3	1	1
2	1	1
1	2	1
0	309	46

Table 4.4: Properties of supply chain C

4.1.5. Supply Chain D

The last supply chain is a bit smaller than the previous ones with 253 PLs, for which HP requires about 1-2 minutes to solve. This supply chain has a clear structure for which PLs to prioritize, as it has 225 priority groups. Essentially, each PL has its own priority, which means that the number of possible sequences is heavily limited in this supply chain. This supply chain only has a single location and almost all PLs are related to each other, similar to the synthetic supply chain.

4.2. Baseline Performance

Before we evaluate the performance of the ML models, we need to look into the performance of the current heuristic for the different datasets. Ideally, we would be able to compare the performance of the PL sequence obtained from the current heuristic with an optimal sequence. However, doing this is not feasible due to the size of the search space, so instead we compare the current heuristics with the best obtained sequences from local search. For each instance a score is computed for the possible improvement by taking the number of function calls for the current sequence and divide it by the minimum number of function calls of any sequence for this instance. Resulting in a value of 1 if there is no improvement possible, and a score above 1 where the higher the value the more improvement there has been found. The results can be found in Figure 4.1.

4.2.1. Synthetic Supply Chain

The local search is run with the following parameters: $d = 6$, $m = 4$, $k = 2$. For this supply chain 80 instances are evaluated with the local search. For this supply chain we can see that there are multiple instances with a lot of room for improvement.

4.2.2. Supply Chain A

The local search is run with the following parameters: $d = 6$, $m = 4$, $k = 2$. For this supply chain 80 instances are evaluated with the local search. For this supply chain not a lot of improvement has been found from running local search with the chosen parameters.

4.2.3. Supply Chain B

The local search is run with the following parameters: $d = 3$, $m = 2$, $k = 1$. However, only 50 instances were collected. We can see that this supply chain now has more room for improvement compared to the supply chain A. Especially, the one instance with a score above 1.6 can be promising.

4.2.4. Supply Chain C

The local search is run with the following parameters: $d = 6$, $m = 2$, $k = 1$, the change in values compared to the other supply chains is to reduce the amount of time required for exploring a single instance. Furthermore, only 13 instances were collected for this. The exploration of a single instance took between 6 to 8 hours. These changes, however, mean that the exploration of each instance is less thorough and the estimation for improvement is less accurate. From the obtained information we can see that the situation for this supply chain is similar to that of supply chain A, where there isn't a lot of improvement found.

4.2.5. Supply Chain D

The local search is run with the following parameters: $d = 10$, $m = 2$, $k = 1$. The depth of the local search is increased for this supply chain due to a limitation found in the implementation of the local search. Only a single priority group is considered while swapping adjacent PLs for a new sequence. This means that the parameter of the depth for the search d will need to scale with the number of priority groups in the current supply chain. The increase of the parameter d is to mitigate some of the issues introduced by this, but it should be resolved properly by adjusting the local search approach to allow for changes in multiple priority groups in one go.

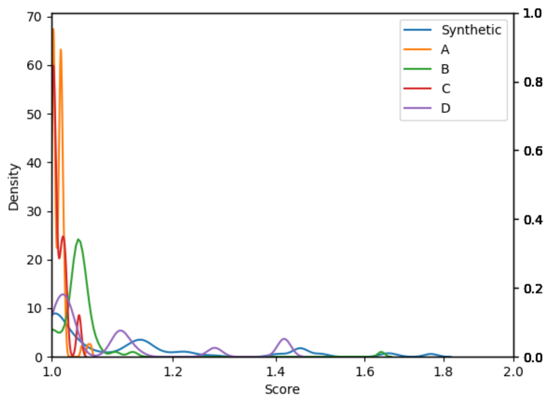


Figure 4.1: Possible improvements for the different supply chains

4.3. Evaluation

The experiments will be run locally, with most of the experiments being run on the same machine. Some smaller experiments such as hyper parameter tuning have been run on a different machine. All final experiments involve running HP, so these processes are integrated in the code of `Outperform` with the usage of C# for the machine learning models.

4.3.1. Performance

To evaluate the performance of a model we can compare the function calls required for the PL sequence obtained from the model's output with the current heuristic. A comparison is made between the two and a score can be computed as follows:

$$S_i = \frac{\#Function\_calls\_HP}{\#Function\_calls\_ML} \tag{4.1}$$

A score of 1 means that there is no improvement from the ML model, a score higher than 1 signifies improvement, whereas a score below 1 means that the ML model results in worse performance for this instance.

This score is then computed for all synthetic instances in the validation set, resulting in a distribution that gives an idea of what to expect from the ML model with regards to the performance on this supply chain.

Additionally, a check is done for each instance that the ML model does not result in worse quality for the resulting planning. The scores for these metrics is computed in the same way as in Equation 4.1.

4.3.2. Hyperparameter Tuning

Other than collecting the right data, ML also benefits from training the right model that fits the data and the problem. For both the models the underlying algorithm used is the `FastTree` from the `ML.NET` library. It has four parameters that you can freely set when training the model:

The number of leaves is the maximum number of leaves each tree is allowed to have, increasing this will allow more complex trees that are able to split the samples better. However, making this too high will increase the cost of training the model, as well as possibly leading to overfitting on the training data. The default value for this is 20.

The number of trees is the maximum number of regression trees to use, increasing this will allow the model to fine-tune its decisions better. This is because each additional tree is boosted by its predecessors, this allows the new tree to make small adjustments to its predecessors' prediction to further optimize the model. The default for this is 100.

The learning rate is used to change how much each successive regression tree affects the prediction. With MARTs it is often the case that the first few trees will be the most dominant in the prediction, with each additional tree having less control [32]. The default value for this is 0.2.

The minimum number of examples per leaf can be used to avoid overfitting when the number of leaves per tree is too high. The default value for this is 10.

In this work we will focus the hyperparameter tuning on the number of leaves and the number of trees, this is because changing the other two hyperparameters did not result in significant changes to the prediction of the trained models. The default values will be used for those two. This also reduces the time required for finding and training the final model used for evaluation.

A grid search will be used to find the optimal parameters for the number of leaves and trees. For the number of leaves the values used are 10, 20, 30, 50, 100, and 200. For the number of trees, the values are 50, 100, 150, 200, and 300. Each combination of these values is then evaluated on a validation dataset consisting of 20 synthetic instances. An example of the results from such a grid search can be found in Figure 4.2. For the pairwise model it is evident that a higher number of leaves leads to a higher score when evaluating it on HP, the numer of trees does not result in much improvement after some point. From this we can, for example, choose the configuration with at most 200 leaves and 150 trees. On the other hand, we can see that with the LTR model it is not advisable to continuously increase the hyperparameters. The optimal model was found around 10 leaves per tree and 200 or 300 trees in total.

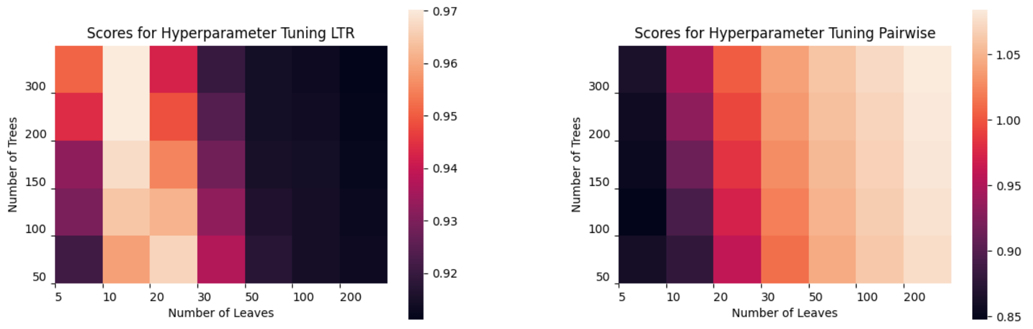


Figure 4.2: Heatmap showing the average scores for 20 evaluated instances per configuration. On the left we have the LTR model and on the right we have the pairwise model.

4.3.3. Experiment Design

We have a total of 5 supply chains with varying properties. For each supply chain we will conduct an experiment to test how well the designed models work as a new heuristic for the PL sequence. Due to the different sizes of the supply chains the amount of training data that each experiment uses will vary, as well as the number of instances used in the validation of the model, see Table 4.5.

The amount of training data for these instances were chosen to be on the larger side and often required multiple days to collect. One experiment that will be conducted is with regards to the amount of training data used. For this we will explore the options of only using a subset of the 80 collected instances for the synthetic supply chain. The sizes for the subsets are: 5, 10, 20, 40, 60, 80.

A different experiment is related to the robustness of the models to changes in the supply chain. This is partially covered by the use of the synthetic instances, however, they only cover a small portion of the possible changes. Due to the relation between supply chain A and B we can create a model trained on the data of the older supply chain A and evaluate it on the newer supply chain B. This gives an idea of whether the model stays robust despite some larger changes in the supply chain.

Dataset	Training Instances	Test Instances
Synthetic	80	500
A	80	500
B	50	25
C	10	26
D	30	200

Table 4.5: Setup for the performance evaluations

A similar experiment is done by adjusting the range for the factors used in the creation of synthetic instances. This is done by training a model using data gathered from instances with the current factors, but we evaluate the model on instances created with a smaller or larger range for these factors. The current range for the daily demand is [0.85, 1.15] and the range for the production rate is [0.95, 1.05]. For the experiment with smaller changes we will use the ranges [0.925, 1.075] and [0.975, 1.025] respectively. For the experiment with larger changes we will use the ranges [0.7, 1.3] and [0.9, 1.1] respectively.

The last experiment is on using the models trained on one supply chain to predict the PL sequences for an entirely different supply chain, not like the experiment with supply chain B where the trained model is still on a previous iteration of the supply chain. We will train a model on the synthetic supply chain and evaluate it on supply chain A, and vice versa.

4.4. Discussion

In this section we will discuss the results from each experiment, the results are visualized in Appendix A and a summary of the results can be found in Table 4.6.

Model	Dataset	Score - Function Calls	Score - Lost Sales
LTR	Synthetic	0.9663	0.9950
	A	0.7649	0.8802
	B	81.5193	0.8292
	C	-	-
	D	1.0000	1.0000
PW	Synthetic	1.0876	0.9986
	A	1.0010	1.0578
	B	0.2286	0.8716
	C	0.9167	0.9189
	D	1.0000	1.0000

Table 4.6: Results for each model and supply chain. The scores are the median in terms of function calls and lost sales.

4.4.1. Performance

Synthetic Supply Chain, see subsection A.1.1
The synthetic instance is one with reasonable room for improvement, the sequences collected in the training data managed to capture this and the trained model should thus be able to reproduce that behavior. From the results we can see that the LTR model is worse in performance than the baseline heuristic, whereas the pairwise model shows a slight improvement on average.

The decrease of performance of the LTR model can likely be attributed to the limitation mentioned in subsection 3.3.3. Only using 5 values for the relevance score is a severe limitation for the model. The idea behind a different relevance score for each PL is to properly distinguish the order between each PL in the sequence used as training data. The current implementation only allows for a less fine-grained order to be used, while it has been mentioned that a single swap between two successive PLs could make or break the entire PL sequence for HP.

The performance of the pairwise model on the other hand shows some promise for the approach. While a large part of the instances hover around a score of 1, it has a second peak around 1.4.

Supply Chain A, see subsection A.1.2
Supply chain A is one where the execution time of HP was relatively fast. However, this was very little room for improvement found using the local search approach. The results from the LTR model show two large spikes, in some instances the results are similar to the current heuristic, while in other cases the results are much worse. This model in its current state was unable to learn a better heuristic.

The pairwise model on the other hand shows that for some instances it was able to find a much better PL sequence. When zoomed in on the instances where the score was below 1.2, we can see that for the majority of the instances there wasn't much improvement found.

It is important to note, however, that the quality of the results has been affected as well. In terms of lost sales we can see that there is an improvement in some of the instances. In terms of below target inventories and changeover we can see that there is a slight decrease in quality. In both cases the decrease isn't too much and can likely be explained. A better PL sequence often results in more opportunities for production/transportation being found, this is because not finding an opportunity and having to do backtracking in the branch and bound part of the algorithm is what takes the most time in HP. With more supply being planned we also have to use more materials, which can then go below target. HP by default is configured to optimize for lost sales, which means that it does not care that inventories go below target and will not try to fix that issue. This can explain the small decrease in quality of the resulting planning from the perspective of below target inventories. Changeover follows a simpler reasoning, where more production opportunities can lead to more time being required to change from producing one product to another.

Supply Chain B, see subsection A.1.3
Supply chain B shares the same source as supply chain A, the two supply chains share a majority of their PLs. This supply chain showed slightly more room for improvement compared to supply chain A. However, this is not reflected in the results of this experiment. The LTR model is able to find some instances where its new PL sequence contributes to a significant improvement, while the majority of the instances are slightly worse than the current heuristic. However, the quality of the plannings are much worse, especially the amount of lost sales.

The pairwise model has a significantly worse performance in this experiment, where not even a single instance had a better PL sequence than the current heuristic. This behaviour was not expected from the pairwise model as this model had a decent performance on supply chain A.

The most likely explanation for the poor performance of the models on this supply chain is related to the training data collected. The range of the local search process was reduced to deal with the increased execution time of HP for this supply chain. This leads to a lower quality for the training data. More experiments need to be run with this supply chain to identify the issue. A simple start would be to run the local search process for a longer duration for the same experiment, despite the increase in the time required. A similar experiment can be run with supply chain A, but decreasing the range of the local search there. This experiment can be used to get an idea whether the issue stems from the quality of the training data or not.

Supply Chain C, see subsection A.1.4
For this experiment only the pairwise model was evaluated as the performance of the LTR model was too low. Evaluating the LTR model on a single instance could require several hours. The performance of the pairwise model is similar to the performance of the LTR model for supply chain B. The new PL sequences have a better performance but are not ideal to the decrease in quality of the resulting planning.

Supply Chain D, see subsection A.1.5
Both the LTR model and the pairwise model had similar results for this supply chain. The results are similar to what was found when initially investigating the room for improvement for this supply chain. The reason for this is likely due to the small size of each group of related PLs, which allows the LTR model to avoid its limitation of only separating the PLs into 5 classes.

In some instances there is a significant decrease in the quality in terms of lost sales. Change over is not included in the evaluation as this supply chain does not use this metric.

4.4.2. Robustness

Later Supply Chain, see subsection A.2.2
The experiment with a model trained on supply chain A evaluated on supply chain B shows promising results for the pairwise model. It shows a similar result as that of supply chain A, as well as the results found when investigating the room for improvement of supply chain B. Furthermore, all metrics for the quality show little to no significant decrease. The metric for lost sales shows an increase in quality on average, but it does have a tail where instances show a decline.

This evaluation outperformed the evaluation of the model trained with data from supply chain B, as collecting data there was much more difficult. This leads to the idea for the future of investigating the use of simpler versions of a supply chain to collect training data that can be used on the more complex supply chains.

Different Sizes for Training Data, see subsection A.2.1
For this experiment we have evaluated both the synthetic supply chain and supply chain A with varying sizes for the training data used. The resulting models were evaluated on 100 instances.

For the synthetic supply chain we can see that the LTR model hits its optimal results around 40 instances used for training, after which it starts to show a decline at the end. A similar behaviour can be found for the pairwise model albeit to a lesser extent. For the pairwise model the optimal amount is around 20, after which there is a small decrease in the score. This leads to the conclusion that the results from these models might not suffer from insufficient data and that the quality.

For supply chain A, the LTR model shows a roughly stable score for the different training sizes chosen. The pairwise model, on the other hand has its optimal score when only 5 instances are used, after which there is a steep decline at 10 instances and a recovery at 20 instances. This might be an indication that the quality of the instances used for the experiment with 10 instances had some issues, which was recovered by including another 10 instances of higher quality.

In conclusion, the quality of the training data might be more important than the amount of training data. This conclusion is in line with the results found in subsection 4.4.1, where the quality of the training data was sacrificed for quantity.

Different Supply Chain, see subsection A.2.4
This experiment is split into two parts, one where a model trained on data from the synthetic supply chain is used to evaluate new instances from supply chain A (subsection A.2.4), and the reverse of this (Figure A.2.4).

For the experiment of applying the model trained with data from supply chain A, the results are similar to that of the experiment using data from the synthetic supply chain but a little bit lower.

For the reverse of applying the model trained with data from the synthetic supply chain, the score with regards to the performance of the PL sequence has increased compared to the experiment where training data from supply chain A was used. However, the amount of lost sales has increased as well. This decrease in quality means that the model would not be usable. This is likely due to the model learning some patterns with which it can reduce the execution time, while keeping the lost sales stable for the synthetic supply chain. However, the stability might be because the synthetic supply chain is made such that there is already a large amount of lost sales, due to the tight capacity of the resources.

On the other hand, supply chain A has a more realistic situation where the amount of lost sales is largely limited. This means that in the data from the synthetic supply chain the changes were unlikely to make it much worse than it already was.

This shows that it might be possible to transfer training data from one supply chain to another or even combine training data. There might be general patterns that are shared along supply chains that the models can learn. However, as was seen the metrics for the quality in the current state are not guaranteed and more work should put into this aspect.

Synthetic Instances, see subsection A.2.3
The experiments with the changes in the synthetic instances are split into two, one where the changes are made larger than in the trained model and the other where the changes are smaller. The results of the experiment with the larger changes can be found in subsection A.2.3 and the experiment with the smaller changes can be found in Figure A.2.3. The results of these experiments can be compared to that of the synthetic supply chain in subsection A.1.1.

We can see that the distribution for the scores stays relatively equal in both cases, especially for the LTR model. The main difference lies in a shift in the score, where the experiment with the larger changes has an obvious lower score than the one with smaller changes. This is in line with the expectations, as the experiment with smaller changes is more in line with the data that is collected. Furthermore, if the changes are sufficiently small, the resulting planning problem will not change a lot. For example, if previously some missing demand would need to be resolved by 2 batches of production, a small change in the demand can still be resolved with these 2 batches.

It is advised to pick the range for the changes in the synthetic instances to be larger than practically usable in the supply chain at hand. This increases the robustness of the trained model by decreasing the chance of overfitting on the current instance. This can be seen as an additional hyperparameter that needs to be tuned for these models.

4.4.3. Feature Importance

The ML models used require not only good sequences to learn from, but also properties with which the models can distinguish the different PLs in this sequence. This information needs to be used to learn which properties are relevant when trying to reproduce the PL sequences. These features are collected based on the assumptions that they contain patterns for PLs that are difficult to plan. However, the assumptions might not be correct and the feature might have no (significant) contribution to the prediction of a new PL sequence. Furthermore, for the pairwise model the features are an aggregation of the values for two different PLs. The usefulness of this aggregated feature might differ from the initial feature in the LTR model. By looking at the importance feature we can investigate which features had the most impact on the results of the models.

An approach that can be used for computing the feature importance is done by using Permutation Feature Importance (PFI). This approach permutes the values for a feature before training a new model, which is then used to compute a score for a chosen metric. The difference in the scores between the initial model and the model with the permuted data can then be used to give an approximation of how important each feature is. This process is implemented in the library `ML.NET`, the resulting scores for each feature for the LTR and pairwise model can be found in Table 4.7.

Both models contain features that when permuted did not change the score of the models, this implies that these features were simply not used in the model at all. One possible explanation for this behaviour is when a feature is not useful, essentially the feature has no predictive power at all not even when combined with other features. Another explanation for this could be that the information in this feature is already stored in another feature. This is the likely explanation for a feature such as the average demand. The features for the average demand, standard deviation of demand, and coefficient of variation of demand are all related to each other. With two out of three values you can compute the remaining one.

This approach, however, only gives an idea of what the model has used during learning. It is difficult to interpret the change in the NDCG and F1-Score in terms of change to the results on the validation in HP. A different approach for looking at the feature importance is by simply leaving a feature out and training a model with it, this model can then be used in HP to compute a score for each synthetic instance. The importance of a feature can then be estimated as the decrease in score of the model without this feature compared to the baseline model with all features included. The results can be found in the last two columns of Table 4.7.

The changes to the score with the pairwise model follows the same trends as the change in the F1-score. Some features are not used at all and can thus be safely removed from the model. For the other features we can get a rough indication of which features are the most important by sorting them based on the change to the scores. The results from the LTR model show a picture where most features are actually not used or in fact make the results worse. This is likely because the model makes fewer adjustments when some features are not used. This in turn leads to the results of the model staying closer to the baseline of the current PL sequence used in HP.

Feature	**Mean Change NDCG**	**Mean Change F1-Score**	**Mean Change LTR**	**Mean Change PW**
Network Level	-0,0019 ±1.83E-04	-0.0044 ±1.65E-04	0 ±0	-0.0403 ±0.1678
Directly Related PLs	-0.0749 ±1.35E-04	-0.0159 ±2.48E-04	0.0001 ±0.0029	-0.0317 ±0.1704
Relative Demand	-0.0192 ±1.12E-04	-0.0064 ±2.46E-04	0.0963 ±0.1453	-0.0383 ±0.1533
Starting Inventory	2.47E-05 ±8.21E-0.5	-0.0145 ±1.29E-04	0.1434 ±0.2425	-0.0357 ±0.1832
Demand	0 ±0	-0.0075 ±2.49E-04	0 ±0	0 ±0
Resources	0 ±0	-0.0001 ±1.27E-04	0 ±0	-0.0175 ±0.1372
Minimum Production Rate	0 ±0	-0.0171 ±5.90E-04	0.1025 ±0.1373	-0.0461 ±0.1581
Average Demand	0 ±0	0 ±0	0 ±0	0 ±0
Standard Deviation Demand	0 ±0	-0.0202 ±8.99E-05	0.0004 ±0.0021	-0.0397 ±0.1099
Coefficient of Variation Demand	-0.0549 ±0.0012	-0.0072 ±1.43E-04	0.1897 ±0.2686	-0.0128 ±0.0807
Scaled Demand	-0.0064 ±1.80E-04	-0.0413 ±3.45E-04	0.0684 ±0.1365	0.0093 ±0.1595
SumDots	-4.59E-0.6 ±2.63E-0.6	-0.0696 ±7.78E-04	-0.0080 ±0.0247	-0.0270 ±0.1235
Count of Tight Days	-0.1138 ±3.68E-04	-0.0188 ±7.73E-04	0 ±0	-0.0307 ±0.1621
Share of Tight Days	-0.0054 ±4.15E-04	-0.0052 ±1.77E-04	-0.0080 ±0.0677	-0.0308 ±0.1557
Related Resources	3.81E-05 ±2.79E-05	0 ±0	0 ±0	0 ±0
Directly Related Resources	-3.40E-04 ±7.09E-05	0 ±0	0 ±0	0 ±0

Table 4.7: Feature importance for the LTR model and the pairwise model. NDCG is the metric used while training the LTR model and F1-Score is used for the pairwise model. The last two columns contain the change to the actual score when evaluated in HP on 100 instances.

5 Conclusion

In this thesis we have looked into learning a new heuristic for the supply planning algorithm `Heuristic Planner`. The learned heuristic should order the PLs that need to be planned such that the execution time of HP is decreased, while keeping the quality of the planning stable. To do this we have designed two ML models that use previous runs of the algorithm to learn which PL sequence results in a faster execution time: the LTR model that learns how to rank the PL sequences in its entirety and the pairwise model that learns the same but taking smaller steps by only learning the relations between pairs of PLs.

To train these models a process for collecting data from previous runs is created, as well as an initial setup for creating different instances for a supply chain. The trained models are evaluated using several experiments which shows that the LTR model in its current state is not useful but shows promise if its limitation is fixed, while the pairwise can be a feasible approach to learning a new heuristic for HP. The pairwise model is able to transfer the improvement found in the training data to improvement in new instances of the same supply chain. All while keeping the quality of the resulting mostly stable.

However, some issues were found when the quality of the training data is less ideal. In which case both the performance of the algorithm as well as the quality of the resulting planning suffers significantly. In the literature the works are often evaluated on well-studied problems that have existing datasets to work with, or some oracle is included that has access to the optimal solution for the given input. Evaluations of the quality of the training data is often not included, while this is an important factor for the performance of the resulting model or algorithm.

This work uses imitation learning, which in hindsight might not have been the most suitable choice. The performance of imitation learning is limited by the performance of the expert. Similar to the work of Namaki et al [23], we create an oracle to fill in the missing expert for our training data. This means that we are limited to the best solutions found in the local search process. Compared to the work of Namaki et al., our oracle has a lower performance in terms of improvements found on the baseline. Their oracle is able to find speedups ranging from 4.5 to 62.5 times for the different datasets they tested. Whereas the speedup of our oracle is generally below 1.2. Using imitation learning in such a situation leads to a much lower upperbound for the possible improvement that can be learned.

Furthermore, for imitation learning only a few sequences of each instance are useful for training. The other sequences evaluated during the process are discarded afterwards. Other approaches that could make use of these discarded sequences can turn out to be more efficient. A possible direction could be a surrogate model that is trained to compute the cost of a PL sequence in terms of execution time. In this case all sequences involved during the local search process can be of interest for training the model.

Another change to the approach lies with the scope of the work. In this work the changes were limited to only changing the order of the PL sequence in the algorithm. This decision made the focus clear during the early stages of this thesis and for the synthetic supply chain it was noticed that the changes in this aspect of the algorithm could lead to significant improvement in the execution time.

However, the large amount of instances where no significant improvement was found was ignored. Attempting to speedup HP by making changes in a different component of the algorithm could prove to be more successful. An example of such a component would be planning of a single PL.

5.1. Future Work

In this section we discuss some of the future directions that can be investigated to continue this work, starting with an issue with LTR model. The LTR model uses the learning to rank approach, where each PL is given a relevance score used when sorting the entire sequence of PLs. However, this model is currently limited by the library used, where the relevance score can only range between 0 and 4. This results in the model only being able to categorize each PL in one of the 5 classes of relevance. When labelling training data with more than 5 PLs in the PL sequence, some PLs will obtain the same label. An implementation with a different library should be considered to escape from this limitation, after which similar experiments can be run with the new model. An adaptation for the process could also be considered, where the LTR model is first used to separate the related PLs into 5 smaller groups, after which another model could be used to again rank the PLs within these smaller groups. This approach, however, still suffers from the problem when the number of PLs is larger, so a different library is suggested, such as this Python implementation of LambdaMart [1].

Another improvement to the existing process is related to the creation of the synthetic instances. In the current work this process only contains some basic changes and there is no guarantee that the instances within the training data or the validation data are sufficiently different. From the experiment with the different sizes for the training data, it was seen that the model performance of the models showed a decrease or stagnation with more instances used for the training data. This means that the additional instances were of no use for training, which could be due to this lack of variation in the new instances. The inclusion of more variables that are altered within these synthetic instances can increase the robustness of the model when used in practice. For the training data this variation could lead to learning more general patterns, whereas in the validation data the variation can contribute to more accurate scores for the model.

A large bottleneck of the whole process lies with the process of collecting of training data, where the local search process helps reduce the time required. However, the local search process has high likeliness to get stuck in a local optimum. Larger changes to the PL sequences benefit the search process in finding more varied results, while smaller changes can help in exploring the neighborhood. Initially, the local search process was created as a tool to help understand which small adjustments to a PL sequence contribute to a faster execution time. This lead to the local search process being designed with small changes at each iteration as the focus. However, this idea does not work well with the larger supply chains with many priority groups. An initial exploration into simply removing all these restrictions was done, by comparing the search process of the current local search with one where the order within a priority group is randomized for the new PL sequence, see Figure 5.1. The performance of the local search with the randomized orders is relatively similar to the current process for the synthetic supply chain as it is quite small. The only noticeable difference would be the higher and more frequent spikes, where the execution time would soar drastically, which the current local search process generally manages to avoid. Applying this to the larger instances resulted in the evaluation of PL sequences that required upwards of a few hours to resolve when such a spike is found. An approach needs to be found to figure out how large the changes within the local search process should be for a particular instance.

An additional change to the local search process could be to integrate the model training with the local search process. Once a small pool of training data has been collected, we could train a model on this data and use the trained model to predict a PL sequence for a new instance. This PL sequence would ideally be in a better neighbourhood than the initial PL sequence for this instance. This would speedup the local search process for subsequent instances.

Lastly, the model currently has no guarantee for the quality of the resulting planning. The only measure taken to nudge the model in the right direction is during the selection of training data. Only

[1]https://github.com/lezzago/LambdaMart

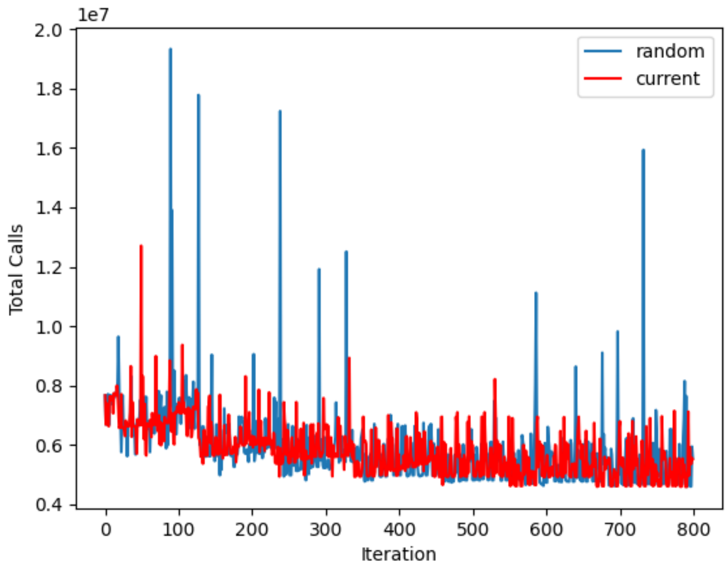


Figure 5.1: Results of the current Local Search (red) and the local search with randomizing the order within a priority group (blue)

PL sequences that do not result in significantly lower quality are selected. A possible future direction would be to treat the problem as multi-objective and selecting the PL sequences that optimize over the different objectives. The scoring function used in the evaluation to be multi-objective, rather than a single scoring function.

Bibliography

[1] Sunday A Afolalu et al. “Overview impact of planning in production of a manufacturing sector”. In: *IOP Conference Series: Materials Science and Engineering*. Vol. 1036. 1. IOP Publishing. 2021, p. 012060.

[2] *AI Lecture on search - State space search*. `https://cseweb.ucsd.edu/classes/sp07/cse150/lectures-pdf/l.newsearch.pdf`. Accessed: 2022-01-19.

[3] Alexandre Attia and Sharone Dayan. “Global overview of imitation learning”. In: *arXiv preprint arXiv:1801.06503* (2018).

[4] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. “Machine learning for combinatorial optimization: a methodological tour d’horizon”. In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[5] Christopher JC Burges. “From ranknet to lambdarank to lambdamart: An overview”. In: *Learning* 11.23-581 (2010), p. 81.

[6] Nikolay Dimitrov. “Surrogate models for parameterized representation of wake-induced loads in wind farms”. In: *Wind Energy* 22.10 (2019), pp. 1371–1389.

[7] Carmel Domshlak, Erez Karpas, and Shaul Markovitch. “Online speedup learning for optimal planning”. In: *Journal of Artificial Intelligence Research* 44 (2012), pp. 709–755.

[8] Alan Fern. *Speedup Learning.* 2010.

[9] Scott E Field et al. “Fast prediction and evaluation of gravitational waveforms using surrogate models”. In: *Physical Review X* 4.3 (2014), p. 031006.

[10] Martina Fischetti and Matteo Fischetti. “Matheuristics”. In: *Handbook of heuristics*. Springer, 2018, pp. 121–153.

[11] Jerome H Friedman and Jacqueline J Meulman. “Multiple additive regression trees with application in epidemiology”. In: *Statistics in medicine* 22.9 (2003), pp. 1365–1381.

[12] Yuan Gao, Lixing Yang, and Shukai Li. “Uncertain models on railway transportation planning problem”. In: *Applied Mathematical Modelling* 40.7-8 (2016), pp. 4921–4934.

[13] Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. “Learning to rank for synthesizing planning heuristics”. In: *arXiv preprint arXiv:1608.01302* (2016).

[14] Georgios P Georgiadis and Michael C Georgiadis. “Optimal planning of the COVID-19 vaccine supply chain”. In: *Vaccine* 39.37 (2021), pp. 5302–5312.

[15] Eduardo Guzman, Beatriz Andres, and Raul Poler. “Models and algorithms for production planning, scheduling and sequencing problems: A holistic framework and a systematic review”. In: *Journal of Industrial Information Integration* (2021), p. 100287.

[16] He He, Hal Daume III, and Jason M Eisner. “Learning to search in branch and bound algorithms”. In: *Advances in neural information processing systems* 27 (2014), pp. 3293–3301.

[17] Ahmed Hussein et al. “Imitation learning: A survey of learning methods”. In: *ACM Computing Surveys (CSUR)* 50.2 (2017), pp. 1–35.

[18] M Kranjčević et al. “Multiobjective optimization of the dynamic aperture using surrogate models based on artificial neural networks”. In: *Physical Review Accelerators and Beams* 24.1 (2021), p. 014601.

[19] Mehdi Lamiri, Frédéric Grimaud, and Xiaolan Xie. “Optimization methods for a stochastic surgery planning problem”. In: *International Journal of Production Economics* 120.2 (2009), pp. 400–410.

[20] Hang Li. “A short introduction to learning to rank”. In: *IEICE TRANSACTIONS on Information and Systems* 94.10 (2011), pp. 1854–1862.

[21] Donald Michie. ““Memo” functions and machine learning”. In: *Nature* 218.5136 (1968), pp. 19–22.

[22] Franco Modigliani and Franz E Hohn. “Production planning over time and the nature of the expectation and planning horizon”. In: *Econometrica, Journal of the Econometric Society* (1955), pp. 46–66.

[23] Mohammad Hossain Namaki et al. “Learning to speed up query planning in graph databases”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 27. 1. 2017.

[24] Xingyuan Pan and Vivek Srikumar. “Learning to speed up structured output prediction”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3996–4005.

[25] *Plan-space search*. https://www.inf.ed.ac.uk/teaching/courses/plan/slides/Plan-Space-Search-Slides.pdf. Accessed: 2022-01-19.

[26] Dnyanesh Rajpathak and Enrico Motta. “An ontological formalization of the planning task”. In: *International Conference on Formal Ontology in Information Systems (FOIS 2004)*. 2004, pp. 305–316.

[27] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[28] Moonsoo Shin et al. “A two-phased perishable inventory model for production planning in a food industry”. In: *Computers & Industrial Engineering* 133 (2019), pp. 175–185.

[29] Thomas Sillekens, Achim Koberstein, and Leena Suhl. “Aggregate production planning in the automotive industry with special consideration of workforce flexibility”. In: *International Journal of Production Research* 49.17 (2011), pp. 5055–5078.

[30] *Supply Chain Management Software Market Statistics: 2030*. Oct. 2021. url: https://www.alliedmarketresearch.com/supply-chain-management-software-market.

[31] Prasad Tadepalli and Balas K Natarajan. “A formal framework for speedup learning from problems and solutions”. In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 445–475.

[32] Rashmi Korlakai Vinayak and Ran Gilad-Bachrach. “Dart: Dropouts meet multiple additive regression trees”. In: *Artificial Intelligence and Statistics*. PMLR. 2015, pp. 489–497.

[33] Kaan Yilmaz and Neil Yorke-Smith. “A study of learning search approximation in mixed integer branch and bound: Node selection in scip”. In: *Ai* 2.2 (2021), pp. 150–178.

[34] Jian Zhang, Mahjoub Dridi, and Abdellah El Moudni. “Column-generation-based heuristic approaches to stochastic surgery scheduling with downstream capacity constraints”. In: *International Journal of Production Economics* 229 (2020), p. 107764.

[35] Terry Zimmerman and Subbarao Kambhampati. “Learning-assisted automated planning: Looking back, taking stock, going forward”. In: *AI Magazine* 24.2 (2003), pp. 73–73.

A
Results

A.1. Performance

A.1.1. Synthetic Supply Chain

LTR

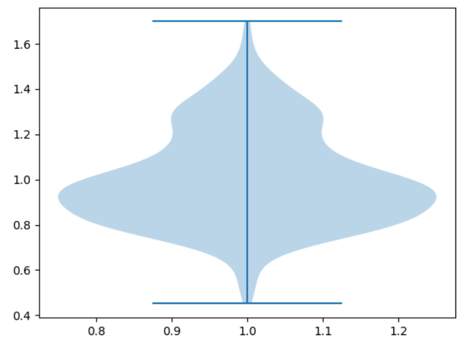


Figure A.1: The score for all 500 evaluated instance with the pairwise model

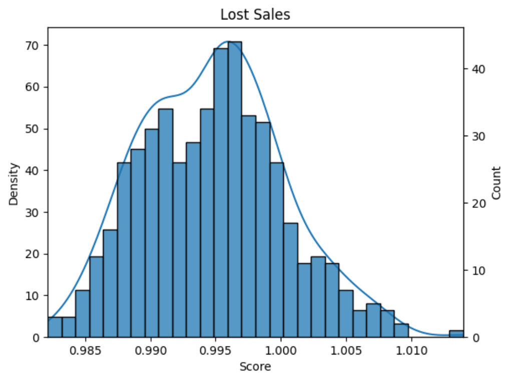


Figure A.2: The change in number lost sales, comparing the current heuristic with the pairwise model

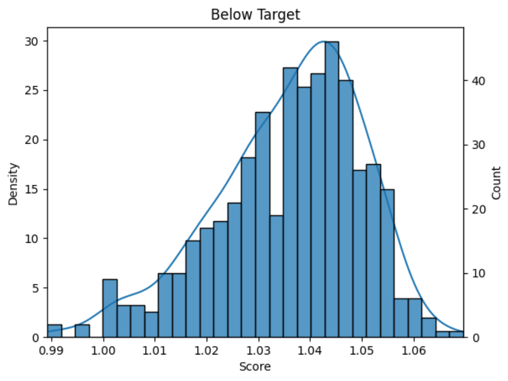


Figure A.3: The change in number below target inventories, comparing the current heuristic with the pairwise model

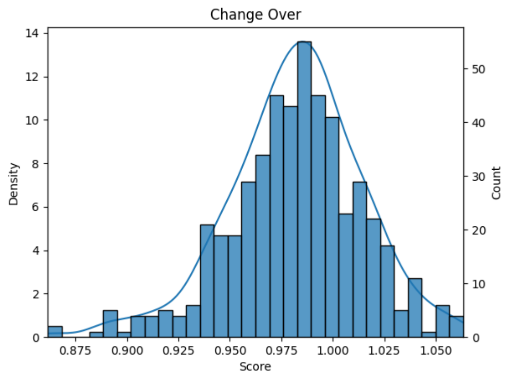


Figure A.4: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.5: Results for the synthetic supply chain using the LTR model

Pairwise

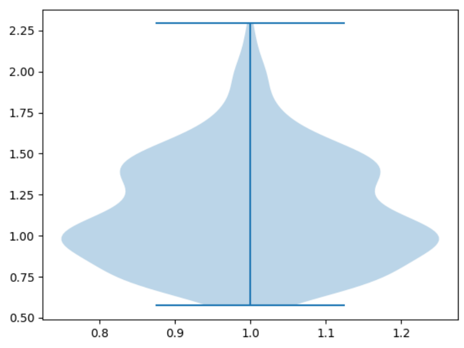


Figure A.6: The score for all 500 evaluated instance with the pairwise model

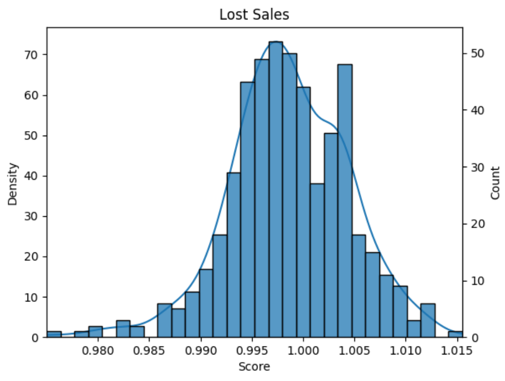


Figure A.7: The change in number lost sales, comparing the current heuristic with the pairwise model

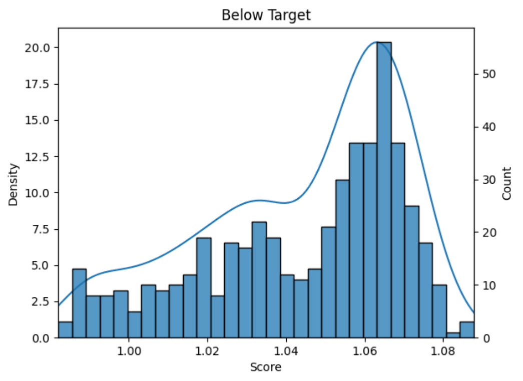


Figure A.8: The change in number below target inventories, comparing the current heuristic with the pairwise model

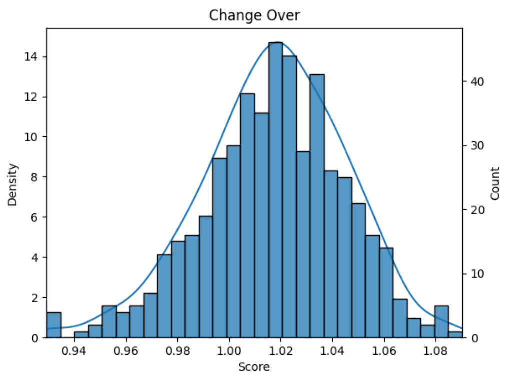


Figure A.9: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.10: Results for the synthetic supply chain using the pairwise model

A.1.2. Supply Chain A

LTR

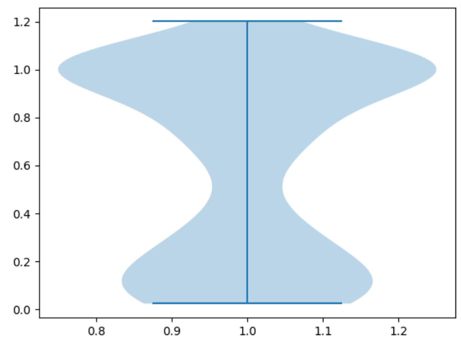


Figure A.11: The score for all 500 evaluated instance with the pairwise model

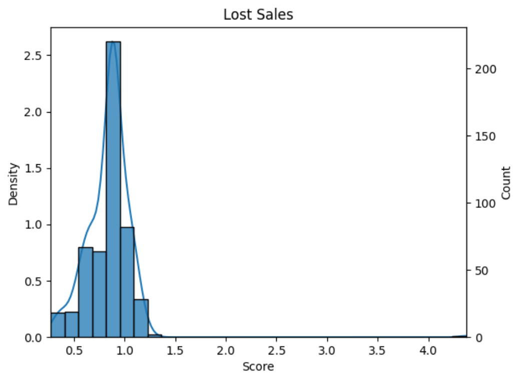


Figure A.12: The change in number lost sales, comparing the current heuristic with the pairwise model

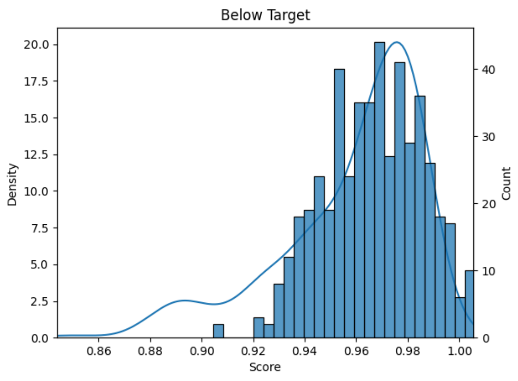


Figure A.13: The change in number below target inventories, comparing the current heuristic with the pairwise model

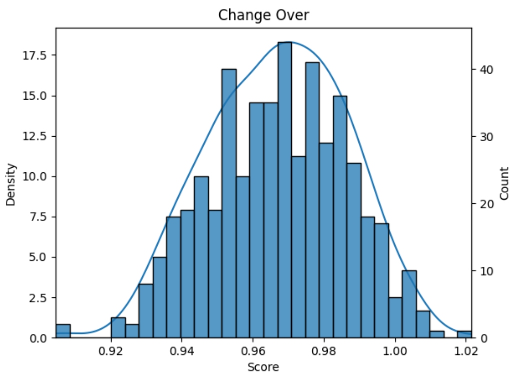


Figure A.14: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.15: Results for supply chain A using the LTR model

Pairwise

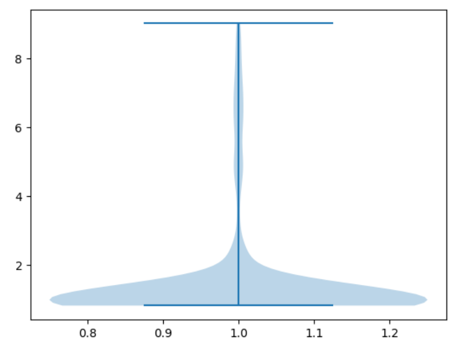


Figure A.16: The score for all 500 evaluated instance with the pairwise model

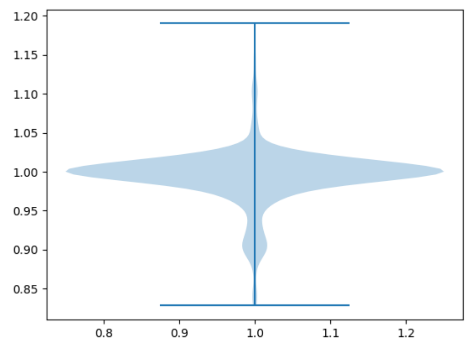


Figure A.17: The score for all 500 evaluated instance with the pairwise model, zoomed in on the instances with a score below 1.2

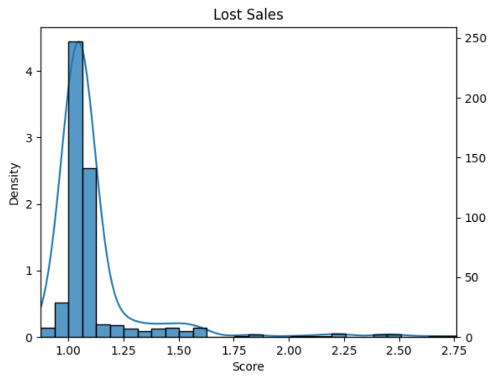


Figure A.18: The change in number lost sales, comparing the current heuristic with the pairwise model

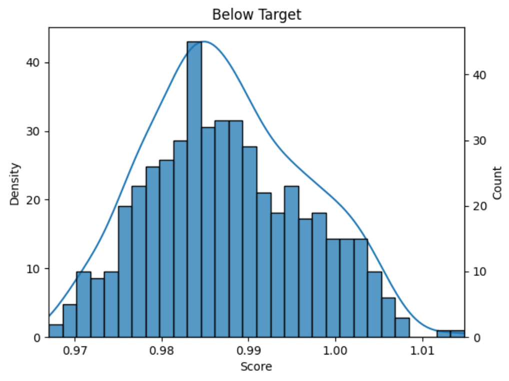


Figure A.19: The change in number below target inventories, comparing the current heuristic with the pairwise model

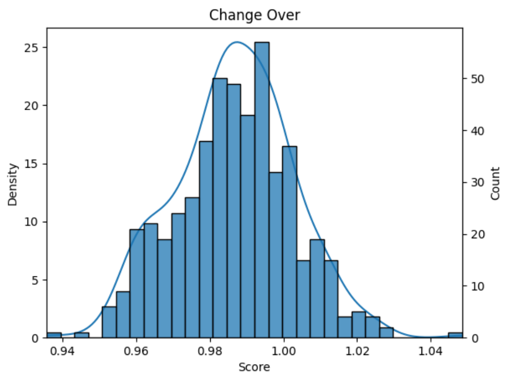


Figure A.20: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.21: Results for supply chain A using the pairwise model

A.1.3. Supply Chain B

LTR

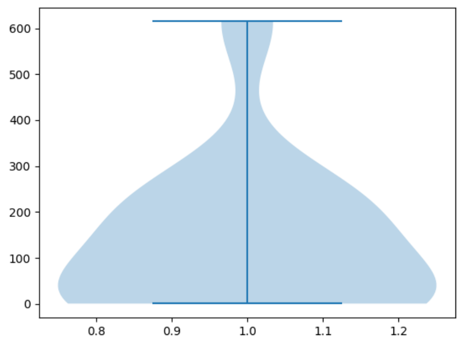


Figure A.22: The score for all evaluated instance with the LTR model

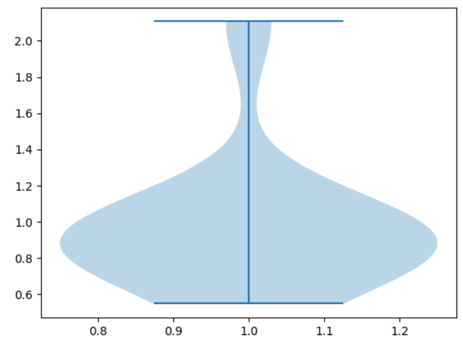


Figure A.23: The score for all evaluated instance with the LTR model, zoomed in on the instances with a score below 2

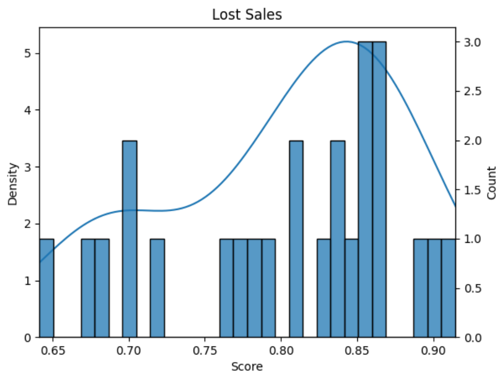


Figure A.24: The change in number lost sales, comparing the current heuristic with the LTR model

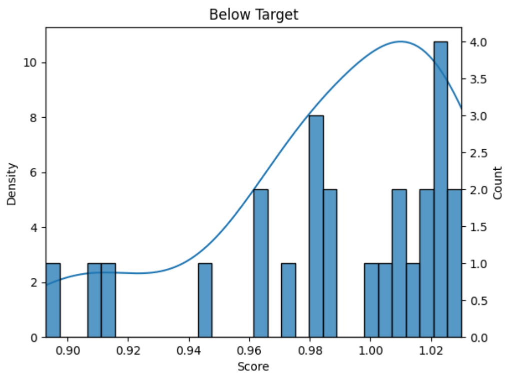


Figure A.25: The change in number below target inventories, comparing the current heuristic with the LTR model

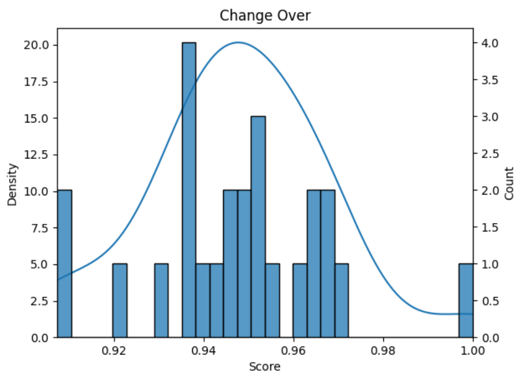


Figure A.26: The change in amount of change over, comparing the current heuristic with the LTR model

Figure A.27: Results for supply chain B using the LTR model

Pairwise

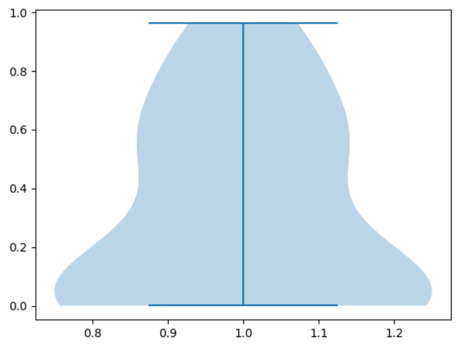


Figure A.28: The score for all evaluated instance with the pairwise model

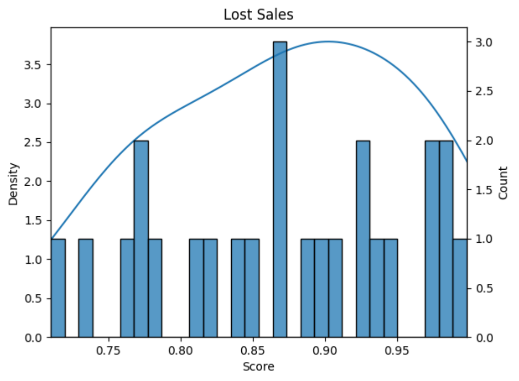


Figure A.29: The change in number lost sales, comparing the current heuristic with the pairwise model

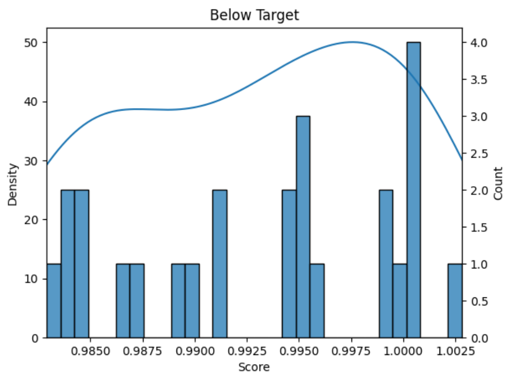


Figure A.30: The change in number below target inventories, comparing the current heuristic with the pairwise model

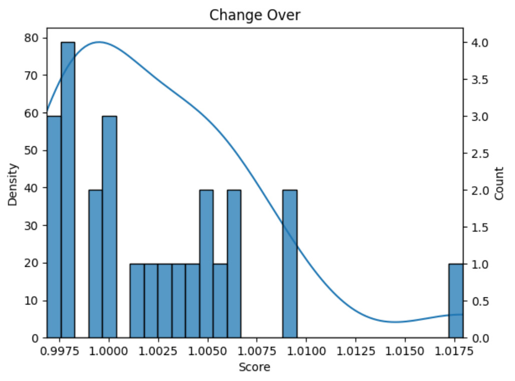


Figure A.31: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.32: Results for supply chain B using the pairwise model

A.1.4. Supply Chain C

Pairwise

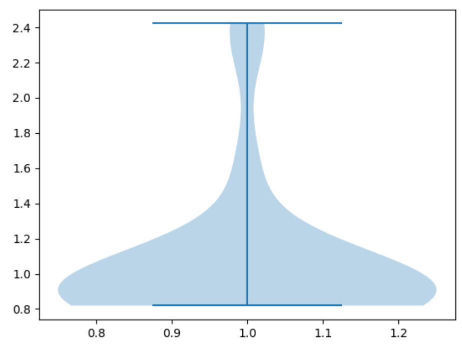


Figure A.33: The score for all evaluated instance with the pairwise model

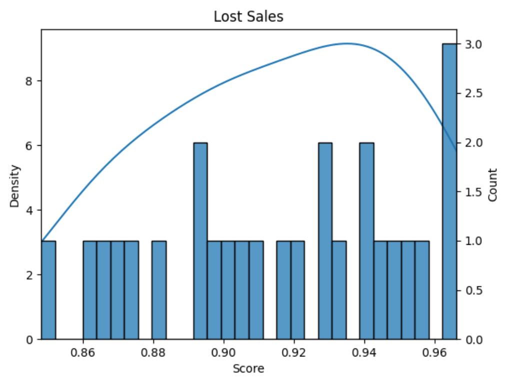


Figure A.34: The change in number lost sales, comparing the current heuristic with the pairwise model

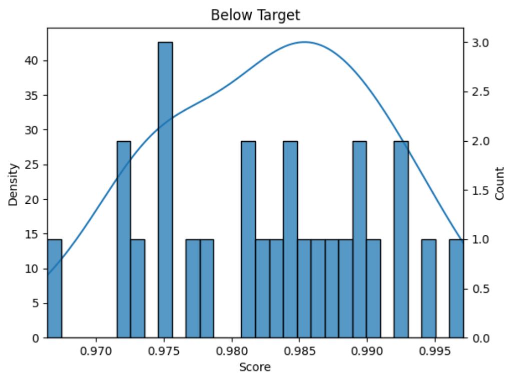


Figure A.35: The change in number below target inventories, comparing the current heuristic with the pairwise model

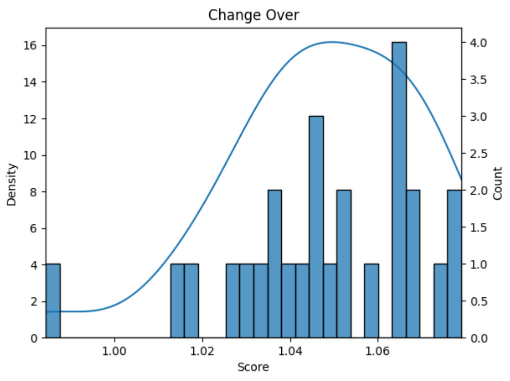


Figure A.36: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.37: Results for supply chain C using the pairwise model

A.1.5. Supply Chain D

LTR

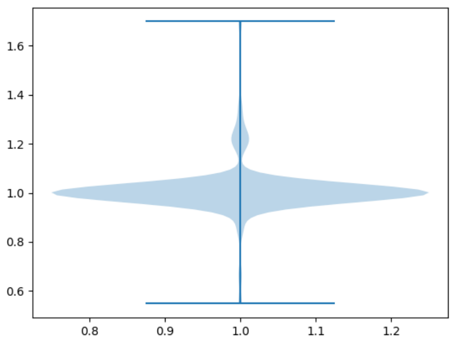


Figure A.38: The score for all evaluated instance with the pairwise model

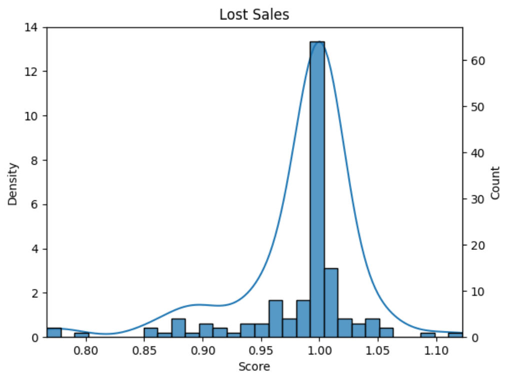


Figure A.39: The change in number lost sales, comparing the current heuristic with the pairwise model

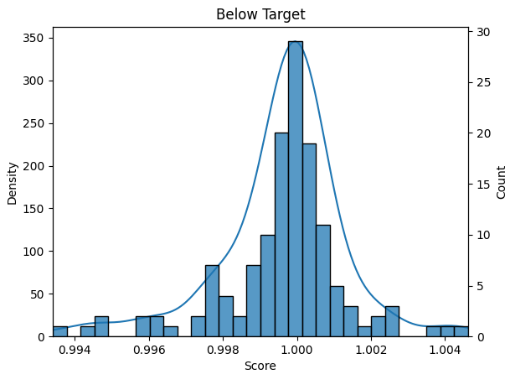


Figure A.40: The change in number below target inventories, comparing the current heuristic with the pairwise model

Figure A.41: Results for supply chain D using the LTR model

Pairwise

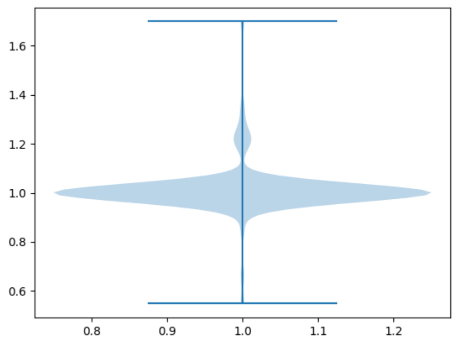


Figure A.42: The score for all evaluated instance with the pairwise model

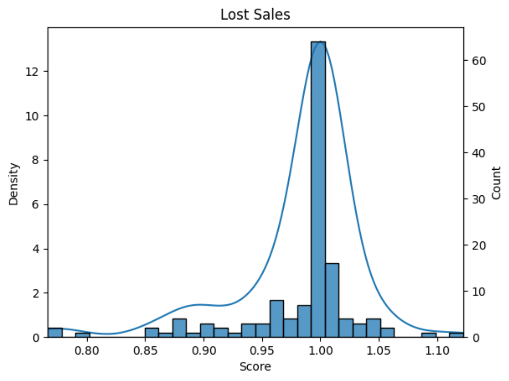


Figure A.43: The change in number lost sales, comparing the current heuristic with the pairwise model

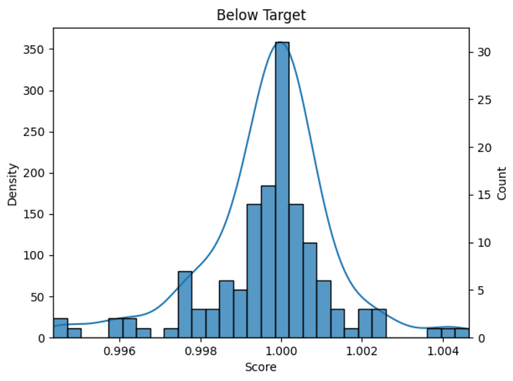


Figure A.44: The change in number below target inventories, comparing the current heuristic with the pairwise model

Figure A.45: Results for supply chain D using the pairwise model

A.2. Robustness

A.2.1. Different sizes for training data

Synthetic Supply Chain

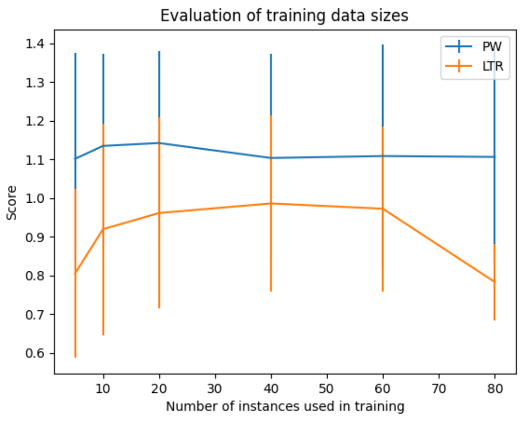


Figure A.46: Varying sizes for the training data chosen as [5, 10, 20, 40, 60, 80]. Each configuration is evaluated on the same 100 instances.

Supply Chain A

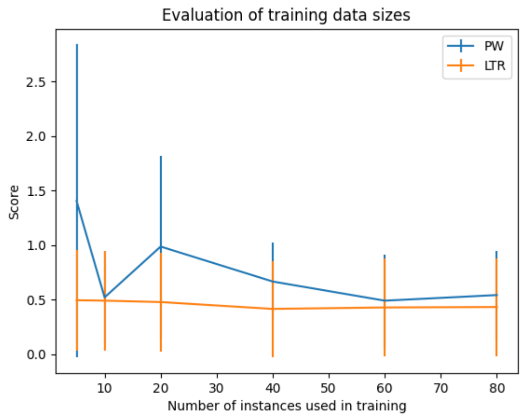


Figure A.47: Varying sizes for the training data chosen as [5, 10, 20, 40, 60, 80]. Each configuration is evaluated on the same 100 instances.

A.2.2. Supply Chain A and B

LTR

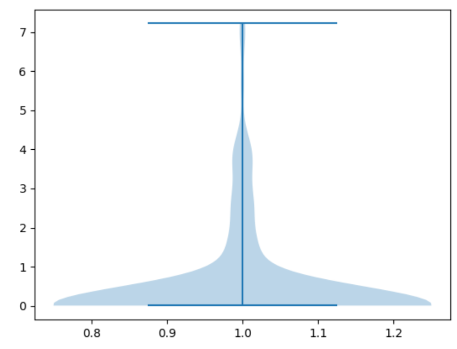


Figure A.48: The score for all evaluated instance with the LTR model

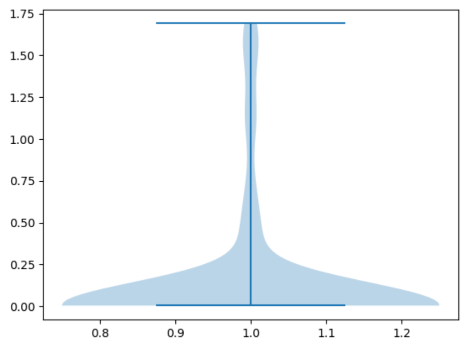


Figure A.49: The score for all evaluated instance with the LTR model, zoomed in on the instances with a score below 2

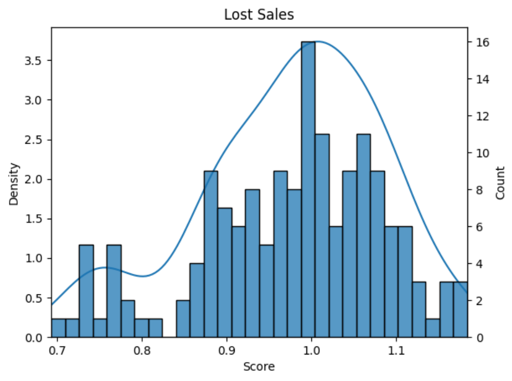


Figure A.50: The change in number lost sales, comparing the current heuristic with the LTR model

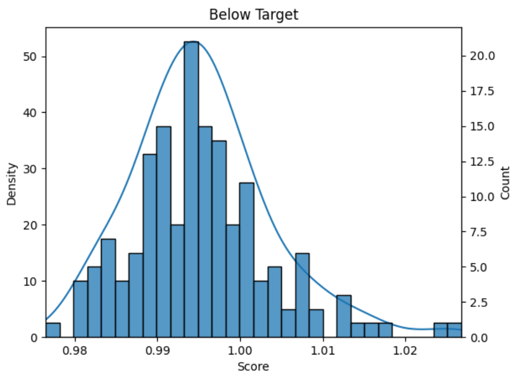


Figure A.51: The change in number below target inventories, comparing the current heuristic with the LTR model

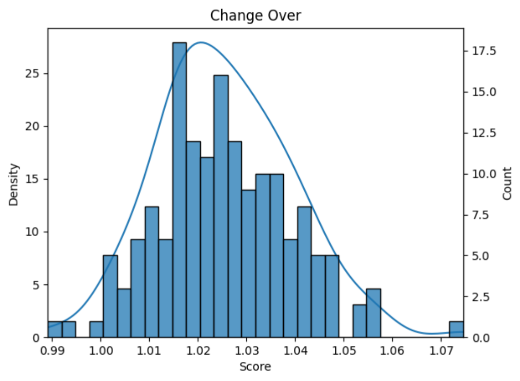


Figure A.52: The change in amount of change over, comparing the current heuristic with the LTR model

Figure A.53: Results for supply chain B using the LTR model

Pairwise

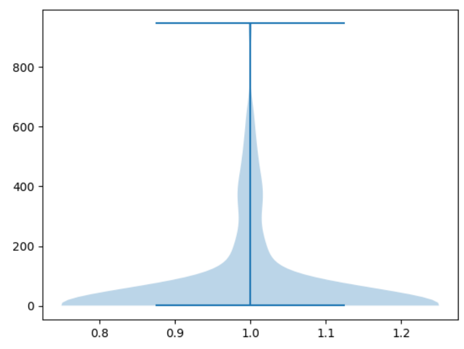


Figure A.54: The score for all evaluated instance with the pairwise model

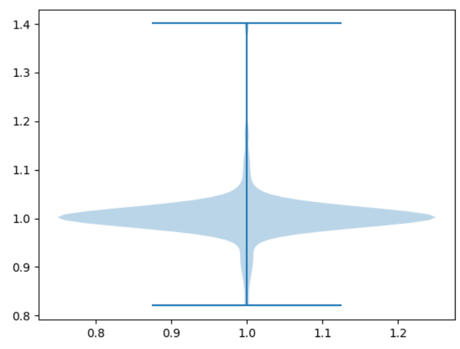


Figure A.55: The score for all 500 evaluated instance with the pairwise model, zoomed in on the instances with a score below 2

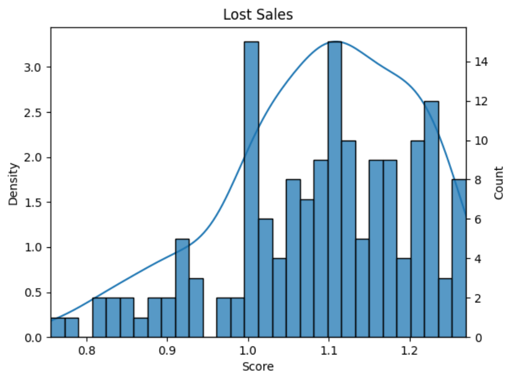


Figure A.56: The change in number lost sales, comparing the current heuristic with the pairwise model

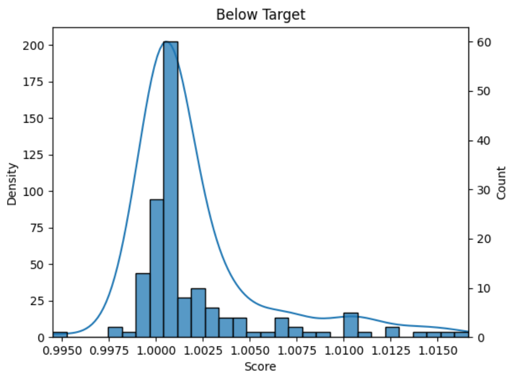


Figure A.57: The change in number below target inventories, comparing the current heuristic with the pairwise model

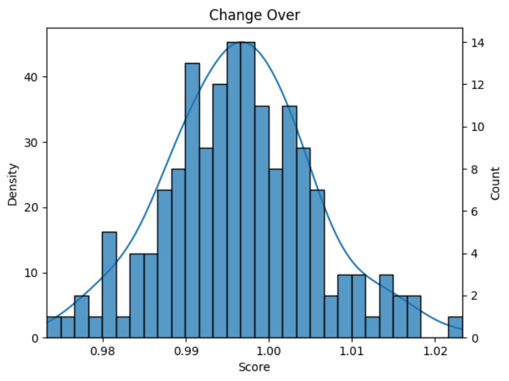


Figure A.58: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.59: Results for supply chain B using the pairwise model

A.2.3. Results on different synthetic instances

Larger Changes

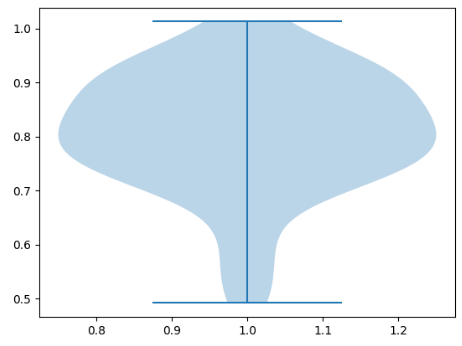


Figure A.60: The score for all 200 evaluated instance with the LTR model

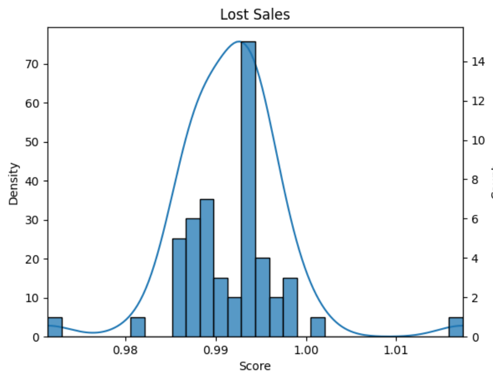


Figure A.61: The change in number lost sales, comparing the current heuristic with the LTR model

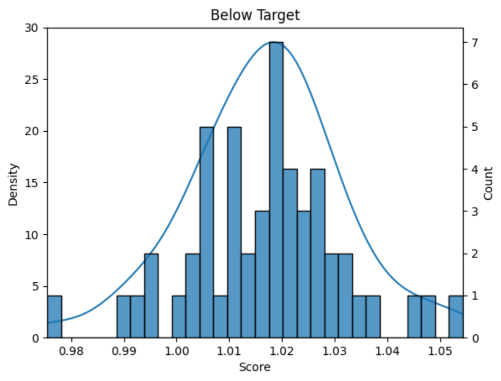


Figure A.62: The change in number below target inventories, comparing the current heuristic with the LTR model

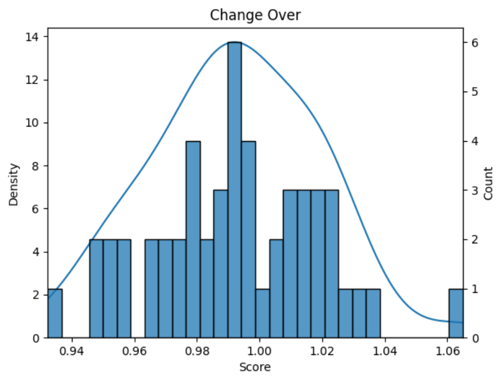


Figure A.63: The change in amount of change over, comparing the current heuristic with the LTR model

Figure A.64: Results for a LTR model trained on the synthetic supply chain when evaluating on synthetic instances with changes to daily demand with a factor of [0.7, 1.3] and a factor for production rate of [0.9, 1.1].

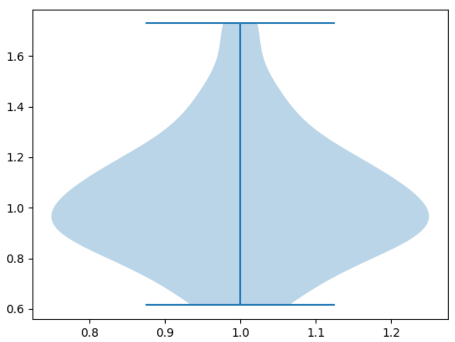


Figure A.65: The score for all 200 evaluated instance with the pairwise model

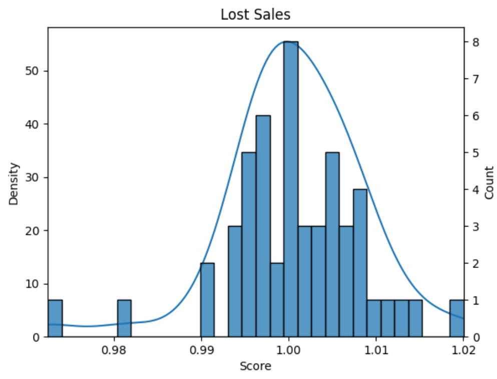


Figure A.66: The change in number lost sales, comparing the current heuristic with the pairwise model

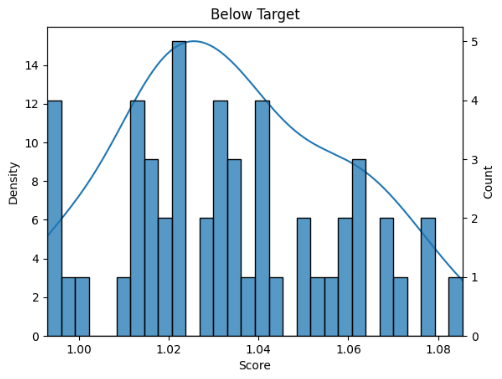


Figure A.67: The change in number below target inventories, comparing the current heuristic with the pairwise model

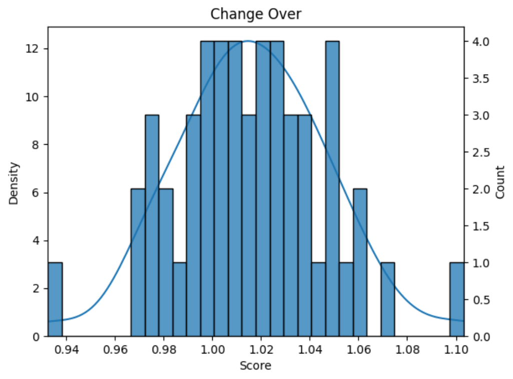


Figure A.68: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.69: Results for a pairwise model trained on the synthetic supply chain when evaluating on synthetic instances with changes to daily demand with a factor of [0.7, 1.3] and a factor for production rate of [0.9, 1.1].

Smaller Changes

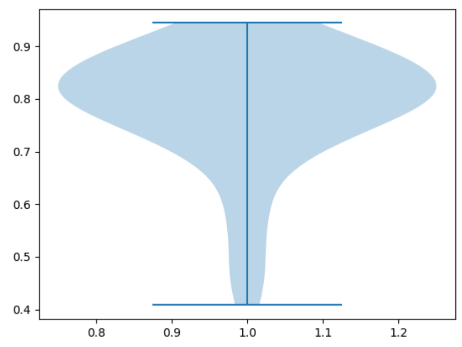


Figure A.70: The score for all 200 evaluated instance with the LTR model

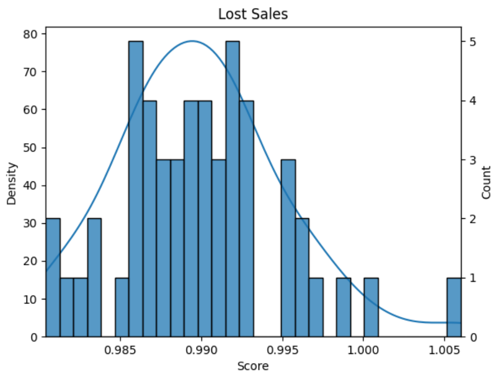


Figure A.71: The change in number lost sales, comparing the current heuristic with the LTR model

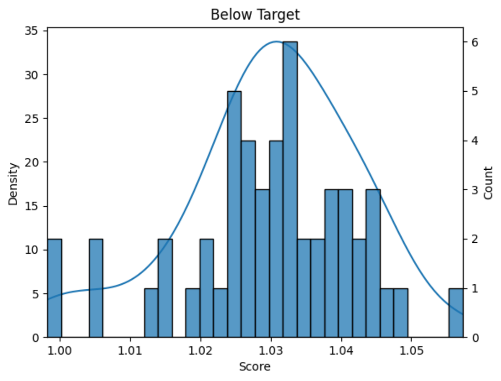


Figure A.72: The change in number below target inventories, comparing the current heuristic with the LTR model

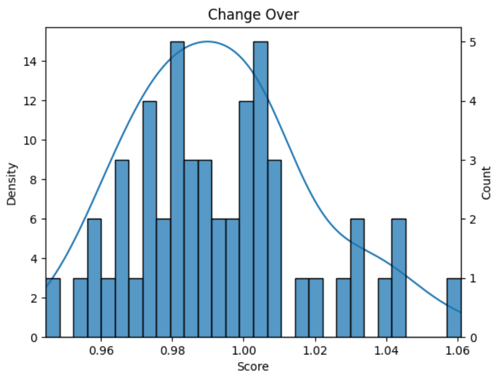


Figure A.73: The change in amount of change over, comparing the current heuristic with the LTR model

Figure A.74: Results for a LTR model trained on the synthetic supply chain when evaluating on synthetic instances with changes to daily demand with a factor of [0.925, 1.075] and a factor for production rate of [0.975, 1.025].

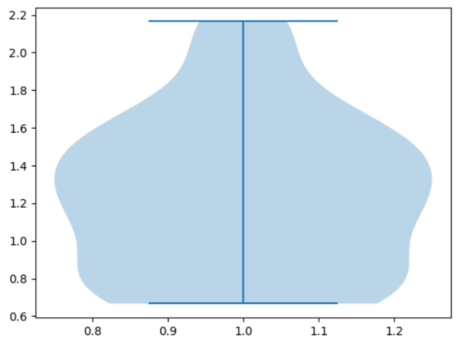


Figure A.75: The score for all 200 evaluated instance with the pairwise model

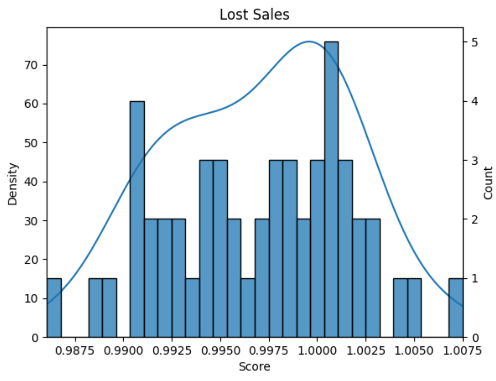


Figure A.76: The change in number lost sales, comparing the current heuristic with the pairwise model

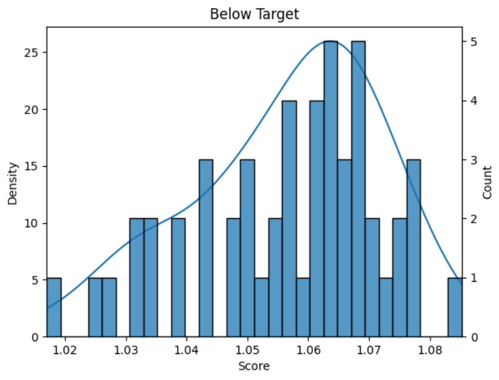


Figure A.77: The change in number below target inventories, comparing the current heuristic with the pairwise model

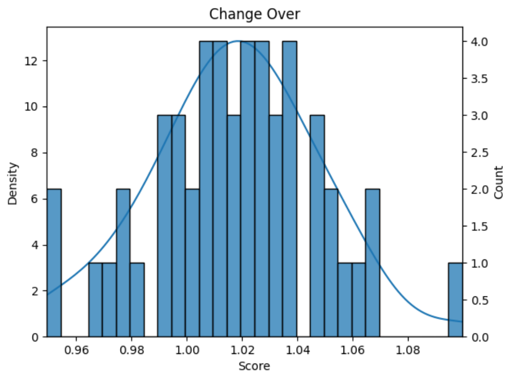


Figure A.78: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.79: Results for a pairwise model trained on the synthetic supply chain when evaluating on synthetic instances with changes to daily demand with a factor of [0.925, 1.075] and a factor for production rate of [0.975, 1.025].

A.2.4. Results from different supply chains

Synthetic supply chain with model trained on supply chain A

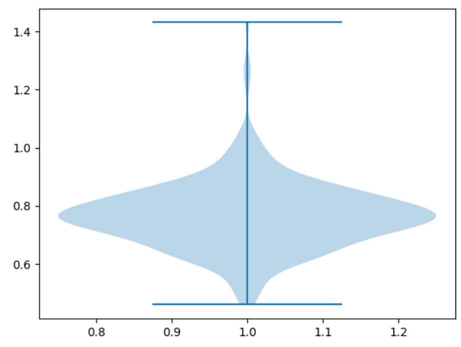


Figure A.80: The score for all 200 evaluated instance with the LTR model

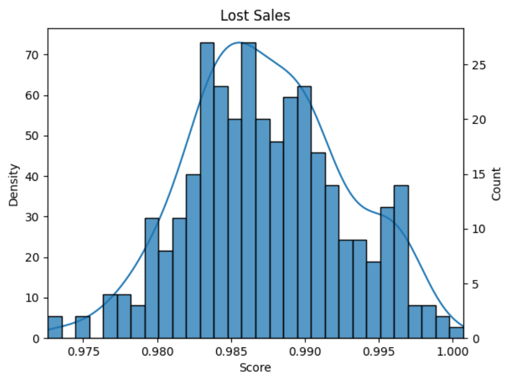


Figure A.81: The change in number lost sales, comparing the current heuristic with the LTR model

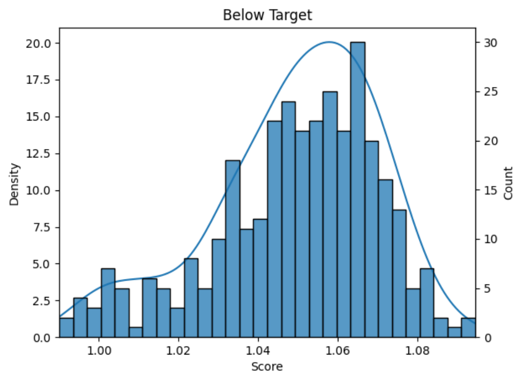


Figure A.82: The change in number below target inventories, comparing the current heuristic with the LTR model

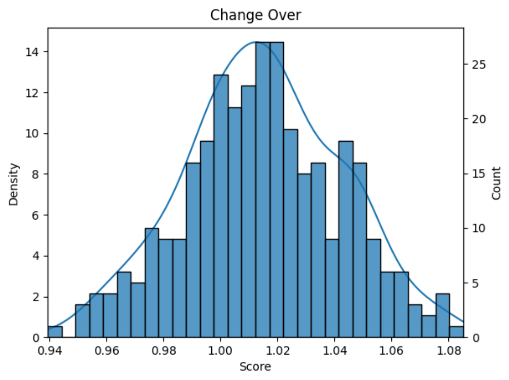


Figure A.83: The change in amount of change over, comparing the current heuristic with the LTR model

Figure A.84: Results for a LTR model trained on supply chain A and evaluated on instances from the synthetic supply chain.

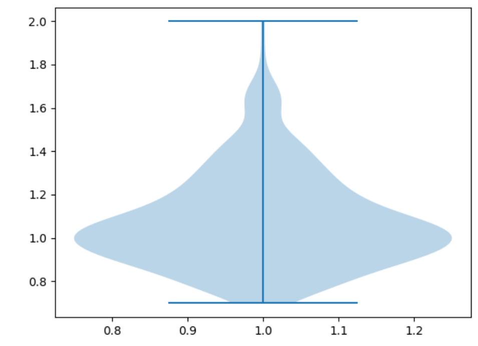


Figure A.85: The score for all 200 evaluated instance with the pairwise model

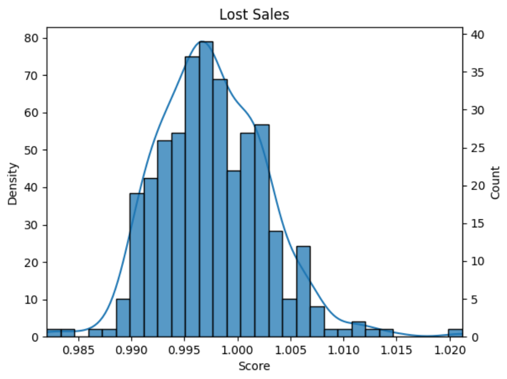


Figure A.86: The change in number lost sales, comparing the current heuristic with the pairwise model

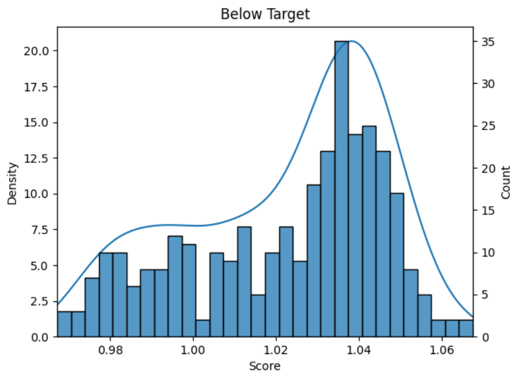


Figure A.87: The change in number below target inventories, comparing the current heuristic with the pairwise model

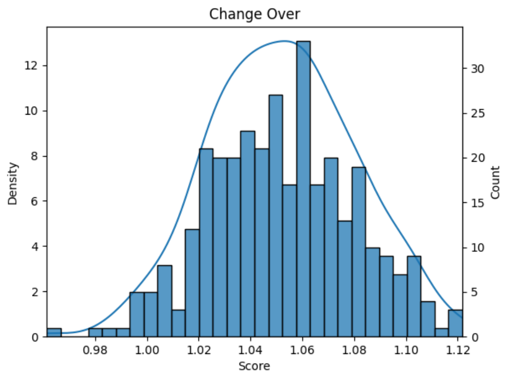


Figure A.88: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.89: Results for a pairwise model trained on supply chain A and evaluated on instances from the synthetic supply chain.

Supply chain A with model trained on synthetic supply chain

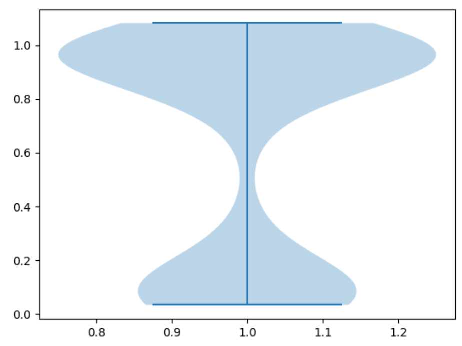


Figure A.90: The score for all 200 evaluated instance with the LTR model

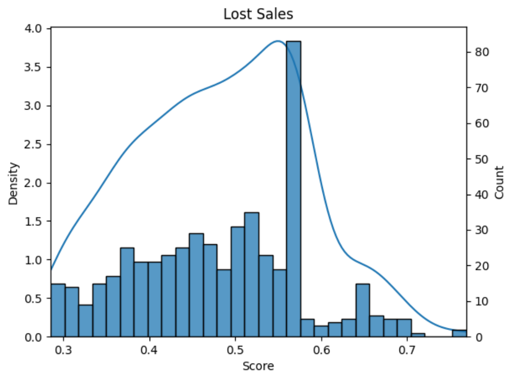


Figure A.91: The change in number lost sales, comparing the current heuristic with the LTR model

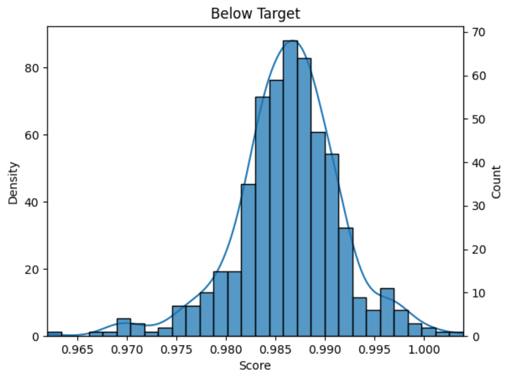


Figure A.92: The change in number below target inventories, comparing the current heuristic with the LTR model

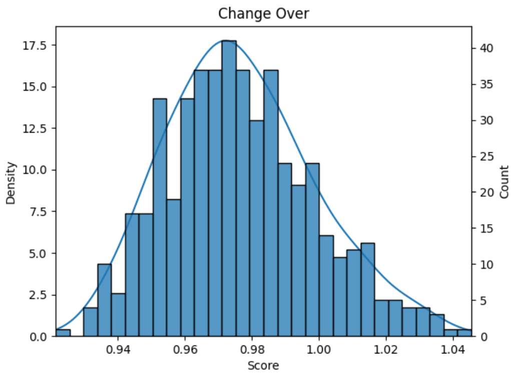


Figure A.93: The change in amount of change over, comparing the current heuristic with the LTR model

Figure A.94: Results for a LTR model trained on the synthetic supply chain and evaluated on instances from supply chain A.

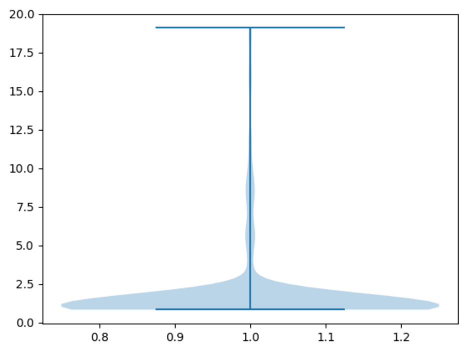


Figure A.95: The score for all 200 evaluated instance with the pairwise model

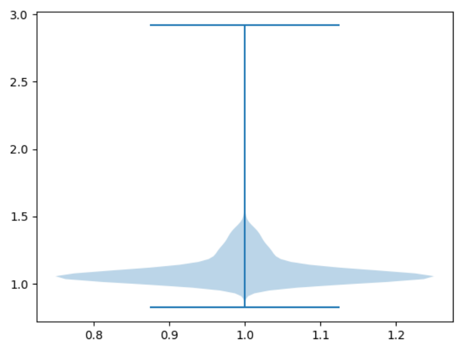


Figure A.96: The scores zoomed in on those below 3

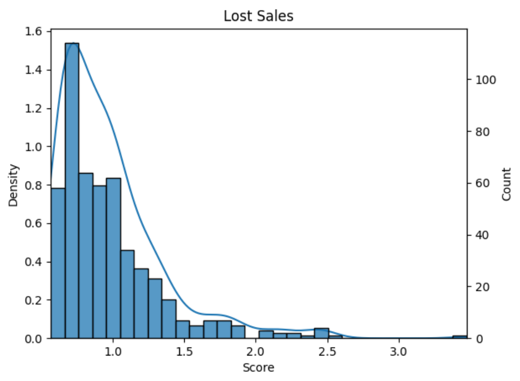


Figure A.97: The change in number lost sales, comparing the current heuristic with the pairwise model

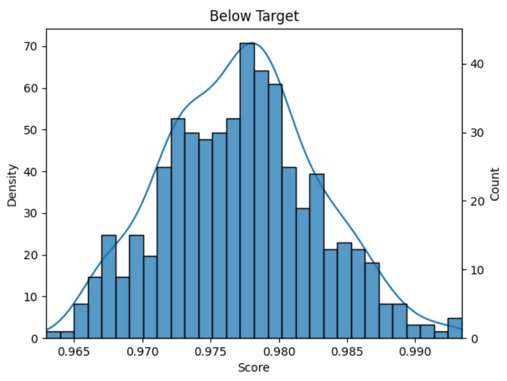


Figure A.98: The change in number below target inventories, comparing the current heuristic with the pairwise model

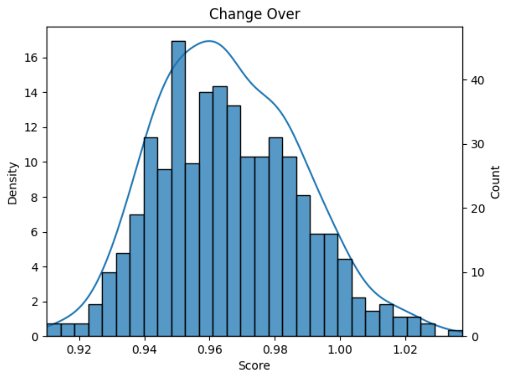


Figure A.99: The change in amount of change over, comparing the current heuristic with the pairwise model

Figure A.100: Results for a pairwise model trained on the synthetic supply chain and evaluated on instances from supply chain A.