**ERASMUS UNIVERSITY ROTTERDAM**

Erasmus School of Economics

**Bachelor Thesis, Seminar in Quantitative Logistics**

Econometrics and Operations Research

July 7, 2019

# A peak period heuristic for the Train Unit Assignment Problem

**Hoogenband, T. van den**

456605th

*Supervisor:*
Lieshout, R.N. van

*First reader:*
Dollevoet, T.A.B.

## Abstract

Train operators are responsible for the scheduling of train units on every trip they carry out. They have to decide the train unit decomposition that can suffice the demand of seats on every trip. This problem is known as the Train Unit Assignment Problem. This thesis is concerned with solving this problem, by first testing the effective peak period heuristic opposed in Cacchiani et al. (2019). The heuristic is finetuned by analyzing the best settings in the heuristic, for the instances used in this thesis. Thereafter, this thesis gives three improvements to the heuristic that strongly decrease the runtime. One improvement is suggested in particular, because it excels in terms of runtime.

# Contents

**1 Introduction** **2**
1.1 Contribution . . . . . 2

**2 Literature** **2**

**3 Problem description** **3**

**4 Methodology** **4**
4.1 Getting a lower bound . . . . . 4
4.2 Constructive phase . . . . . 5
4.2.1 Single depot vehicle scheduling problem . . . . . 7
4.3 Feasibility phase . . . . . 8
4.4 Combining phases in the peak period heuristic . . . . . 8

**5 Improvements to the heuristic** **9**
5.1 Fine tuning . . . . . 10
5.2 Additional stopping criteria in the assignment problem of $CJ$ . . . . . 10
5.3 Improved trip assignment in the feasibility phase . . . . . 11
5.3.1 Implementation in the heuristic . . . . . 12

**6 Results** **13**
6.1 Comparing set-ups . . . . . 13
6.2 Improvements . . . . . 15

**7 Conclusion** **16**

**8 Discussion** **17**

# 1 Introduction

When train operators are managing the railway system in a certain region, they have the responsibility to provide every trip in the region with enough train units. A trip has enough train units, if these train units combined contain enough seats to satisfy the demand on the trip. When a train operator knows how many seats a train on each trip needs, it is also known which combinations of different types of train units together satisfy the demand, for every trip. The train operator has to decide how to assign their train units to the trips, such that all the demand is satisfied. This problem is generally known as the *Train Unit Assignment Problem* (TUAP). Train operators need to know the least cost total decomposition of train units to buy. Solving the TUAP provides this information, with the goal of minimizing the total train unit purchase cost in mind. The total cost of the needed train units is especially important for train operators that want to take control over a new region. Namely, which train operator is going to take control over the region is decided by means of a bidding competition between the potential train operators.

Of course, operating a railway system will provide the company with more costs than only the payment of train units. But the train unit costs are generally very high, and train units can not be acquired for a short time, as mentioned in Peeters and Kroon (2008). This means that the company will have to pay for unused train units in uneventful times, because a higher amount of train units is needed in the peak period. Moreover, Lin and Kwan (2014) suggest that the crew planning is partly decided by the TUAP. Having a good solution to the TUAP during the peak period is therefore really important for train operators.

Several contents will be discussed in this thesis, starting with the contribution in Chapter 1.1. After that, the existing literature related to this thesis is summarized in Chapter 2. The problem that is tackled is then clarified in Chapter 3. Chapter 4 elaborates the methodology that is used to get the desired results of Chapter 6. Chapter 6 also shows the results of different improvements to the heuristic, that are explained in Chapter 5. The conclusions that are a result of this thesis, are stated in Chapter 7. Finally, the results are discussed in Chapter 8.

## 1.1 Contribution

This thesis is a rework of Cacchiani et al. (2019), in which a heuristic for solving the TUAP is proposed. The aim is to both investigate and extend their recommended heuristic for solving the TUAP. One of the things the extensions focus on, is reducing the runtime of the heuristic. Cacchiani et al. (2019) show that all forms of the TUAP that are used, are NP-hard. It is therefore very useful to get insight and improvement in the running time of the heuristic.

The idea of the heuristic is to get the lower bound cost, and to iteratively let the upper bound solution decrease. After the iteration process is done, the gap between the lower bound cost and the upper bound cost is preferably zero. If not, the best solution cost found is equal to the upper bound. The lower bound cost is found by solving a LP-model and an ILP-model for a specific set of trips. Then the upper bound cost is considered multiple times by solving LP-models in the heuristic procedure. Although small gaps were found in Cacchiani et al. (2019), this thesis aims to reduce the upper bound cost by making changes in the used heuristic.

To improve the heuristic in terms of solution cost and runtime, the main causes of high upper bounds and long runtimes have to be identified. In order to investigate those causes, the heuristic as in Cacchiani et al. (2019) is first tested on multiple data sets. Once the bottlenecks are detected, purposeful extensions are developed.

# 2 Literature

The train unit assignment problem is a well-studied problem, especially when small variations are taken into account. For example, similar problems exist for cars, ships and such, instead of train units. Railway optimization problems in general are studied intensively. An overview of

optimization models for the most commonly studied passenger railway transportation problems is made by Huisman et al. (2005). The crew planning of the train units is an example of a railway optimization process that is related to the TUAP.

This thesis, however, is only concerned with the TUAP, so only existing literature about that specific problem is mentioned here. Given a timetable and the passengers' seat demand, a model and branch-and-price algorithm determine an allocation of rolling stock to the daily trips in Peeters and Kroon (2008). Lin and Kwan (2014) solve the same problem as in Peeters and Kroon (2008) with a model of an integer fixed-charge multicommodity flow problem, by using this in a heuristic. This article is followed by Lin and Kwan (2016), in which the problem is solved with a branch-and-price algorithm instead. As an extension to Peeters and Kroon (2008), combining and splitting trains at stations of different lengths is taken into account in Fioole et al. (2006).

Another form of a TUAP is described by Schrijver (1993). One day with trips is considered, on which the passenger seat demand must be satisfied for every trip. The problem is solved by integer linear programming. Borndörfer et al. (2015) give a mixed integer program for the rolling stock problem, in which maintenance is also considered. The problem is solved by means of an algorithm. When only the adjustment of the rolling stock planning is considered, because of urgent maintenance requests, a multicommodity flow type model can be solved, as suggested by Maróti and Kroon (2005). Other papers concerning rescheduling of the rolling stock planning because of maintenance are Giacco et al. (2014) and Wagenaar et al. (2017). The paper Lusby et al. (2017) is concerned with rescheduling of the rolling stock planning in general, where the problem is solved with a branch-and-price algorithm. In Cordeau et al. (2001) a similar problem for freight trains is introduced. Instead of using enough train units to satisfy passengers' seat demand, this problem is about deciding the minimum number of locomotives, in order to give every train enough power to transport the weight of the freight.

A slightly different problem, and with that, another solution approach is considered by Abbink et al. (2004). Here the number of available train units is known, and given this number, the best allocation of train units has to be found in order to give as many travelers as possible a seat. An article that focuses on specific rules that must be considered for coupling and uncoupling train units, as opposed to this thesis, is Alfieri et al. (2006).

There are also articles that connect the TUAP with other closely related problems. One of them is Haahr and Lusby (2017), in which the rolling stock scheduling problem is combined with a train unit shunting problem.

This thesis resembles Cacchiani et al. (2019) the most, obviously, as it is a rework of that article. The exact version of this TUAP is used by these authors twice before, in Cacchiani et al. (2010) and Cacchiani et al. (2013). The first one solves the integer linear programming model that is constructed by means of a developed heuristic. Due to the quite long running time of this procedure, a faster heuristic based on the Lagrangian relaxation of the problem is suggested in Cacchiani et al. (2013). However, the quality of the solutions in the latter article tend to slightly decrease when bigger instances are considered. Finally, the proposed heuristic in Cacchiani et al. (2019) is supposed to give good, fast solutions.

## 3 Problem description

In the TUAP there is a set of *trips* $J = \{1, ..., N\}$ and a set of *train units* $I = \{1, ..., M\}$. By solving the TUAP, the aim is to find a least cost assignment of the train units to the trips. This is done by sequencing every train unit to different trips in an efficient way. Train units that arrive at a certain train station can thereafter be used for other trips leaving the same station. The assumption is that an arrived train unit needs $t_j$ minutes of time before it can depart on another trip. Another assumption is that, in order to meet demand, all different types of train units can be combined with each other, and no specific order is needed. As opposed to other related works, deadheading is not taken into consideration in this thesis. In other words, train

units can not travel to other train stations without being assigned to a trip $j \in J$. Ultimately, the goal is to find the least cost set of train units $I$ with which the TUAP can be solved.

In the set of trips $J$, the total number of trips to be covered is given by $N$. Each trip $j \in J$ has the following information: its departure (set off) time $s_j$, its arrival time $a_j$, the arrival and departure station, the demanded number of seats $d_j$, and the rail line number. Each rail line can handle a maximum train length of $l_j$. This parameter has subscript $j$ of a trip, because each trip runs over a single rail line.

There are $p$ different types of train units available for purchase, each $k = 1, ..., p$ indicates a different type of train unit. The yearly cost for a single train unit of type $k$ is $C^k$, the number of available seats inside a train unit of type $k$ is $S^k$ and the length of a train unit is $L^k$.

In the TUAP, these are the restrictions that must be satisfied:

-*Demand* The demand of every trip $j \in J$ must be covered.

-*Sequencing* Trips $z, j \in J$ can only be performed with the same train unit, if and only if:

1. the destination station of trip $z$ is the same as the departure station of trip $j$.
2. $s_j \geq a_z + t_z$, meaning that there is enough time for an arrived train unit to get ready for the next departure.

-*Length* The total length of the train units combined on trip $j$ can be $l_j$ at most.

# 4 Methodology

In this chapter, the heuristic as proposed in Cacchiani et al. (2019) is described. They state that the heuristic basically consists of three parts: getting an initial solution lower bound, checking whether this lower bound is suitable as a solution for the whole instance (e.g. the upper bound equals the lower bound), and working towards a feasible solution if the upper bound and lower bound differ. These three parts are elaborated in Chapter 4.1, 4.2 and 4.3, respectively. The methodology for getting a lower bound on the peak period schedule cost (PP-LB) was first introduced in Cacchiani et al. (2010). The constructive phase of 4.2 and the feasibility phase of 4.3 are the building blocks of the effective heuristic in Cacchiani et al. (2019). How to combine the three parts in the heuristic, is shown in Chapter 4.4.

## 4.1 Getting a lower bound

The first step in getting the PP-LB is based on the notion of *incompatible* trips. The incompatible trips are the trips that are impossible to sequence, because of the sequencing and demand restrictions mentioned in Chapter 3. These trips are really important, because distinct train units have to be used for each incompatible trip. Therefore, the train units needed for all trips together are at least those train units needed to serve all the incompatible trips.

The idea is to start with reducing the set of trips $J$ to a set of trips $S$ that only contains the incompatible trips. The PP-LB is then obtained by assigning the trips in $S$ to train units in the most efficient way. The procedure to obtain $S$ starts with making a directed graph $G = (V, A)$, where the nodes in $V$ are the trips in $J$. Each node has corresponding demand $d_j$. The nodes are connected by the arcs in set $A$, where an arc $(z, j) \in A$ exists if and only if for two trips $z, j \in J$, the two requirements for the sequencing restriction described in Chapter 3 are met. To get set $S$, the maximum-weight stable set in $G$ has to be constructed. This is the same as finding the minimum number of paths such that each node in $V$ is covered by at least $d_j$ paths. By adding a sink $\tau$ and a source $\theta$ to the set of nodes to obtain $\bar{V}$, a minimum flow problem with the demands on the nodes can be solved. The source node $\theta$ is connected to the nodes in $V$ without in-going arcs and the sink node $\tau$ is connected to all nodes in $V$ without out-going arcs.

The addition of these arcs to $A$ gives the new set of arcs $\bar{A}$. Here in-going and out-going arcs of node $j$ are represented by $\delta^-(j)$ and $\delta^+(j)$, respectively. Furthermore, a continuous variable $f_{zj}$ is introduced, which equals the flow of paths on arc $(z,j) \in \bar{A}$. The linear programming model is shown below:

$$\min \quad \sum_{(\theta,j)\in\delta^+(\theta)} f_{\theta j} \tag{1}$$

$$\text{s.t.} \quad \sum_{(z,j)\in\delta^-(j)} f_{zj} = \sum_{(j,z)\in\delta^+(j)} f_{jz} \qquad j \in V \tag{2}$$

$$\sum_{(z,j)\in\delta^-(j)} f_{zj} \geq d_j \qquad j \in V \tag{3}$$

$$f_{zj} \geq 0 \qquad (z,j) \in \bar{A} \tag{4}$$

The objective function (1) minimizes the flow leaving the source $\theta$. Constraints (2) make sure that no flow is lost for all nodes except the sink and the source. Restrictions (3) make sure that the demand is satisfied for all nodes in $V$. Finally, (4) makes sure that the variables in the model are continuous and positive. With integer demands on the nodes, integer flows on the arcs will be attained.

By solving the model (1)-(4), the maximum-weight stable set S can be constructed. Because the paths $d_j$ decide the weight, this is done by taking the dual variables of the constraints that are concerned with the demand, constraints (3). These dual variables select the nodes that will construct set $S$ of incompatible trips, if they are strictly positive valued. Now the train units have to be assigned to the trips in $S$, to get the PP-LB. To solve the assignment problem, integer variable $w_j^k$ is introduced. This variable equals the number of train units of type $k$ that cover trip $j$. The following integer linear programming model is attained:

$$\min \quad \sum_{k=1}^{p} \sum_{j\in S} C^k w_j^k \tag{5}$$

$$\text{s.t.} \quad \sum_{k=1}^{p} S^k w_j^k \geq d_j \qquad j \in S \tag{6}$$

$$\sum_{k=1}^{p} L^k w_j^k \leq l_j \qquad j \in S \tag{7}$$

$$w_j^k \in \mathbb{N} \qquad k = 1, ..., p, \quad j \in S \tag{8}$$

The goal here is to minimize the total cost of train units, given in (5). The constraints (6) ensure that there are enough seats on every trip in $S$. Additional constraints (7) have to be added to make sure that trains do not become too long for the rail line. The variables are integer valued because of (8). The models (1)-(4) and (5)-(8) together satisfy the demand, sequencing and length constraints of Chapter 3, for all the incompatible trips. Therefore, the objective function (5) gives the PP-LB. By summing the values of $w_j^k$ over $j$ for every train unit type $k$ separately, the minimum number of train units per type is obtained. These values are noted as $\bar{d}^k$.

### 4.2 Constructive phase

In the constructive phase, train units are assigned to so-called *critical* and *uncritical* trips. It is tested whether or not the values of $\bar{d}^k$ found in the PP-LB are sufficient for serving the critical and uncritical trips together. If these values are indeed sufficient, an optimal solution is found. If these values are not sufficient, additional train units are needed, and an upper bound higher than the PP-LB will be found. To test this, the results of the models in Chapter 4.1 are needed. The set of trips $J$ is initially separated into the critical trips $CJ := S$ and the uncritical trips

$UJ := J \setminus CJ$. First the assignment of the critical trips is considered. There are five steps included in this assignment problem, that will be repeated for $n_c$ iterations or until all critical trips have been assigned:

1. Take a single trip $j$ of the set $CJ$ at a time, beginning with the first trip in the set.

2. Make a set $FSEL$ that contains all subsets of the available train units, in which each of those subsets has a total capacity of seats that is at least equal to the demanded number of seats for trip $j$, and in which each of those subsets has a combined train unit length of at most $l_j$. Then order these subsets by ascending total seat capacity and choose the first one of those subsets, such that the subset with the capacity closest to $d_j$ is chosen first. The available train units are given by the $\bar{d}^k$ values. Multiple train units of the same type $k$ can appear in the same subset. The subset of train units selected in this step is called $TU$.

3. For every train unit type in $TU$, solve the SDVSP (9)-(12) given in Chapter 4.2.1. This model decides the currently optimal scheduling of train units of this type, to all the trips that make use of this train unit type. With that, the model gives information about how many train units of this type are needed to serve all trips that use this train unit type, called SDVSP(k).

4. The $TU$ chosen in step 2 can only be accepted if SDVSP(k) does not exceed $\bar{d}^k$ for any of the types $k$ that are part of $TU$. If it is accepted, go to step 1 with the successor trip of trip $j$, and follow the steps again. If not, go back to step 2 and select the next best subset of train units $TU$, before following the steps again. If none of the subsets considered in step 2 gets accepted in step 4, let trip $j$ remain uncovered and go to the successor trip of the current trip in step 1. Go to step five if there are no trips left to consider.

5. (a) Stop the iteration process if none of the critical trips remains uncovered. If that is not the case, up the iteration counter by one and change the order of the trips in $CJ$. This different order is determined by considering the uncovered trips $u \in CJ$. If $u \in S$, this trip is placed just before the covered trips in $S$. Else, it is placed just after the trips in $S$ and just before the covered trips in $CJ$. With this new set $CJ$, reset all the trips in the set to uncovered and start at step 1 again.

   (b) If the number of iterations equals $n_c$ for the first time while there are still trips left uncovered, reset the iteration counter $n_c$ and set all the trips in the set to uncovered. Now repeat steps 1-5, but change step 5 in this second stage. Now move every trip left uncovered before the covered trips in set $S$, regardless of the uncovered trip belonging to $S$ or not.

   (c) If the number of iterations equals $n_c$ for the second time while there are still trips left uncovered, stop the procedure and save the trips left uncovered.

The pseudocode for the 5 steps of the assignment procedure can be seen in Figure 1. The assignment of the trips in $UJ$ follows the same procedure as the trips in $CJ$, only step 5 is disregarded. So when assigning the uncritical trips, the pseudocode only consists of lines 1-4 and 6-26 together, and $CJ$ becomes $UJ$.

```
1   CJ := set of critical trips
2   uncovered := set of uncovered trips
3   i := 0
4   partTwo := FALSE
5   while i < n_c and uncovered ≠ ∅ do
6       uncovered := CJ
7       for j ∈ CJ do
8           covered(j) := FALSE
9           FSEL(j) := ordered set of possible subsets of train units for trip j
10          accept := TRUE
11          while FSEL(j) ≠ ∅ and covered(j) = FALSE do
12              select the first subset in FSEL(j), called TU
13              for k ∈ TU do
14                  solve SDVSP for type k
15                  if SDVSP(k) > d̄^k then
16                      accept := FALSE
17                      remove first subset of FSEL(j)
18                      break
19                  end
20              end
21              if accept then
22                  covered(j) := TRUE
23                  uncovered := uncovered\{ j}
24              end
25          end
26      end
27      if i = n_c − 1 and partTwo = FALSE do
28          i := −1
29          partTwo := TRUE
30      end
31      if partTwo = FALSE do
32          change trip order in CJ as described in step 5.(a)
33      end
34      if partTwo do
35          change trip order in CJ as described in step 5.(b)
36      end
37  end
```

Figure 1: The five step procedure for trip assignment, in the form of pseudocode

#### 4.2.1 Single depot vehicle scheduling problem

In step 3 of the assignment problem, the *Single Depot Vehicle Scheduling Problem* (SDVSP) is needed. Solving this problem decides whether or not the selected $TU$ can be accepted. To solve this problem for every type of train unit that is used, again a graph is needed. Every trip that uses this type of train unit becomes a node in this graph, and a source node $\sigma$ and sink node $\tau$ are added. All trips that satisfy the sequencing rules of Chapter 3 are connected by an arc. Furthermore, every trip is connected with an arc to both the source and sink node. The trips are part of set $V'$ and all the arcs are part of set $A'$. The source and sink node are added to set $V'$, noted as set $V''$. Now a directed graph $G = (V'', A')$ can be made, with costs $c_{zj}$ on the arcs. The cost on an arc from the source to a trip equals $C^k$, the cost on every other arc is set to 0. Some trips can be in need of multiple train units of the same type, which has to be taken into account. Therefore, the parameter $n_j^k$ is introduced, which equals the number of train units of type $k$ that cover trip $j$. There are $n_j^k - 1$ dummy nodes in the graph, for every value of $n_j^k$ that exceeds 1. This ensures that trips covered by multiple train units of the same type will get distinct train units of that type, because the dummy nodes and original node violate the sequencing restriction. The continuous variable $y_{jz}$ equals the number of times a train unit of this type uses arc $(z, j) \in A'$. This variable equals 0 or 1, with demanded values on the nodes being the integer value of 1. The model that has to be solved now becomes:

$$\min \quad \sum_{(z,j)\in A'} c_{zj}\, y_{zj} \tag{9}$$

$$\text{s.t.} \quad \sum_{(j,z)\in\delta^+(j)} y_{jz} = 1 \qquad j \in V' \tag{10}$$

$$\sum_{(z,j)\in\delta^-(j)} y_{zj} = 1 \qquad j \in V' \tag{11}$$

$$y_{zj} \geq 0 \qquad (z,j) \in A' \tag{12}$$

The minimum cost for the SDVSP is attained by (9). By dividing this objective function with $C^k$, the needed number of train units of this type is obtained, called SVSP(k). The constraints in (10) and (11) together ensure that a train unit of type $k$ enters and leaves the node of each (dummy) trip. Namely as before, in-going and out-going arcs of node $j$ are represented by $\delta^-(j)$ and $\delta^+(j)$, respectively. Finally, restrictions (12) make sure that the variables in the model are continuous and at least zero.

### 4.3 Feasibility phase

After the assignment of the trips in $CJ$ and $UJ$, there are no trips left uncovered in an ideal situation. In this case, the values of $\bar{d}^k$ were enough to satisfy the demand for the whole set of trips $J$, a feasible solution has been found. The total cost of the upper bound equals the PP-LB in this case. However, this is only the case if there are no uncovered trips coming out of the constructive phase.

If not all the trips in $J$ can be covered with the amount of train units $\bar{d}^k$ (in other words, there are uncovered trips left at the end of the constructive phase for at least one of the sets $CJ$ and $UJ$), additional train units have to be added to find a feasible solution. The trips left uncovered will be assigned to the additional train units, by using the same procedure that is used for the uncritical trips, as mentioned in Chapter 4.2. The $\bar{d}^k$ values are increased by a big amount for every train unit type, such that every trip in $J$ will be covered after this phase. The additional amount is big enough, if no subset of train units for an uncovered trip is denied because of the need of additional train units. Adding the objective functions of the last executed SDVSP models for each type $k$, gives the feasible solution cost.

### 4.4 Combining phases in the peak period heuristic

The constructive phase and the feasibility phase are generally not executed only once. They are sequentially done at most $n_{iter}$ times, if no optimal solution is found before that number is reached. The order of the trips in set $CJ$ is determined by the set that comes out of the constructive phase of the last iteration. The order of the trips in $UJ$ stays chronological in every iteration. The uncovered trips of both the critical and the uncritical sets are added to the end of $CJ$ at the end of the constructive phase, as long as no optimal solution is found. At the end of the feasibility phase, the lowest solution cost is updated if a feasible solution better than the old one is found.

It is important that the assignment of the trips in $CJ$ and $UJ$ is done sequentially in the constructive phase. The trip assignment in the feasibility phase is done after the assignment of the trips in the constructive phase. Assigning trips in $CJ$, $UJ$ and the set of uncovered trips, can be seen as three stages of trip assignments. Each stage has a different set of trips to be assigned, but the train units assigned to the trips already covered have to be remembered in every consecutive assignment stage. The subsets of train units already assigned have to be saved, in order to solve the correct SDVSP in every trip assignment. For the trips covered, the values of $n_j^k$ have to be stored as well. Only when a new iteration of the heuristic starts, the list of train units already assigned to trips and the values of $n_j^k$ can be reset. The procedure of the

peak period heuristic can be seen in Figure 2.

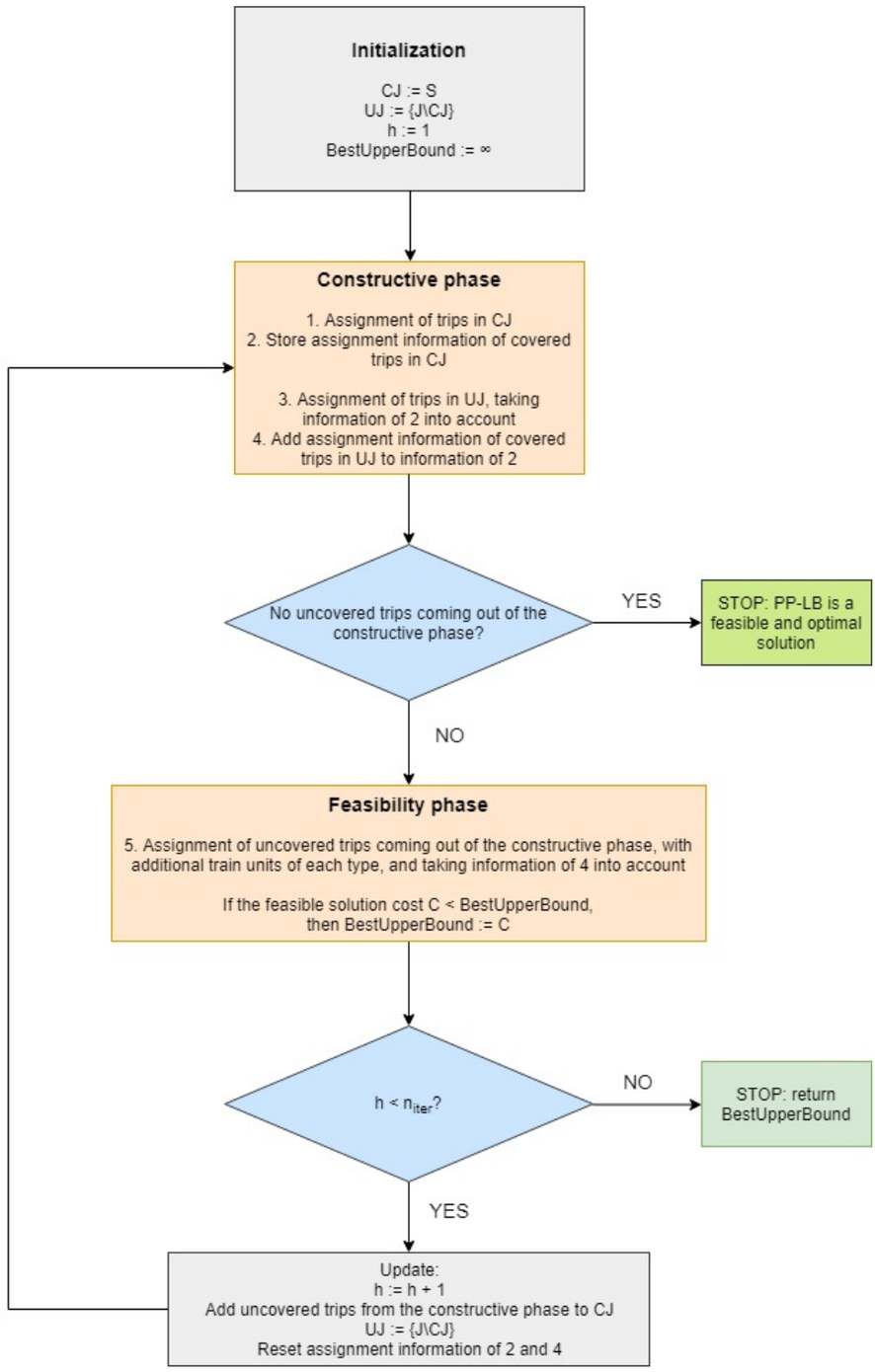


Figure 2: A flowchart of the peak period heuristic

## 5 Improvements to the heuristic

The heuristic as proposed by Cacchiani et al. (2019) is mainly focused on covering all the trips with only the use of the number of train units found in the PP-LB solution. There is no guarantee that this is possible, or that the number of trips left uncovered decreases in each iteration of $n_{iter}$. However, the only stopping criterion of the heuristic, besides the maximum number of iterations being reached, is when no trips are left uncovered after the constructive phase. Therefore, the efficiency of the algorithm decreases when finding an upper bound equal to the lower bound seems unlikely. Especially the running time of the heuristic can be decreased by slight changes in the heuristic, as explained in Chapter 5.1. Furthermore, Chapter 5.2 explains how possibly better and certainly faster solutions can be obtained by adding new stopping criteria to the heuristic.

The methodology of Chapter 4.3 stated that the values of $\bar{d}^k$ have to be increased by an

amount big enough, such that no subset of train units will be denied in the feasibility phase. This is different than the methodology used by Cacchiani et al. (2019), they increased the values of $\bar{d}^k$ to the maximum number of train units available per type. No such increase is possible in this thesis, as no maximum availability of train units is given. The used methodology may cause bad trip assignments in the feasibility phase, as the first selected $TU$ will be accepted for all the trips. Therefore, a different trip assignment procedure for the feasibility phase is given in Chapter 5.3.

## 5.1 Fine tuning

With no maximum on the train units available for purchase, the set $FSEL$ could become enormous if the train length restriction is not taken into account. Even if the length restriction is considered, one could argue that not having a maximum amount of train units per subset would not lead to better solutions. Take for instance the case that the demand on trips can all be satisfied with 2 train units, when taken individually. When including subsets of train units in $FSEL$ of size 5, this will most likely lead to unnecessary attempts to solve the SDVSP models. Namely, when assigning 2 train units to a trip is not possible with the $\bar{d}^k$ values, assigning even more train units to this trip is in general also not possible with these $\bar{d}^k$ values. So the subsets of size 3,4 and 5 all have to be tested by executing SDVSP models, while these additional executions most likely will not lead to a feasible solution. Because of this, the assumption is that the runtime of the heuristic will increase without getting better solutions, when the maximum size of subsets in $FSEL$ increases. However, the heuristic will miss the opportunity to find better solutions, if the size of the subsets becomes too restricted. To test this assumption, the heuristic will be executed with different maximum sizes $a$ of train unit subsets, noted by $FSEL$ $\#a$.

Another assumption that can be tested is that the second stage, as described in 5.(b) of the trip assignment procedure in Chapter 4.2, leads to additional runtime without giving better solutions. To test this assumption, the results of the heuristic are gathered twice: once with inclusion of the second stage, and once without using this second stage.

The last adjustment that can possibly lead to better results, is changing the order of the train unit subsets in $FSEL$. Initially, these subsets are given in ascending order of total seat capacity that these train units together provide. However, when dealing with a cost minimization problem as in this thesis, intuitively it might be more suitable to order the subsets in ascending order of the total cost of the train units in a subset. The heuristic is therefore also performed with the subsets in $FSEL$ being ordered by ascending total cost.

## 5.2 Additional stopping criteria in the assignment problem of $CJ$

After each iteration in the assignment of critical trips, there are zero or more trips left uncovered. When all these trips are added to the set of uncovered trips, the size of this set can be obtained, called $uncoveredTripSize$.

In an extension to the heuristic, the first step is to use the *set-up* that seems the most optimal after studying the results of Chapter 5.1. Here a set-up instruction states the following characteristics of the heuristic: the number $a$ used in $FSEL$ $\#a$, whether or not a second stage is included, whether the subsets in $FSEL$ are ordered by ascending capacity or cost. With the optimal set-up, additional stopping conditions are added to the heuristic. The goal is to cover as many trips as possible with the given $\bar{d}^k$ values in the PP-LB solution. In order to work towards this goal, stopping conditions need to lead to a decreasing pattern of the $uncoveredTripSize$ values in the assignment of the critical trips. When this pattern appears to go in an increasing direction, the heuristic has to stop with the assignment of the trips in which $uncoveredTripSize$ was lowest. In order to do so, in each iteration of $n_c$ the following information has to be saved:

- The $uncoveredTripSize$ of this iteration, the last iteration and the second to last iteration.

– The set of uncovered trips in this iteration, in the last iteration and in the second to last iteration.

– The trip assignment per train unit type for the trips covered in this, the last and the second to last iteration.

– The values $n_j^k$ in the trip assignment of the covered trips in this, the last and the second to last iteration.

Take the last three parts together in a collection for every iteration of $n_c$, such that an information collection for the current, last and second to last iteration is obtained. With the first part of information, three stopping conditions can be checked after each iteration. When the values of $uncoveredTripSize$ are $A$, $B$ and $C$ in three consecutive iterations ($C$ is the the value in the current iteration, $A$ the value in the second to last iteration), the assignment of critical trips has to be stopped if:

1. $B < A$ and $B < C$, stop the assignment problem with the information collection of the last iteration, belonging to the value $B$.

2. $A < B \leq C$, stop the assignment problem with the information collection of the second to last iteration, belonging to the value $A$.

3. $A = B$ and $B < C$, stop the assignment problem with the information collection of the last iteration, belonging to the value $B$.

The intuition of these stopping conditions is to get better solutions. In the original heuristic, the assignment of trips in $CJ$ ends with the trips left uncovered after iteration $n_c$, if none of the iterations before that result in covering all the critical trips. There is no guarantee that the number of trips left uncovered after iteration $n_c$ is lower than the number of trips left uncovered in any other iteration. In the contrary, the additional stopping conditions focus on ending the trip assignment with the lowest number of trips left uncovered. In most iterations of $n_{iter}$, this extension will therefore lead to smaller values of $uncoveredTripSize$ in the feasibility phase. Less uncovered trips will most likely need less additional train units to become covered, resulting in better solutions.

Besides that, the runtime could decrease drastically as well, because the assignment problem of critical trips is stopped earlier when one of the stopping conditions is met before the iteration counter reaches $n_c$. These additional stopping conditions are added to the five step procedure of Chapter 4.2, when executing the assignment problem of set $CJ$. The pseudocode in Figure 1 shows that these conditions influence the runtime by possibly shortening the while loop of line 5.

### 5.3 Improved trip assignment in the feasibility phase

The idea of the feasibility phase remains the same. The objective is to find a feasible solution, by assigning the uncovered trips to additional train units. Instead of adding a big amount of train units of every type instantly, small amounts of train units will now be added iteratively. The numbers of train units used in each iteration are $d_{feas}^k$, initiated as the $\bar{d}^k$ values. For every trip that remains uncovered, the cheapest subset of train units $TU \in FSEL$ is considered. The amount of train units per type, that are part of this $TU$, are counted. This gives a number of train units to be added for every type $k$, referred to as $needed(k)$. When these numbers are obtained for every trip left uncovered, the highest of those numbers is chosen for every type, named $increase(k)$. In every iteration, the values of $d_{feas}^k$ are increased by $increase(k)$. With these values, do the trip assignment as in lines 7-26 in Figure 1 with lines 11,17 and 25 disregarded. With the deletion of these lines, the trip assignment is only tested for the cheapest $TU$ of every trip. If there are still trips left uncovered, go to the next iteration of the feasibility

phase. If all trips are covered, a feasible solution is found. If the total cost of the amount of train units in $d^k_{feas}$ is the cheapest solution yet, update the best solution cost and end the feasibility phase. The pseudocode for the trip assignment in the improved feasibility phase can be seen in Figure 3.

```
1   d^k_feas := d̄^k for all types k
2   uncovered := set of uncovered trips
3   while uncovered ≠ ∅ do
4       for j ∈ uncovered do
5           FSEL(j) := ordered set of possible subsets of train units for trip j
6           select the first subset in FSEL(j), called TU
7           for k ∈ TU do
8               count needed(k)
9               if needed(k) > increase(k) then
10                  increase(k) := needed(k)
11              end
12          end
13      end
14      d^k_feas := d^k_feas + increase(k) for all types k
15      for j ∈ uncovered do
16          covered(j) := FALSE
17          FSEL(j) := ordered set of possible subsets of train units for trip j
18          accept := TRUE
19          select the first subset in FSEL(j), called TU
20          for k ∈ TU do
21              solve SDVSP for type k
22              if SDVSP(k) > d^k_feas then
23                  accept := FALSE
24                  break
25              end
26          end
27          if accept then
28              covered(j) := TRUE
29              uncovered := uncovered\{ j}
30          end
31      end
32  end
33  get the solution cost for d^k_feas values found
```

Figure 3: The pseudocode for the improved feasibility phase

### 5.3.1 Implementation in the heuristic

The improved feasibility phase replaces the feasibility phase of the heuristic shown in Figure 2. The heuristic that is used when testing this new procedure, has the additional stopping criteria of Chapter 5.2 as well. This implementation is referred to as the *improved feasibility heuristic*. Besides this implementation, there are other ways to use this new procedure of trip assignment.

The feasibility phase is only executed after the constructive phase fails to find a solution equal to the PP-LB. By iterating over $n_c$ in the constructive phase, the heuristic tries to find such a solution equal to the PP-LB multiple times. If it is unlikely that such a solution will be reached, a lot of runtime is wasted in the constructive phase. In the contrary, the feasibility phase always finds a feasible solution. Furthermore, the improved feasibility phase goes towards a feasible solution with a cost-minimizing approach. When it is unlikely that the $\bar{d}^k$ values are enough to cover all the trips, it is therefore useful to let the heuristic focus more on the improved feasibility phase instead of the constructive phase. To do this, the trip assignment of the critical trips is only executed once (as if $n_c = 1$) in the constructive phase, before going to the feasibility phase. While doing this, updating the set $CJ$ has to be done differently. After the constructive phase, the trips left uncovered in $CJ$ get added to the new set $CJ$ in the same way as described in step 5.(a) of Chapter 4.2. The trips left uncovered in $UJ$ are added to the end of $CJ$. The

procedure as described here, is named the *short heuristic*.

Another implementation of the improved feasibility phase leads to the *new heuristic*. This heuristic starts with the assignment of trips in $CJ$ as in the *short heuristic* (as if $n_c = 1$). If there are trips left uncovered, the trip assignment of Figure 3 is executed on these trips. If this is the case, then the trip assignment of the trips in $UJ$ is done as usual (with lines 1-4 and 6-26 of Figure 1), but with the values of $d^k_{feas}$ attained in the trip assignment of uncovered trips in $CJ$. If the set $UJ$ has trips left uncovered after this trip assignment, the procedure of Figure 3 is done again with these uncovered trips. This implementation basically splits the feasibility phase into one feasibility phase for the trips in $CJ$ and one feasibility phase for the trips in $UJ$. The assignment of the trips in $UJ$ is only started if all trips in $CJ$ are covered, and the trip assignment of the trips in $UJ$ uses the number of train units needed for covering all the trips in $CJ$. For the next iteration of $n_{iter}$, the sets $CJ$ and $UJ$ are updated in the same way as in the *short heuristic*.

# 6 Results

A data set containing four different instances is given, each instance having only trips in the north of Italy. The instances consist of a list of 200, 318, 564 and 1010 trips, respectively. All the trips in a single instance have to be executed between 5:00 a.m. and 11:30 a.m., on the same day. In all instances, the assumption is that an arrived train unit needs five minutes of time before it can depart on another trip, so $t_j = 5 \quad \forall j \in J$. There are three different types of train units available in each instance: type $OC$, type $OH$ and type $OT$. The specifications of each train unit type are shown in Table 1. In every set-up of the heuristic, $n_c$ is set to 10 and $n_{iter}$ is set to 20. The programs are written in Java, making use of the CPLEX optimization package in the process.

Table 1: The specifications of all train unit types

| | **Type OC** | **Type OH** | **Type OT** |
|---|---|---|---|
| Yearly cost | €230,000.- | €190,000.- | €330,000.- |
| Seat capacity | 500 | 360 | 640 |
| Length | 100 | 75 | 125 |

The PP-LB is the same with every set-up of the heuristic, because the methodology of Chapter 4.1 is never changed. The results of the PP-LB's of the four instances can be seen in Table 2. The PP-LB cost of instance 4 more than doubles that of instance 3, while there are not twice as many trips in instance 4 as in instance 3. This means that a bigger percentage of the total trips is incompatible in instance 4 than in instance 3.

Table 2: The PP-LB cost and values of $\bar{d}^k$

| *PP-LB* | **Instance 1** | **Instance 2** | **Instance 3** | **Instance 4** |
|---|---|---|---|---|
| Cost | €49,950,000.- | €51,820,000.- | €60,570,000.- | €143,090,000.- |
| Train units | 215 | 224 | 259 | 609 |
| $\bar{d}^{OC}$ | 119 | 123 | 144 | 310 |
| $\bar{d}^{OH}$ | 65 | 70 | 75 | 192 |
| $\bar{d}^{OT}$ | 31 | 31 | 40 | 107 |

## 6.1 Comparing set-ups

The heuristic is executed 16 times in total: the four instances are executed with four different set-ups. The results of these 16 executions are shown in Table 3. The initial setup is the one discussed in Chapter 4, called *#3*. This name indicates that the subsets of $FSEL$ are of size 3 at most. The other 3 set-ups are the ones described in Chapter 5.1. The name *#5* indicates the set-up with subsets of $FSEL$ being of size 5 at most. The name *first* indicates the same set-up as *#3*, in which the second stage of the critical trip assignment is not executed. The

set-up similar to *first*, but in which the subsets in $FSEL$ are ordered by ascending cost instead of ascending capacity, is called *cost*.

Table 3: The results for different set-ups of the heuristic

| | **Instance 1** | | | | **Instance 2** | | | |
|---|---|---|---|---|---|---|---|---|
| | *#3* | *#5* | *first* | *cost* | *#3* | *#5* | *first* | *cost* |
| Cost (million euro's) | 53.41 | 53.52 | 53.54 | 51.91 | 63.00 | 62.76 | 63.31 | 61.39 |
| Gap PP-LB and cost (%) | 6.7 | 6.9 | 6.9 | 3.8 | 19.5 | 19.1 | 20.0 | 16.9 |
| Train units | 231 | 230 | 230 | 223 | 270 | 270 | 271 | 263 |
| OC | 119 | 123 | 120 | 123 | 128 | 122* | 131 | 142 |
| OH | 78 | 72 | 74 | 67 | 95 | 101 | 93 | 80 |
| OT | 34 | 35 | 36 | 33 | 47 | 47 | 47 | 41 |
| Runtime (seconds) | 229 | 277 | 112 | 57 | 603 | 827 | 365 | 170 |

| | **Instance 3** | | | | **Instance 4** | | | |
|---|---|---|---|---|---|---|---|---|
| | *#3* | *#5* | *first* | *cost* | *#3* | *#5* | *first* | *cost* |
| Cost (million euro's) | 80.34 | 80.37 | 78.86 | 77.30 | 171.58 | 170.01 | 166.22 | 157.33 |
| Gap PP-LB and cost (%) | 28.1 | 28.1 | 26.2 | 24.3 | 18.1 | 17.2 | 15.0 | 9.5 |
| Train units | 342 | 341 | 336 | 326 | 730 | 727 | 710 | 669 |
| OC | 167 | 169 | 169 | 174 | 318 | 307* | 335 | 339 |
| OH | 113 | 109 | 108 | 92 | 268 | 280 | 247 | 211 |
| OT | 62 | 63 | 59 | 60 | 144 | 140 | 128 | 119 |
| Runtime (seconds) | 1389 | 1991 | 833 | 484 | 4515 | 5326 | 2476 | 1413 |

* These values are lower than the corresponding $\bar{d}^k$ values of Table 2. This is possible, since the lower bound is on the cost, not on the $\bar{d}^k$ values.

When looking at the results of Table 3, a notable observation is that the gaps between the PP-LB and the upper bound cost are bad for instances 2, 3 and 4. Especially the gaps in instance 3 are really big, regardless of the set-up that is used. The gaps of instance 1 and 4 are the smallest, which shows that the quality of the solutions does not generally decrease as the size of the problem increases. However, the runtime increases significantly as the size of the problem increases. When the size of the problem approximately doubles, for example when comparing instance 3 to instance 4, the runtime of the heuristic more than doubles. This is the case for every set-up used.

When adjusting the heuristic, the goal is to analyze the percentage difference between the solutions found and the runtimes found. These results are obtained by comparing the results shown in Table 3. A summary of those percentage differences is made in Table 4. The three comparisons being made are between:

- *#3* and *#5*. The cost difference between the solutions of these set-ups is really small. Besides that, the solution cost of *#3* sometimes is smaller than the solution cost of *#5*. The runtime of *#3* is way faster for every instance. Choosing *#3* is therefore the better choice.

- *#3* and *first*. Similar observations as when comparing *#3* to *#5* are made. The *first* set-up is way faster than the *#3* set-up, and the differences in cost can be slightly better for either of the set-ups. Therefore, *first* is preferred.

- *first* and *cost*. When ordering the subsets in $FSEL$ by ascending cost, in general better results are obtained. This was to be expected, as the main goal is to minimize the cost. Surprisingly, the runtime is also shorter with the *cost* set-up. Because of the better solutions and shorter runtime, set-up *cost* is the best choice.

Table 4: The percentage difference between different set-ups, for cost and runtime

| | | **Difference (%)** | | |
|---|---|---|---|---|
| | | *#3 to #5* | *#3 to first* | *first to cost* |
| **Instance 1** | Cost | 0.21 | 0.24 | -3.00 |
| | Runtime | 21.30 | -50.93 | -49.06 |
| **Instance 2** | Cost | -0.38 | 0.49 | -3.03 |
| | Runtime | 37.16 | -39.53 | -53.39 |
| **Instance 3** | Cost | 0.04 | -1.84 | -1.98 |
| | Runtime | 43.39 | -39.99 | -41.87 |
| **Instance 4** | Cost | -0.92 | -3.12 | -5.35 |
| | Runtime | 17.97 | -45.15 | -42.93 |

## 6.2 Improvements

There are four different heuristics that have implemented improvements to the heuristic of Chapter 4. These heuristic all use the same set-up as in the *cost* heuristic, because that set-up is the best one found. The four improved heuristics are *stop*, *feas*, *short* and *new*. The name *stop* refers to the heuristic with the added stopping conditions, as described in Chapter 5.2. The names *feas*, *short* and *new* refer to the *improved feasibility heuristic*, *short heuristic* and *new heuristic* of Chapter 5.3, respectively.

With these four heuristics being executed for the same four instances, there are again 16 executions of a heuristic. The results are shown in Table 5. The gaps for the improved heuristic are still quite big, but smaller than the gaps in Table 3. The solutions found in the four different heuristics are all really similar. However only slightly better, the *short* heuristic gives the best solutions for each instance. The runtime of the *new* heuristic is lower than the runtimes of the other heuristics, for every instance. The runtimes of the *short* heuristic are really low for small instances, but drastically increase as the size of the instance increases. For the big instances, the runtime of this heuristic surpasses the runtime of the *stop* heuristic. The *feas* heuristic is the worst one in terms of runtime.

Table 5: The results for different improvements to the heuristic

| | **Instance 1** | | | | **Instance 2** | | | |
|---|---|---|---|---|---|---|---|---|
| | *stop* | *feas* | *short* | *new* | *stop* | *feas* | *short* | *new* |
| Cost (million euro's) | 51.83 | 51.83 | 51.69 | 51.73 | 61.39 | 61.39 | 60.42 | 61.39 |
| Gap PP-LB and cost (%) | 3.7 | 3.7 | 3.4 | 3.5 | 16.9 | 16.9 | 15.3 | 16.9 |
| Train units | 223 | 223 | 223 | 223 | 263 | 263 | 260 | 263 |
| OC | 121 | 121 | 121 | 122 | 142 | 142 | 139 | 142 |
| OH | 69 | 69 | 70 | 69 | 80 | 80 | 82 | 80 |
| OT | 33 | 33 | 32 | 32 | 41 | 41 | 39 | 41 |
| Runtime (seconds) | 28 | 29 | 12 | 12 | 75 | 95 | 56 | 41 |

| | **Instance 3** | | | | **Instance 4** | | | |
|---|---|---|---|---|---|---|---|---|
| | *stop* | *feas* | *short* | *new* | *stop* | *feas* | *short* | *new* |
| Cost (million euro's) | 78.03 | 78.03 | 77.17 | 78.36 | 157.33 | 157.33 | 156.04 | 156.67 |
| Gap PP-LB and cost (%) | 25.2 | 25.2 | 24.1 | 25.6 | 9.5 | 9.5 | 8.7 | 9.1 |
| Train units | 329 | 329 | 325 | 330 | 669 | 669 | 666 | 667 |
| OC | 171 | 171 | 172 | 171 | 339 | 339 | 342 | 339 |
| OH | 96 | 96 | 92 | 96 | 211 | 211 | 211 | 211 |
| OT | 62 | 62 | 61 | 63 | 119 | 119 | 130 | 117 |
| Runtime (seconds) | 281 | 501 | 405 | 209 | 850 | 1609 | 1236 | 695 |

When improving the heuristic, the goal is again to analyze the percentage difference between the solutions found and the runtimes found. Therefore, the results of Table 5 are compared to the results of the base case, which is the *cost* set-up. The percentage differences between the *cost* set-up and the four new heuristics, are shown in Table 6. The four comparisons are as follows:

– In the *stop* heuristic, the addition of extra stopping conditions is tested. The goal of lowering the cost does not seem to be achieved with this heuristic, as the cost difference is really small, or non-existent. However, the runtime drastically decreases while obtaining almost similar solution costs. In that regard, the extended heuristic is an improvement to the original heuristic.

– When changing the feasibility phase in the *feas* heuristic, the solution costs also barely change. The runtime again drastically decreases when compared to the *cost* set-up, for the smaller instances 1 and 2. However, the execution of this heuristic takes longer than the *cost* heuristic, when looking at the bigger instances 3 and 4. This heuristic therefore is not more useful when dealing with big data sets.

– With the implementation of the improved feasibility phase as in the *short* heuristic, the solution cost decreases for every instance. These differences are really small however, which raises the question whether or not these decreases are attained for all data sets. The runtime decreases immensely for the small instances, when compared to the *cost* set-up. These percentages decrease for the bigger instances, which raises the assumption that the *cost* heuristic will be faster for even bigger instances.

– The *new* heuristic is an extreme improvement to the *cost* heuristic in terms of runtime. Although the percentage decrease becomes smaller as the size of the instances increases, the decrease in runtime is still enormous for big instances. The solutions found are really similar to the solutions of the *cost* heuristic. They can be better or worse, but the difference is small enough to not prefer one over the other in terms of solution quality.

When comparing the four improved heuristics with each other, the *short* heuristic seems to give the best solutions. The solutions given by the four heuristics are almost the same though, so all four heuristics could be used for getting quality solutions. Because of this similarity, deciding which heuristic is the most effective is mostly decided by the runtime. Although all four heuristics except the *feas* heuristic are a big improvement in terms of runtime, the *new* heuristic excels. Looking at the percentages found in Table 6, one could argue that only the *stop* heuristic possibly becomes faster for instances of enormous size, as the runtime percentages do not seem to strongly decrease as the size of the instance increases.

Table 6: The percentage difference between the *cost* set-up and improved heuristics, for solution cost and runtime

| | | **Difference (%)** | | | |
|---|---|---|---|---|---|
| | | *cost to stop* | *cost to feas* | *cost to short* | *cost to new* |
| **Instance 1** | Cost | -0.15 | -0.15 | -0.42 | -0.35 |
| | Runtime | -50.36 | -49.27 | -79.01 | -79.01 |
| **Instance 2** | Cost | 0 | 0 | -1.58 | 0 |
| | Runtime | -55.98 | -44.11 | -67.05 | -75.88 |
| **Instance 3** | Cost | 0.94 | 0.94 | -0.17 | 1.37 |
| | Runtime | -41.95 | 3.41 | -16.40 | -56.86 |
| **Instance 4** | Cost | 0 | 0 | -0.82 | -0.42 |
| | Runtime | -39.86 | 13.86 | -12.54 | -50.82 |

# 7 Conclusion

In this thesis, the heuristic as described in Cacchiani et al. (2019) is tested, analyzed and improved. The proposed heuristic tries to solve the TUAP by iteratively decreasing the upper bound cost, until the same cost as in the PP-LB is achieved. If no such solution is reached after $n_{iter}$ iterations, the lowest found upper bound cost becomes the best solution to the TUAP. The heuristic is tested on four instances, initially with four different set-ups of the heuristic. Thereafter, the results of these four different set-ups are analyzed. Lastly, the results of four

different improved heuristics are analyzed. In total, Chapter 6 contains results of eight slightly different heuristics that are all based on the one suggested by Cacchiani et al. (2019). For the instances used in this thesis, a preferable set-up of the original heuristic is found. This thesis also gives three improvements to the heuristic, suggesting the use of one improvement in particular.

First, the results obtained in this thesis are somewhat similar to those of Cacchiani et al. (2019). The gap between the lower bound and upper bound is smaller in the instances of Cacchiani et al. (2019) than in this thesis. The used heuristic is initially the same, so the slightly different results are due to other data instances. The runtimes are comparable, but longer in this thesis than in Cacchiani et al. (2019). The main cause of this difference, is that the PP-LB is never reached in this thesis, which causes the heuristic to execute all iterations. In Cacchiani et al. (2019), the gap found is zero in many instances, which terminates the heuristic before all iterations are executed. So again, the difference is caused by the difference in the data sets.

When doubling the size of the instance, the runtime more than doubles. The solution cost shows no clear indication of change, when increasing the instance size. The gap between the lower bound and the upper bound becomes bigger sometimes, and smaller for other cases.

The heuristic spends most of its runtime on solving SDVSP models. Reducing the number of times this problem has to be solved, is therefore a good way to reduce the runtime. For the instances used in this thesis, this is achieved by having the following set-up: the size of subsets in $FSEL$ is 3 at most, only the first stage of the critical trip assignment is executed and the subsets in $FSEL$ are ordered by ascending cost. When making adjustments in this set-up to reduce the runtime, the quality of the upper bound solution barely changes. When the order of subsets in $FSEL$ changes from ascending capacity to ascending cost, the quality of the upper bound found even increases.

Besides the set-up, improvements to the heuristic strongly decrease the runtime as well. Namely, adding additional stopping conditions for better solutions also reduces the number of times a SDVSP has to be solved. The implementation of a new feasibility phase does not necessarily decrease the number of times a SDVSP has to be solved, but when implemented as in the *feas* and *short* heuristic, the runtimes of the small instances still decrease. For bigger instances however, the runtime of these heuristics increases faster than the runtime of original heuristic as in the *cost* set-up. The implementation of a new feasibility phase as in the *new* heuristic, causes an enormous decrease in runtime for every instance size. Although the *short* heuristic gives the best upper bound cost for every instance, there is no reason to believe that any of the improved heuristics gives better solutions. When comparing the original heuristic with the *cost* set-up to the improved heuristics, the differences in solution cost are really small. So the improved heuristics all give similar solutions, while decreasing the runtime enormously, with the *new* heuristic excelling.

# 8 Discussion

All the runtimes found in this thesis, that have the subsets in $FSEL$ ordered by ascending capacity (set-ups *#3*, *#5* and *first*), are longer than they should be. The implementation of the $FSEL$ set is done wrong during the execution of these heuristics. Namely, identical subsets, in which only the order of the train units differs, are not deleted. This causes the program to potentially run the same SDVSP multiple times, increasing the runtime needlessly. This is more problematic for the  cases than for the *#3* cases. Despite this mistake, the preference of the *cost* set-up over these three wrongly implemented set-ups can still be justified. When executing the *cost* heuristic with the same mistake implemented, this heuristic is still faster than the other three. Moreover, the solution quality is not affected by this mistake, so the *cost* set-up is also still better in terms of solution quality.

For further improvements on the heuristic, more finetuning could be done. The best solutions found are many times found with low numbers of $h$, in some cases the upper bound is not even

changed after the first iteration. One could question if $n_{iter}$ of 20 is too high for that matter. An example of the change in best upper bound throughout the iteration process, is shown in Figure 4, with the UB cost in million euros. The upper bound cost starts at infinity, which is shown by the green vertical line segment at $h = 1$. This is the iteration process of the *new* heuristic executed on instance 1. There are only better solutions in three iteration, and the upper bound remains the same for the last twelve iterations.

Also more research to find better solutions could be done. The improvements discussed in this thesis strongly reduce the runtime, but the solution quality remains almost the same.

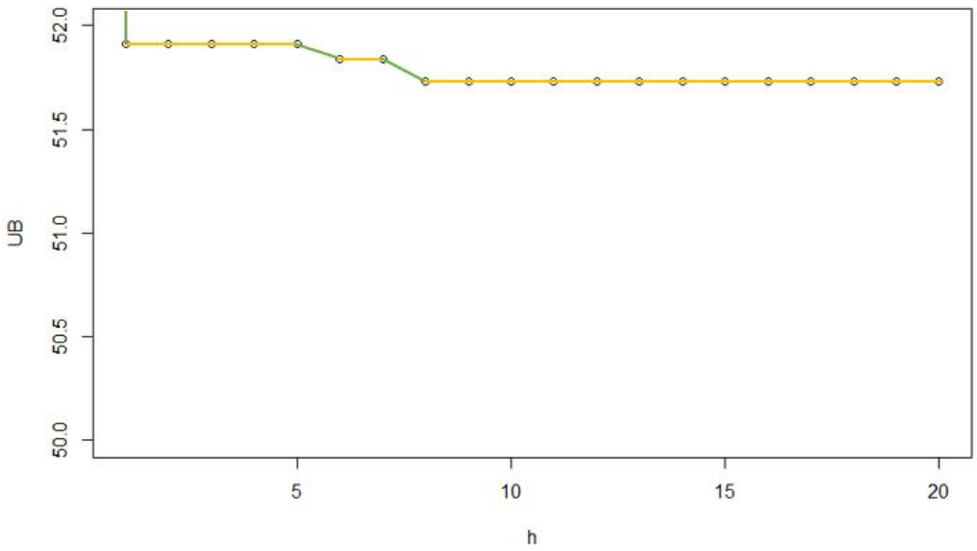


Figure 4: The best upper bound found in every iteration of $n_{iter}$, in which green line segments depict a change

## References


Abbink, E., Van den Berg, B., Kroon, L., and Salomon, M. (2004). Allocation of railway rolling stock for passenger trains. *Transportation Science*, 38(1):33–41.

Alfieri, A., Groot, R., Kroon, L., and Schrijver, A. (2006). Efficient circulation of railway rolling stock. *Transportation Science*, 40(3):378–391.

Borndörfer, R., Reuther, M., Schlechte, T., Waas, K., and Weider, S. (2015). Integrated optimization of rolling stock rotations for intercity railways. *Transportation Science*, 50(3):863–877.

Cacchiani, V., Caprara, A., and Toth, P. (2010). Solving a real-world train-unit assignment problem. *Mathematical programming*, 124(1-2):207–231.

Cacchiani, V., Caprara, A., and Toth, P. (2013). A lagrangian heuristic for a train-unit assignment problem. *Discrete Applied Mathematics*, 161(12):1707–1718.

Cacchiani, V., Caprara, A., and Toth, P. (2019). An effective peak period heuristic for railway rolling stock planning. *Transportation Science*, 53(3):746–762.

Cordeau, J.-F., Soumis, F., and Desrosiers, J. (2001). Simultaneous assignment of locomotives and cars to passenger trains. *Operations research*, 49(4):531–548.

Fioole, P.-J., Kroon, L., Maróti, G., and Schrijver, A. (2006). A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297.

Giacco, G. L., D'Ariano, A., and Pacciarelli, D. (2014). Rolling stock rostering optimization under maintenance constraints. *Journal of Intelligent Transportation Systems*, 18(1):95–105.

Haahr, J. and Lusby, R. M. (2017). Integrating rolling stock scheduling with train unit shunting. *European Journal of Operational Research*, 259(2):452–468.

Huisman, D., Kroon, L. G., Lentink, R. M., and Vromans, M. J. (2005). Operations research in passenger railway transportation. *Statistica Neerlandica*, 59(4):467–497.

Lin, Z. and Kwan, R. S. (2014). A two-phase approach for real-world train unit scheduling. *Public Transport*, 6(1-2):35–65.

Lin, Z. and Kwan, R. S. (2016). A branch-and-price approach for solving the train unit scheduling problem. *Transportation Research Part B: Methodological*, 94:97–120.

Lusby, R. M., Haahr, J. T., Larsen, J., and Pisinger, D. (2017). A branch-and-price algorithm for railway rolling stock rescheduling. *Transportation Research Part B: Methodological*, 99:228–250.

Maróti, G. and Kroon, L. (2005). Maintenance routing for train units: the transition model. *Transportation Science*, 39(4):518–525.

Peeters, M. and Kroon, L. (2008). Circulation of railway rolling stock: a branch-and-price approach. *Computers & operations research*, 35(2):538–556.

Schrijver, A. (1993). Minimum circulation of railway stock. *Cwi Quarterly*, 6(3):205–217.

Wagenaar, J. C., Kroon, L. G., and Schmidt, M. (2017). Maintenance appointments in railway rolling stock rescheduling. *Transportation Science*, 51(4):1138–1160.

## Appendix

The code used in this thesis is listed below. The code is written for the instances used here, keeping in mind that there are only three types of train units. The code is not written for the use of other types of train units in general, only for the use of these three types. The zip file contains the following files as code:

- DirectedGraph.java: this file is a simple class that can be used to model directed graphs, where arbitrary types of data are associated with the nodes and arcs of the graph.
- DirectedGraphArc.java: class that models arcs in the directed arcs.
- MainCacchiani.java: the main program that runs the heuristic as described in Cacchiani et al (2019).
- MainFeas.java: the main program that runs the heuristic with the additional feasibility phase, named *feas*.
- MainNew.java: the main program that runs the heuristic with the additional feasibility phase implemented as in the *new* heuristic.
- MainShort.java: the main program that runs the heuristic with the additional feasibility phase implemented as in the *short* heuristic.
- MainStop.java: the main program that runs the heuristic with the additional stopping criteria, named *stop*.
- ModelILP.java: the CPLEX model that is used to solve the model (5)-(8).
- ModelMFPD.java: the CPLEX model that is used to solve the model (1)-(4).
- ModelSVSP.java: the CPLEX model that is used to solve the Single Depot Vehicle Scheduling Problem (SDVSP).
- TrainUnit.java: the train unit object, which consists of the type, costs per year, amount of seats and length in meters.
- Trip.java: the 'Trip' object, which consists of the departure time and location, arrival time and location, the demand on the trip and the maximum length of the train on this trip.

Furthermore, there are text files for the four instances, that are read by the main programs.