Erasmus University Rotterdam

Erasmus School of Economics

Multiple Solution Approaches for the Assignment of Train Units

Train Unit Assignment Problem

Bachelor Thesis
BSc Econometrie en Operationele Research

Author:
L. Leijten

Supervisor:
R.N. Van Lieshout

Second reader:
T.A.B. Dollevoet

Abstract

In this thesis, we discuss different solution approaches for the train unit assignment problem (TUAP). The TUAP covers the aspect of distributing train units over a known timetable. The goal is to minimise the costs of the used train units while the capacity of the train units must exceed the estimated demand. In this paper, we first discuss a mixed-integer linear program which could solve the TUAP exactly. Afterwards, we discuss the peak period heuristic as introduced by Cacchiani et al. (2019). This heuristic defines the peak period of a timetable as a set of so-called incompatible trips and iteratively tries to search for the least cost solution. Then we come up with a heuristic which is similar to the exact formulation of the set cover problem. However, instead of considering all possible routes, we only use a smaller subset of 'significant' routes, such that the problem is also solvable for larger instances within sufficient time. Finally, we come up with a local search algorithm which can improve the outcome of any given feasible solution. This local search technique checks for a randomly chosen trip, whether there is an alternative composition of train units, which decreases the total costs until no further improvements can be found. We compare these heuristics with the exact formulation and observe that the performance of the peak period heuristic is relatively poor. The results of the set cover heuristic and local search are decent. However, the exact formulation still solves the instances optimally within an acceptable time.

4 July 2019

Contents

1 Introduction

A frequent train traveller faces various problems during their journey. From undesirable departure times to excessive delays, a passenger can find their experienced train trips far from perfect. However, the constructing of a train schedule is a time-consuming process, when at the same time taking the interests of all involved parties into account. Yearly, almost 30 billion passengers travel over approximately 800,000 kilometres of train lines (UIC, 2018). Therefore, satisfying all the passengers' interests would be like squaring the circle for railway companies.

What will railway companies do to satisfy the passengers as much as possible and what factors or preferences of passengers will they take into account when determining the train schedule? As stated in Caprara (2015), railway planning can be divided into four smaller subproblems: train timetabling, train platforming, train-unit assignment and crew assignment problem. The first step of a complex process is the determination of arrival and departure times at each station, labelled as train timetabling problem. This process has already been proven to be NP-hard (Caprara et al., 2002). Next, the entrance and exit routes of trains and the allocation of platforms must be determined. This so-called train platforming problem is closely related to the train unit assignment problem (TUAP), where various train units are combined to satisfy the demand for a specific train trip. Finally, crew assignment problem decides the working plan for each employee. Such a complicated process needs to be executed as perfect as possible, while an incorrect implementation of one of the stages could lead to high costs for the railway company or dissatisfaction by the passengers.

In this thesis, the main focus is on the train-unit assignment problem: the determination of train units over all the trips. The arrival and departure times and stations of all trips, the demand and the set of train units are assumed to be given. Train units are used to satisfy the given demand for every trip. The goal is to minimise the total costs of the assignment of the used train units. However, the context introduced by Cacchiani et al. (2019) is "that of a competitive bid process whereby a train operator competes to win a contract for providing rolling stock circulation in a regional railway network." By estimating the yearly costs for the usage of train units, a train operator can give their best valuation of the proposed contract. An overestimation of the costs will probably not lead to winning the bid, while an (infeasible) underestimation will probably lead to winning the bid, but eventually to plausible losses. However, Cacchiani et al. (2019) prove that the problem is NP-hard.

This research aims is to analyse the implemented method used in Cacchiani et al. (2019) and replicate these results with comparable data sets. The authors propose a heuristic based on the peak period, such that it can approximate the yearly costs for using train units. The paper of the authors includes a real-world application for a train operator in the North of Italy, and their results show decent performance and computational time.

Before we explain the peak period heuristic, we introduce an integer linear program which is able to solve instances exactly. Furthermore, we introduce two new proposed heuristics to solve the TUAP. Firstly, we propose a set cover heuristic, which is similar to the well-known set cover formulation. The difference is that we do not consider every possible route as usually, but we use a smaller subset of 'significant' routes. Furthermore, we will add a weight factor to each route to emphasise its significance. Lastly, we introduce a local search technique, which is meant to improve any given feasible solution. It does this for every trip by finding a cheaper composition of train units for that trip.

Whereas Cacchiani et al. (2019) conclude that the peak period heuristic performs well within decent computational time, our results show that the algorithm is slow and the outcomes are far from the optimal solution. The integer linear program can obtain the optimal solution, although it takes more time for bigger instances. The set cover heuristic is a speedy algorithm, which has, especially when including weight factors, excellent results compared to the peak period heuristic. The local search approach seems to improve any given solution significantly, and in combination with the set cover heuristic and using weight factors, we obtain the best results.

This thesis is organised as follows. The related literature is reviewed in Chapter 2. Afterwards, we start to explain the train unit assignment problem in Chapter 3. Then we propose an integer linear program which could solve the TUAP exactly in Chapter 4, followed by the proposed solution approach in Cacchiani et al. (2019) in Chapter 5. We clearly explain the defined lower bound and the proposed heuristic to tackle the TUAP in respectively Chapters 5.1 and 5.2. Afterwards, we introduce our proposed set cover heuristic in Chapter 6. Chapter 7 describes the functioning of the local search algorithm. With the explanation of all our used solution approaches, we present our results in Chapter 8. Lastly, we end this thesis by summarising and concluding our mentioned results in Chapter 9.

2 Literature Review

This chapter reviews some previous findings of related research. Firstly, we introduce some interesting articles about general railway planning. Secondly, we inspect different points of view about the train unit assignment problem, which is just a small subproblem of railway planning.

2.1 Railway Planning

As already mentioned, Caprara (2015) states that the railway planning exists of four smaller categories: train timetabling, train platforming, train-unit assignment and the crew assignment problem. However, some argue that railway planning consists of seven subproblems, as stated in Goossens et al. (2006). They add two stages before and one stage after the already introduced railway planning of Caprara (2015). The long-term estimates of the demand and the line planning process are both stages ahead of the already introduced four categories. Both these stages are strategic decisions, which have a time horizon of five till fifteen years (Ghoseiri et al., 2004). The line planning consists for the main part the determining of currently non-existing train rails to build or to demolish currently existing train rails. Nowadays, this process is expensive and therefore, negligibly profitable. The seventh and last stage is shunting and maintenance planning, which does not have a major influence on the final result of railway planning.

Cordeau et al. (1998) show an overview of optimisation formulations for different train problems. Although their article does not contain the most recent up-to-date developments, it summarises different solution approaches for different transportation problems. Also, Fang et al. (2015) show a clear overview of different procedures, although this article is more focused on the rescheduling in railway networks. There are over 100 references to other papers and it reveals that most of the referenced papers use a mixed-integer linear programming formulation.

When a train schedule has been made, the duty of a railway company does not stop. The reality is that possible delays and maintenance cause other problems the company has to deal with. Another article of Cacchiani and Toth (2012) describes the so-called nominal and robust train schedules. A nominal train schedule does only incorporate the given constraints, while robust train schedules also take possible disruptions into account. For this reason, nominal train schedules approach the optimal scenario, while robust train schedules reach reality.

2.2 Train Unit Assignment Problem

The TUAP is a strongly NP-hard problem, as it has been proven in Cacchiani et al. (2010), even in the case when all the trips overlap in time. In their paper, they introduce an integer linear program, where one decision variable corresponds to a feasible trip schedule for one train unit type. Based on this formulation, the authors propose a so-called diving heuristic. However, due to its time-consuming process, they propose another approach in Cacchiani et al. (2013). The definition of the variable of this integer linear program changed to one pair of trips assigned in a sequence to one train unit. The Lagrangian relaxation of this formulation is less time consuming than the diving heuristic formulated in Cacchiani et al. (2010). Cacchiani et al. (2019) compare these two methods with another introduced peak period heuristic. As noted, we will discuss this peak period algorithm in more detail in Chapter 5.

The train unit assignment problem is similar to the rolling stock circulation problem as discussed in Fioole et al. (2006). The addition of the authors to this problem is the consideration of the order of train units during a trip. By doing this, it performs a more realistic combining and splitting of trains during a sequence of trips. A mixed-integer formulation is solved to compute its results. This method is compared with a column and row generation approach described in Haahr et al. (2016).

Lin and Kwan (2014) describe a complicated process, as it also includes the shunting process at train stations. It is a two-phase approach where it starts with a similar solution concept as in Cacchiani et al. (2010) to solve the train unit scheduling. On top of that, their article comes up with a mixed-integer linear programming formulation to solve the railway shunting problem. It furthermore considers the rescheduling of train units as a result of possible delays or maintenance. Also, Cadarso and Marín (2011) address shunting operations in combination with the railway rolling stock circulation. They suggest a mixed-integer linear program where it penalises shunting operations like deadheading and rotation maneuvres. Based on this formulation, Cadarso and Marín (2014) present another robust formulation to tackle this problem. It uses a Benders decomposition in combination with their proposed heuristic.

3 Problem Description

In this section, we give a general description of the TUAP and introduce the constraints and parameters. For the sake of simplicity, we adopt as much as possible to the same notation as introduced in Cacchiani et al. (2019).

Given is a set of timetabled train trips $T = \{1, ..., n\}$. Each trip $t \in T$ has a departure time s_t and an arrival time f_t, and both corresponding to their respective train station. We introduce a set of nodes V in which each station corresponds to one of the nodes $i \in V$. Furthermore, each trip has a given positive deterministic demand r_t of passenger seats, and every train track has a maximum train length l_t. Also, a set of different train unit types $K = \{1, ..., p\}$ is presented. Each type $k \in K$ has a yearly cost c^k, a capacity R^k of passenger seats and a length L^k in meters.

Some specific operational requirements are needed to direct to possible solutions. First of all, the demand for passenger seats r_t must be satisfied for every trip. To do this, we connect different train units such that their total capacity exceeds the estimated demand. Secondly, for each trip, the length of a train unit combination $\sum_{k \in K} L^k$ cannot be higher than the given maximum train length l_t. Lastly, two trips i and j can be performed in sequence by a train unit if and only if $s_j \geq f_i + t_{ij}$, where t_{ij} is the transfer time between the arrival location of trip i and the departure location of trip j. Our goal is to minimise the total costs required to cover all the trips. In this paper, there is no restriction to the maximum amount of train units and neither to specific train unit combinations which can be combined.

Concluding, we can define the TUAP as the minimisation of costs of used train units, such that the capacity of the used train unit combination exceeds the estimated demand for every trip. Furthermore, the length of the train unit composition cannot exceed the maximum train length. Lastly, train units can be used for multiple trips whenever these trips can be sequenced. This applies if and only if $s_j \geq f_i + t_{ij}$.

The opportunity of deadheading will be excluded from this paper. However, train unit combinations can be expanded and broken into smaller combinations. While deadheading is not possible, t_{ij} can be interpreted as the time it takes to (un)load passengers and to connect or disconnect train units from a combination. It also implies that trip j can only be performed after trip i if the departure station of i is the same as the arrival station of trip i.

4 Integer Linear Program

Before we move on to the peak period heuristic described in Chapter 5, we start by introducing an integer linear program which can solve our proposed problem exactly. We, therefore, introduce a vehicle scheduling formulation in which we define the decision variable based on the number of train units used at an arbitrary arc. This model is based on the path formulation presented in Cacchiani et al. (2010). We represent every trip in an instance as a node, and we include an arc between two nodes i, j, whenever these two trips can be sequenced ($s_j \geq f_i + t_{ij}$) after each other.

Therefore, we introduce an undirected graph $G = (V, A)$, where the set V consists of all trips $t \in T$. Two nodes $i, j \in V$ are connected as an arc whenever $s_j \geq f_i + t_{ij}$, such that it meets the sequencing constraint. An arc is defined as $(i, j) \in A$, where A is the set of all arcs. By introducing an artificial source θ and an artificial sink τ, we obtain the graph $G' = (\bar{V}, \bar{A})$, where $\bar{V} = V \cup \{\theta\} \cup \{\tau\}$ and $\bar{A}$ is the set A combined with introduced arcs between the source, the sink and other nodes. Every node from the set V can be connected to θ as well as to τ. The flow 'starts' at the source and 'ends' at the sink. Besides that, we introduce the decision variable x_{ij}^k, which is equal to the number of train units of type k, which perform trip j in sequence of trip i. Note that the variable x_{ij}^k is only defined for arcs between i and j, which are defined as described above. Putting this all together, we can create the following integer linear program:

$$\min \quad \sum_{k \in K} \sum_{j \in V} c^k x_{\theta j}^k, \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in V \cup \{\tau\}} x_{ij}^k = \sum_{i \in V \cup \{\theta\}} x_{ji}^k, \qquad k \in K,\ j \in V, \tag{2}$$

$$\sum_{k \in K} \sum_{i \in V \cup \{\theta\}} R^k x_{ij}^k \geq r_j, \qquad j \in V, \tag{3}$$

$$\sum_{k \in K} \sum_{i \in V \cup \{\theta\}} L^k x_{ij}^k \leq l_j, \qquad j \in V, \tag{4}$$

$$x_{ij}^k \in \mathbb{Z}^{\geq}, \qquad k \in K,\ (i, j) \in A. \tag{5}$$

The goal (1) is to minimise the costs of the used train units. Since every train units 'starts' at the source θ, it is sufficient to minimise the costs of the train units coming out of the source. Constraint (2) makes sure the same amount of train units arrive and depart from a train station. Restrictions (3) and (4) require that the demand and the maximum length constraints are satisfied. Lastly, restriction (5) imposes x_{ij}^k to be a positive integer.

5 Peak Period Heuristic

The peak period heuristic is introduced in Cacchiani et al. (2019). Its goal was to construct an algorithm to tackle the TUAP into rapidly, qualitative results, especially compared to the methods described in Cacchiani et al. (2010) and Cacchiani et al. (2013). Firstly, we present the description of the lower bound in Chapter 5.1. Secondly, Chapter 5.2 explains the heuristic itself including the upper bound.

5.1 Lower Bound

The computation of the lower bound is divided into two stages. Firstly, we determine the set of so-called incompatible trips, which is the largest subset of trips which cannot be sequenced with other trips in this set. Secondly, we use this subset to incorporate the maximum train length and the usage of train units, such that it creates a lower bound. The outcome is a decomposition of train units over the set of incompatible trips. This result can be defined as a lower bound because we only consider a subset of all the given trips.

5.1.1 First stage

In this first stage, we want to determine the largest subset of trips which cannot be sequenced with each other, the so-called set of incompatible trips S. Therefore, we introduce the same undirected graph $G' = (\bar{V}, \bar{A})$ as introduced in Chapter 4. With that graph, we want to come up with a model which is comparable to the minimum flow problem with demands. We therefore introduce the adjacency matrix of graph G with a_{ij}, where $a_{ij} = 1$ if trip j can be assigned after trip i, and $a_{ij} = 0$ otherwise. Furthermore, each node $j \in V$ has weight r_j, which must be strictly satisfied by the flow. Finally, we introduce the continuous, positive decision variable x_{ij}, which represents the flow on the edge between node i and j and where $i \neq j$. With the introduction of these variables, we can come up with the following mathematical formulation:

$$\min \quad \sum_{j \in V} x_{\theta j}, \tag{6}$$

$$\text{s.t.} \quad \sum_{i \in V \cup \{\tau\}} x_{ij} = \sum_{i \in V \cup \{\theta\}} x_{ji}, \qquad j \in V, \tag{7}$$

$$\sum_{i \in V \cup \{\theta\}} x_{ij} \geq r_j, \qquad j \in V, \tag{8}$$

$$0 \leq x_{ij} \leq M a_{ij}, \qquad i \in V \cup \{\theta\},\ j \in V \cup \{\tau\}. \tag{9}$$

The objective function (6) minimises the total flow coming out of the source θ. Constraint (7) makes sure that the flow at every node will not be lost or added. Restriction (8) imposes that the demand at every node must be satisfied. Constraint (9) requires that the sequencing restriction must hold and therefore, the parameter M must be at least as high as the highest possible flow. This restriction is in combination with the fact that the flow must be non-negative. Since the imposed demand r_j is integer, the decision variable x_{ij} would also be integer (Cacchiani et al., 2019).

This above-described model minimises the flow coming out from the source θ, such that it provokes flow between nodes within the set V. If we consider the dual formulation of the model (6)-(9), we can obtain the set of incompatible trips S from the dual variables of Constraint (8). Whenever a dual variable of this constraint is not equal to zero, it belongs to the set of incompatible trips.

5.1.2 Second stage

Using the solution of Chapter 5.1.1, a set of so-called incompatible trips S can be defined. We can construct a lower bound when we solve the optimal assignment of trips for this subset of incompatible trips S. We do not have to consider sequencing, because trips within the set S cannot be sequenced with each other. Therefore, we introduce the following decision variable w_j^k, where w_j^k serves as the number of train units of type $k \in K$ assigned to trip $j \in S$. The following model will lead to the lower bound:

$$\min \quad \sum_{k \in K} \sum_{j \in S} c^k w_j^k, \tag{10}$$

$$\text{s.t.} \quad \sum_{k \in K} R^k w_j^k \geq r_j, \qquad j \in S, \tag{11}$$

$$\sum_{k \in K} L^k w_j^k \leq l_j, \qquad j \in S, \tag{12}$$

$$w_j^k \in \mathbb{Z}^{\geq}, \qquad k \in K,\ j \in S. \tag{13}$$

The goal (10) is to minimise the costs of the used train units. The constraint (11) satisfies the given required demand. Restriction (12) makes sure a combination of train units will not exceed the given maximum train length of that track l_j. The last constraint (13) verifies that w_j^k is a non-negative integer variable. The objective function of this mathematical model presents the lower bound of the TUAP. The used decomposition of train units can be represented as $\bar{d}^k$, where $d^k = \sum_{j \in S} w_j^k$ for every $k \in K$.

5.2 Heuristic Algorithm

Our goal is to compute an upper bound which coincides exactly with the decomposition $\bar{d}^k$ as described in Chapter 5.1.2. When this happens, an optimal solution has been found, because the lower bound corresponds to the upper bound. The peak period heuristic is executed iteratively. Every iteration, two phases need to be performed. Firstly, the *constructive phase* seeks to solve the minimisation of costs with respect to the given constraints and in the meanwhile only using the $\bar{d}^k$ train units for every type $k \in K$. An optimal solution has been found whenever all trips are covered regarding the given decomposition $\bar{d}^k$. Secondly, if there are still some trips left uncovered, the *feasibility phase* will try to obtain a feasible solution. While in the constructive phase, the number of train units cannot exceed the given decomposition $\bar{d}^k$, in the feasibility phase this is nonetheless possible. Therefore, a feasible, but possibly not an optimal, solution is guaranteed.

A general overview of this heuristic can be found in Algorithm 1. Firstly, the initialisation process starts and is covered in Chapter 5.2.1. Afterwards, the explanation of the constructive phase follows, where the assignment of the critical and uncritical trips will be explained. Information regarding the assignment of these different trips can be respectively found in Chapters 5.2.2.1 and 5.2.2.3. Chapter 5.2.2.2 covers the assignment problem which is necessary for the assignment of trips. As can be seen in Step 1, if at the end of the constructive phase, a feasible solution has been found, an iterative process is not needed. In that case, we return the best found optimal solution. In the other case, we move on to the feasibility phase. In this stage, as described in Chapter 5.2.3, we assign the trips which were not covered in the constructive phase. Step 3 is needed whenever a solution has not been found in the first iteration, which means that some updates are needed before we repeat the constructive phase. Although more iterations will give a better solution, we decided to set a maximum of n_{iter} iterations principally due to time constraints.

Algorithm 1: General Overview of Peak Period Heuristic

Step 0: *Initialisation*;
- $CJ \leftarrow S$;
- $UJ \leftarrow J \setminus CJ$;
- $BestSolution \leftarrow \infty$;

Step 1: *Constructive Phase*;
- Assignment of critical trips CJ;
- Assignment of uncritical trips UJ;
- **If** feasible solution has been found, STOP: go to Step 4; **Else**, continue to Step 2;

Step 2: *Feasibility Phase*;
- Assignment of uncovered trips;
- **If** feasible solution found $C < BestSolution$, **then** update $BestSolution \leftarrow C$;
- **If** $h \geq n_{iter}$, STOP go to Step 4; **Else**, continue to Step 3;

Step 3: *Update*;
- $h \leftarrow h + 1$;
- $CJ \leftarrow CJ \cup \{\text{uncovered trips}\}$;
- **Return** to Step 1;

Step 4: *Termination*;
- Return the best (optimal) solution found;

5.2.1 Initialisation

Before this upper bound heuristic can start, the computation of the lower bound as described in Chapter 5.1 is necessary. We want to separate the given trips $j \in J$ into two sets: the set of critical trips CJ and the set of uncritical trips $UJ = J \setminus CJ$, wherein each set the trips are ordered according to their departure times. The set CJ will be initialised as $CJ = S$, where the set S is the set of incompatible trips, obtained during the computation of the lower bound. Furthermore, the decomposition $\bar{d}^k$ is assumed to be given.

5.2.2 Constructive Phase

As already discussed, the goal of the constructive phase is to converge to the computed decomposition $\bar{d}^k$ of train units, although, because of the composed restrictions, there is no guarantee. If no solution has been found, the feasibility phase will be executed. We have two different solution approaches for the critical CJ and uncritical trips UJ. The constructive phase is a complex stage, so it has been divided into multiple paragraphs in which we describe the process as clearly as possible.

5.2.2.1 Assignment of Critical Trips

The assignment of critical trips is a complex process and we will explain it as clearly as possibly with the help of Algorithm 2. The input of the heuristic is the set of critical CJ and uncovered trips $UNCOV$. The algorithm will be repeated for n_c iterations or in case there are no uncovered critical trips left (*line 1-4*). For every iteration, we consider all the critical trips to be uncovered. For every critical trip $j \in CJ$, we search for all the subsets of train units which satisfy the given demand and the maximum length restriction. The set of all subsets of train units is named $FSEL(j)$ (*line 6-9*). Then, we consider every subset $TU \in FSEL(j)$ where we start with combinations which are as close, but also higher, than the demand r_j (*line 10-11*). For each train unit k, given a combination $TU \in FSEL(j)$, we solve the assignment problem $AP(k)$, as Chapter 5.2.2.2 discusses. If for one of the train units k, the costs of $AP(k)$ exceeds the given decomposition $\bar{d}^k$, the combination TU will not be used, and we return to *line 10*. If for every $k \in TU$, the costs of $AP(k)$ do not exceed $\bar{d}^k$, we consider this composition and move on

to *line 20* (*line 12-19)*. If we advance to *line 20*, we found a train unit composition TU for trip j. For this reason, we exclude this trip from the set of uncovered trips, and we go back to line *line 6*, where we consider the next trip (*line 20-25*).

If we have considered every trip, we end up at *lines 26 - 28*. Here, we increase the number of iterations by one. At the same time, we change the trip order in the set of critical trips CJ, where we consider the following order for every $u \in CJ$ respectively:

1. Every uncovered trip where $u \in S$, in random order;
2. Every covered trip where $u \in S$, in random order;
3. Every uncovered trip where $u \notin S$, in chronological order;
4. Every covered trip where $u \notin S$, in chronological order;

If we have still some trips left uncovered or $i < n_c$, we start another assignment of critical trips as described above, starting at *line 4*. If we end up without a solution after n_c iterations, we set the iteration counter to zero and start another n_c iterations. However, during the second event of iterations, we move the uncovered trips before the covered trips, no matter whether a trip belongs to the set S. If, even after this change, there are still critical trips left uncovered, we start the procedure for uncritical trips and afterwards the feasibility phase.

Algorithm 2: Assignment of Critical Trips

```
   Result: The assignment of the critical trips with eventually some uncovered trips left over
 1 CJ: set of critical trips;
 2 UNCOV: set of the uncovered trips;
 3 i ← 0;
 4 while i < n_c and UNCOV ≠ ∅ do
 5 |   UNCOV ← CJ;
 6 |   for j ∈ CJ do
 7 |   |   cover(j) ← FALSE;
 8 |   |   FSEL(j) ← set of all the subsets of train units with sum of the capacities ≥ r_j and ≤ l_j;
 9 |   |   while FSEL(j) ≠ ∅ and cover(j) =FALSE do
10 |   |   |   accept ← TRUE;
11 |   |   |   select a subset TU ∈ FSEL(j) of train units for j;
12 |   |   |   for k ∈ TU do
13 |   |   |   |   solve AP(k);
14 |   |   |   |   if cost(AP(k)) > d̄^k then
15 |   |   |   |   |   accept ← FALSE;
16 |   |   |   |   |   FSEL(j) ← FSEL(j) \ {TU};
17 |   |   |   |   |   break;
18 |   |   |   |   end
19 |   |   |   end
20 |   |   |   if accept then
21 |   |   |   |   cover(j) ← TRUE;
22 |   |   |   |   UNCOV ← UNCOV \ {j};
23 |   |   |   end
24 |   |   end
25 |   end
26 |   i ← i + 1;
27 |   change the trip order in CJ;
28 end
```

5.2.2.2 Assignment Problem Solution

The goal of the assignment problem is to determine the optimal sequence of a train unit type k. Therefore, we introduce the following graph $G_{AP} = (V', A)$, where V' consists of nodes for each trip where train unit type k has been used. If a trip is assigned to multiple train units of type k, then the same number of identical nodes are included in the graph. For example: if a trip is assigned to two train units of type k, there are two identical nodes for this trip. Arcs A between the nodes have different costs g_{ij} assigned to them, where $g_{ij} = 0$ if trip j can be performed in sequence after trip i by the same train unit and 1 otherwise. Furthermore, we introduce the decision variable y_{ij}, which equals 1 if trip j is performed in sequence after trip i by the same unit and 0 otherwise. With this information, we achieve the following AP-model:

$$\min \quad \sum_{(i,j)\in A} g_{ij} y_{ij}, \tag{14}$$

$$\text{s.t.} \quad \sum_{i\in V'} y_{ij} = 1, \qquad j \in V', \tag{15}$$

$$\sum_{j\in V'} y_{ij} = 1, \qquad i \in V', \tag{16}$$

$$y_{ij} \geq 0, \qquad (i,j) \in A. \tag{17}$$

Our goal (14) is to minimise the number of train units to execute all the given trips. Constraints (15) and (16) make sure each node has been visited and departed exactly once. Because the constraint matrix is totally unimodular, it is technically unnecessary to define y_{ij} to be binary (Cacchiani et al., 2019). This formulation is comparable to the assignment problem, although it can also be seen as a vehicle scheduling problem. In this case, we add a source and a sink, in a similar way as described in Chapter 5.1.1. We then minimise the number of train units coming out of the source.

5.2.2.3 Assignment of Uncritical Trips

The set of uncritical trips UJ is also considered in chronological order. We follow a nearly identical procedure as with the critical trips. The only difference is the absence of *line 26 - 28* of Algorithm 2, because iterations will not be necessary to derive a feasible solution for the uncritical trips.

5.2.3 Feasibility Phase

In case a feasible, and so an optimal, solution has not been found in the constructive phase, the feasibility phase will follow. In this stage, the uncovered trips are assigned to the remaining train units, such that the new composition will exceed the composition $\bar{d}^k$. The used procedure will be the same as described in Chapter 5.2.2.3. Since there is no restriction to the maximum of train units used, a feasible solution is guaranteed.

6 Set Covering Heuristic

In this section, we introduce a heuristic based on the set covering formulation. With a set covering model, all possible combinations between trips are considered with the goal to find the minimum number of combinations or minimum amount of costs to cover all trips. Since determining all possible routes and solving those is nearly impossible for bigger instances, we come up with a smaller subset of possible combinations which are acknowledged as promising. We start by presenting the set cover formulation, where we will use a subset of all possible trips. Afterwards, we continue with determining this subset of important trips and eventually determining their respective weights. Lastly, we determine the cheapest way to sequence different combinations of trips with each other.

6.1 Set Cover Formulation

We start by introducing the set cover formulation which we will use. We define the set F as the set of all possible combinations of trips, where we mention a possible combination $f \in F$ as a route. However, since we use a subset of all possible routes, we introduce the set $\bar{F} \subseteq F$, which is the set of significant routes. Our main goal is to minimise the costs of the used train units, and we thereby prefer more significant, more plausible routes. The weight factor C_f is the penalisation of choosing route f in the optimal solution. In Chapter 6.2, we describe more precisely which routes are included in $\bar{F}$ and how the weight factor is computed. We introduce the parameter b_{jf}, which is equal to 1 when trip j is part of route f and otherwise $b_{jf} = 0$. As discussed, c^k is the cost of using a train unit of type k. Having that in mind, we can introduce the decision variable x_f^k. This variable is equal to the number of times route f needs to be performed for a train unit of type k. Finally, we come up with the following set covering formulation:

$$\min \quad \sum_{k \in K} \sum_{f \in \bar{F}} (c^k + C_f)\, x_f^k, \tag{18}$$

$$\text{s.t.} \quad \sum_{k \in K} R^k \sum_{f \in \bar{F}} b_{jf} x_f^k \geq r_j, \qquad j \in V, \tag{19}$$

$$\sum_{k \in K} L^k \sum_{f \in \bar{F}} b_{jf} x_f^k \leq l_j, \qquad j \in V, \tag{20}$$

$$x_f^k \in \mathbb{Z}^{\geq}, \qquad f \in \bar{F},\ k \in K. \tag{21}$$

The objective function (18) minimises the total cost of the used train units while taking the penalties C_f into account. Restrictions (19) and (20) make sure that every trip j does not exceed its maximum given train length and its train capacity is at least larger than the given demand r_j. Lastly, the last constraint (21) imposes the decision variable x_f^k to be a positive integer, because a train unit type k can be chosen multiple times for trip f.

6.2 Determination of Significant Routes

The determination of the set of significant trips and their weights is the most intuitive step. If we include all possible routes F and solve this with the set covering formulation, we obtain the optimal solution. However, the set F may become too big when the given data sets become too big. Therefore, we introduce the subset $\bar{F} \subseteq F$. $\bar{F}$ is the set of routes considered 'significant' enough to be considered for the set cover problem. To incorporate the significance of a specific route $f \in \bar{F}$ even further, we can use a weight factor C_f to determine the significance or more plausible chance of picking a route f in comparison to other routes. This weight factor C_f could also be seen as a penalty factor for less important trips.

Although there is no perfect prescription for every instance, there are certain factors to take into account. First of all, to guarantee a feasible solution, we add a route with only trip j in it to the set $\bar{F}$ and do this for every $j \in V$. Besides that, our goal is to find routes of trips which are similar in characteristics and will probably have the same or similar composition of train units. To give a concrete example, let us consider the train stations map as a star graph as represented in Figure 1. The blue and red circles represent train stations and the black lines illustrate train tracks between them. In such a train network, every trip will arrive or depart at the central station (the red circle). That means that lots of train units will frequent the central station and that the other 'outer' stations will not have a large stock of train units. Therefore, we prefer a trip leaving an 'outer' station to have a similar composition as a trip just arriving before that leaving trip, such that no additional train units will be required. For this reason, adding such a combination of two trips can be considered as a significant route.

Another thing to mention is the number of trips that will be included in a route. If this number is high, the heuristic will try as much as possible to combine train units between these trips. However, this implies that other compositions will not be encouraged and thus leading to inefficient combinations. Therefore, try to keep the number of trips on a route as low as possible. In conclusion, the layout of the train stations and their train tracks, and the number of trips on a route are the most crucial factors in determining the set $\bar{F}$.

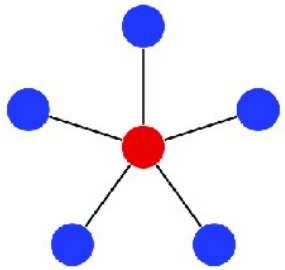


Figure 1: A star graph, where the red circle represents the central station, the blue circles represent the 'outer' stations and the black lines represent trail tracks between these stations

When a route f is chosen to be included in the set $\bar{F}$, a penalty factor C_f can be given if necessary. Also, for this weight factor, there is no explicit general instruction for every data set, although a few factors are considered as essential in penalising a route. As already mentioned, a high number of trips in a route will increase the number of same train units used for multiple trips, but it will decrease the flexibility of the heuristic. Let n_f be the number of trips in a route f. When n_f equals one or when n_f is relatively high, a route should be penalised in some sort. Furthermore, one of the most crucial factors is the transfer time between two different trips. The most straightforward approach is to take the difference between the arrival time of trip i and the departure time of trip j: $s_j - f_i$ or $\sum_{i=1,\dots,n_f-1}(s_{i+1} - f_i)$ whenever $n_f > 2$. Moreover, the minimum demand r_t can be considered valuable. We want to penalise routes whenever a route covers a trip with a deficient demand and one with extremely high demand. Similarly, the maximum length of a train of a trip L_t can be compared with other trips within a given route.

The computation of C_f can be any combination of these mentioned factors, for example using a summation, but also by a product of different factor. Also, the essence of some factors can be emphasised by multiplying it with a large number or taking its square. Keep in mind that C_f must not approach the value of $\min_k c^k$, because our primary goal is still to minimise the number of used train units.

We want to explain the essence of including a weight factor C_f by an example. Let us consider one trip arriving at an 'outer' station at 8:00, named trip A. Furthermore, two trips leave the 'outer' station at 8:10 and 8:15, called respectively trip B and C. Two questions arise: which routes should be included in $\bar{F}$ and which penalty factor should be given to these routes? A route of trip A and B seems more significant than a trip of route A and C because its transfer time is lower. However, if we exclude the route of A and C, no train units of trip A can be used for trip C. If, for example, trip A needs two carriages of type k and trip B and C only one of that same type, both routes can be travelled once. Therefore, we both include these routes in the set of significant routes. To emphasise our preference for route A and B above route A and C because of the lower transfer time, we use a higher penalty factor for the second mentioned route.

Note that examining whether a route belongs to the set of significant routes is just as important as determining the penalty value C_f of such a route. Not only excluding significant routes from the set $\bar{F}$, but also specifying incorrect weight factors will be harmful to the outcome of this algorithm. However, the problem of including insignificant routes in the set $\bar{F}$ can be tackled by penalising them extremely. Nevertheless, proper analysis and trying multiple points of view on the approach will improve the quality of this heuristic.

6.3 Sequencing

The formulation as described above does not consider combining or splitting train units within a route $f \in \bar{F}$. Also, it should theoretically possible that two routes f^1 and f^2 can be sequenced, such that the first route f^1 is performed and afterwards route f^2 without violating any constraints. The last phase will incorporate these problems. To do this, we will execute the AP(k)-model as described in Chapter 5.2.2.2. In order to achieve this, we must determine the number of used train units at every trip w_j^k from the solution of the set covering formulation. If trip j is part of route f, we set $w_j^k \leftarrow w_j^k + x_f^k$ whenever $x_f^k > 0$. Then we can compute the costs of only using train unit type k. We do this by constructing graph G_{AP} based on the trips wherefore $w_j^k > 0$. When this is done for every $k \in K$, we can sum the costs of every AP(k)-model in order to obtain the total costs for the used train units.

7 Local Search

The goal of local search is to improve the objective value of any given feasible solution. To do this, we arbitrarily consider every trip. For every trip t we analyse every possible and feasible combination of train units to see if we end up with a better solution. If this is not the case, we consider another trip until no further improvements are possible.

A general overview of the local search algorithm can be found in Algorithm 3. Before we can start with the heuristic, we need a feasible composition of train units and its total costs (*line 1-2*). Each time we consider a arbitrarly trip $t \in T$. Then, we consider the set of all the subsets of train units, which satisfy the maximum length and capacity constraints, for that random trip t (*line 5-7*). We do this arbitrary such that we do not always end up with the same solution for this heuristic. If we obtain the set $FSEL(t)$, we consider every subset $TU \in FSEL(t)$. For that composition of train units we compute its costs by executing the $AP(k)$-model as described in Chapter 5.2.2.2 (*line 9-14*). If we found a better solution, we update it and in any case remove the set TU from $FSEL(t)$ (*line 15-19*). We do this until no better solution can be found (*line 3*).

The selection of a trip t of set T is done randomly. However, a trip t cannot be chosen twice when in the meanwhile no better solution has been found. Whenever a better solution is found, the *improvement* boolean becomes true, and we can randomly start selecting all the trips in the set T. We only end the process whenever every trip $t \in T$ has been considered (*Line 5*) and no better solution has been found.

Algorithm 3: Local Search

```
   Result: A possible improvement of the given solution
 1 A feasible composition of train units;
 2 BestSolution;
 3 while improvement do
 4 |   improvement ← FALSE;
 5 |   for t ∈ T do
 6 |   |   select a random trip t;
 7 |   |   FSEL(t) ← set of all the subsets of train units with sum of the capacities ≥ r_t and ≤ l_t;
 8 |   |   while FSEL(t) ≠ ∅ do
 9 |   |   |   select a subset TU ∈ FSEL(t) of train units for t;
10 |   |   |   costs_TU ← 0;
11 |   |   |   for k ∈ TU do
12 |   |   |   |   solve AP(k);
13 |   |   |   |   costs_TU ← cost(AP(k)) + costs_TU;
14 |   |   |   end
15 |   |   |   if costs_TU < BestSolution then
16 |   |   |   |   BestSolution ← costs_TU;
17 |   |   |   |   improvement ← TRUE;
18 |   |   |   end
19 |   |   |   FSEL(t) ← FSEL(t) \ {TU};
20 |   |   end
21 |   end
22 end
```

Because we randomly select a trip for consideration, we could execute the local search algorithm for multiple iterations. If we do this, the heuristic will most likely return a different train unit composition and therefore possibly a different minimisation value. Since we search in the set of feasible solutions, we could end at a local minimum. One way to 'leave' this local minimum is instead selecting one random trip (*line 6*), to select two random trips and find a different, cheaper composition for these two trips. Since this process is time-consuming, we stick to selecting only one random trip.

8 Results

Before we move on to the results of our integer linear program and the presented heuristics, we will first describe our four given instances. The layout of the train stations is similar to a star graph, as is shown in Figure 1. As a result, every train trip either departs or arrives at the central station. Furthermore, we have three different types of train units, noted as type A, B and C. The characteristics of each of these train units are presented in Table 1. The instances consists of respectively 200, 318, 564 and 1010 trips.

Table 1: Characteristics of the train units

Train Unit Type	**Costs/year**	**Number of Seats**	**Length (in meters)**
Type A	230,000	500	100
Type B	190,000	360	75
Type C	330,000	640	125

Notice that the used currency is unknown, the length is represented in meters and we use minutes to express time. Furthermore, we set the value of t_{ij}, the minimum time for a train unit to perform trip j after trip i, to 5 minutes, unless stated differently. With that being said, we first start by introducing the results of the integer linear program in Chapter 8.1. Afterwards, in Chapter 8.2 and 8.3, we respectively present the outcomes of the peak period period and the set cover heuristic. In both these chapters, we present the results of these two heuristics while executing the local search technique.

8.1 Integer Linear Program

We start by introducing the results of the integer linear program, as explained in Chapter 4. The outcomes are obtained by using a commercial solver called CPLEX. The results of this model can be seen in Table 2. For each considered instance, we provide the composition and total amount of train units, the total costs and the time it took to compute. The composition represents the number of train units of respectively type A, B and C. The total number of train units is simply the summation of this composition. Lastly, the total costs and the computational time are respectively given in millions and seconds. These total costs can also be computed manually as a summation for every train unit type of the costs per year (Table 1) multiplied by the composition amount (Table 2).

Table 2: The results of the integer linear program

Instance	**Composition**	**Total Train Units**	**Total Costs (in millions)**	**Time (in s)**
1	[121, 65, 31]	217	50.41	1.64
2	[126, 70, 32]	228	52.84	7.16
3	[146, 85, 42]	273	63.59	107.05
4	[319, 198, 107]	624	146.30	328.05

It is remarkable that the computational time is relatively low, especially for smaller instances. For larger instances, the computing time seems to grow exponentially. Also, it is noteworthy that a train unit of type A is the most chosen one. If we inspect the characteristics of the train units more closely, one possible reason seems logical. We compute the costs per year for one seat for only one train unit or stated differently: dividing the costs/year by the number of seats. We then obtain respectively for type A, B and C the following (rounded) values: 460, 528 and 516. In conclusion, the seats of a train unit of type A are relatively the cheapest, such that it would be more convenient to use them.

Furthermore, we want to examine whether changing the 'transfer time' of a train unit t_{ij} will influence the total costs substantively. Therefore, we computed the total costs for every integer $0 <= t_{ij} <= 30$. Figure 2 shows the total costs for every value of t_{ij}, for instances 1, 2 and 3. We excluded instance 4 because it does not particularly fit in this graph, because its total costs are significantly larger than the other three instances. The lines are (obviously) non-decreasing when t_{ij} is increasing. The total costs seem to increase more rapidly for instances with more trips. When we compare the adjustment of $t_{ij} = 5$ to $t_{ij} = 20$, instances 1 and 2 'only' have a percentage increase of 14.2% and 15.7%, while instance 3 and 4 increase by 18.0% and 18.3%. The average difference in percentage when we increase the value of t_{ij} by one, is respectively 0.92%, 1.03%, 1.08% and 1.08% for instances 1 to 4.

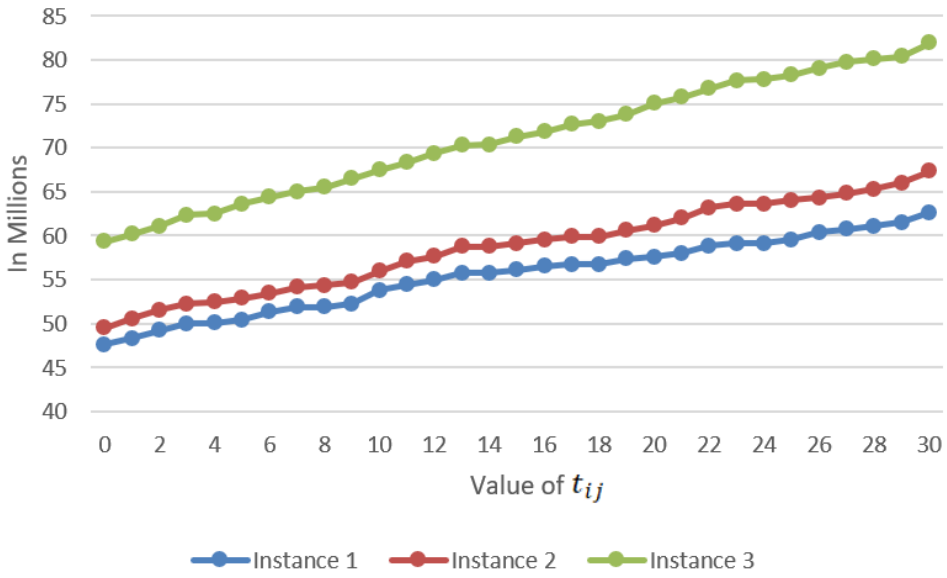


Figure 2: This figure captions the total costs of instance 1, 2 and 3 when we change the value of t_{ij}

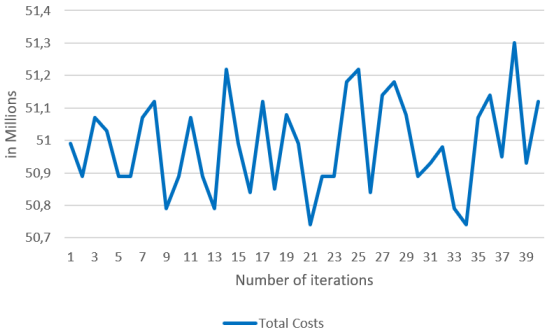
8.2 Peak Period Heuristic

Before we start to present the upper bound results of the heuristics, we start by showing the lower bound as computed in Chapter 5.1. These results can be seen in Table 3. Our main point of interest, the difference between the lower bound and the exact model, is outstanding, while the gap to the solution is only a few percent. Furthermore, it is also remarkable to see how close the number of used train units of the lower bound are in comparison to the integer linear program, with respectively only 1, 2, 8 and 6 train unit(s) difference. Also the computational time is noteworthy, especially for the smaller cases where the computed time is only a few seconds. We will use this lower bound as comparison for all the other heuristics.

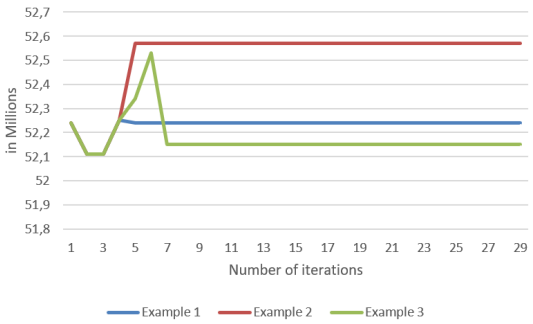
Table 3: The results of the lower bound

Instance	**Composition**	**Total Train Units**	**Total Costs (in millions)**	**Gap (in %)**	**Time (in s)**
1	[119, 66, 31]	216	50.14	-0.54	0.9
2	[124, 71, 31]	226	52.24	-1.14	1.2
3	[146, 76, 43]	265	62.21	-2.17	2.4
4	[316, 195, 107]	618	145.04	-0.86	16.2

For every instance, we will consider the results of that upper bound itself, but also the plausible better solution computed by the local search approach (Chapter 7). Although the peak period heuristic provides an outcome for every iteration, we only consider the local search technique for the lowest objective value of all iterations. Since there is a trade-off between execution time and performance, we must decide an adequate number of iterations. In Figure 3a, we performed 40 iterations of the local search technique over one given solution of instance 1. It shows that the total costs fluctuate heavenly. Since the execution time is high and the performance of the local search technique is decent, as we will see, we set the number of iterations for this algorithm to 10. By doing this, we compute decent solutions within sufficient time.



(a) This figure displays the total costs for every executed iteration of the local search.



(b) This figure shows the total costs for 3 different executions of the peak period heuristic.

Figure 3: Extensive analysis for the parameter setting

The first upper bound that we will consider is the peak period heuristic established by Cacchiani et al. (2019). For the peak period heuristic we had to set two different parameter values: n_c and n_{iter}. Their paper decided to fix these values to $n_c = 10$ and $n_{iter} = 20$. However, by extensive analysis we come to another conclusion. The main goal in setting the value of n_c is finding the largest number of trips to be covered within n_c iterations. The value $n_c = 10$ seems reasonable, since increasing it would practically never increase the performance of the heuristic, but increase its execution time. Lowering the value could possibly harm the solution, but for convenience we stick to $n_c = 10$.

In Figure 3b we display the objective values of each iteration, where each example is an execution of the peak period heuristic for instance 1. The best solution does not seem to improve after seven iterations, and the same happens if we compare this for multiple executions and different instances. Therefore, we set $n_{iter} = 7$, such that the execution time will decrease and it will most likely not harm the performance. The results of the peak period heuristic with these specific parameter settings can be found in Table 4.

Table 4: The results of the peak period heuristic

Inst.	**Local Search?**	**Composition**	**Total Train Units**	**Total Costs (in millions)**	**Gap with optimal solution (in %)**	**LB Gap (in %)**	**Time (in s)**
1	*Without*	[122, 67, 34]	223	52.01	3.17	3.73	17.2
	With LS	[119, 67, 32]	218	50.66	0.59	1.04	221.4
2	*Without*	[138, 83, 41]	262	61.04	15.52	16.85	87.2
	With LS	[126, 74, 35]	235	54.59	3.31	4.50	617.4
3	*Without*	[171, 97, 63]	332	78.78	23.89	26.64	298.5
	With LS	[153, 84, 48]	285	66.99	5.35	7.68	3138.6
4	*Without*	[345, 221, 117]	683	159.95	9.33	10.28	2,526.0
	With LS	[317, 205, 114]	636	149.48	2.17	3.06	29,004.0

As can be seen, the results of the peak period itself are relatively weak compared to the lower bound and the optimal solution. Especially instance 3 performs dramatically with a percentage gap to the lower bound of over 25%. Not only is the heuristic far from accurate, but also the algorithm is extremely time-consuming. The local search technique is as expected slow, but its results are quite promising as it approaches the exact solution. Although the results with this algorithm are better, there is still room for improvement, e.g. instance 3 has a percentage gap of 7.7% compared to the lower bound.

8.3 Set Cover Heuristic

As being explained in Chapter 6, the set cover heuristic is an intuitive solution approach, which must be tackled differently for each different data set. Since the train stations of our given data set can be illustrated as a star graph illustrated Figure 1, we use the same approach as the example given in Chapter 6.2. In that chapter, we stated that for such a layout, a significant route is a combination of a trip leaving an 'outer' station after a trip which just arrived at this 'outer' station. To conclude, the set of significant trips $\bar{F}$ consists of a route which departs at the central station and finally arrives at the central station and in the meanwhile visiting only one other station. Besides these routes, we include routes which contain only one trip j, for every $j \in V$. We want to compare the effect of using a weight factor and not using one. Therefore, we first use the set of significant trips $\bar{F}$ and calculate the outcome without using weight factor C_f (by setting $C_f = 0$) The results of this approach can be viewed in Table 5.

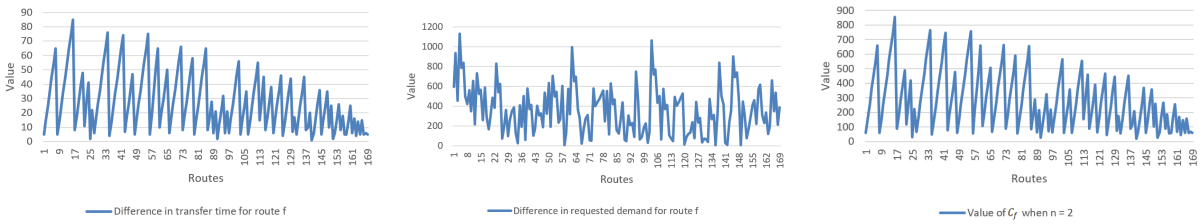
Table 5: The results of the set cover heuristic without including a weight factor C_f

Inst.	Local Search?	Composition	Total Train Units	Total Costs (in millions)	Gap with optimal solution (in %)	LB Gap (in %)	Time (in s)
1	*Without*	[122, 68, 33]	223	51.87	2.90	3.45	1.4
	With LS	[118, 66, 33]	217	50.57	0.32	0.86	188.3
2	*Without*	[139, 85, 38]	262	60.66	14.80	16.12	1.6
	With LS	[126, 73, 34]	233	54.07	2.33	3.50	796.8
3	*Without*	[160, 106, 49]	315	73.11	14.97	17.52	1.8
	With LS	[148, 83, 48]	279	65.65	3.24	5.53	4,368.9
4	*Without*	[339, 213, 114]	666	156.06	6.67	7.60	4.8
	With LS	[316, 200, 114]	630	148.30	1.37	2.25	20,036.4

If we compare these results with the results of the peak period heuristic, we observe that for each instance the set cover heuristic performs better. Not only are the total costs lower, but also the execution time is significantly better compared to Cacchiani et al. (2019). Also, when we compare the results while using local search, the objective values of the set cover heuristic are always lower than the one of the peak period heuristic. Although the results are obtained rapidly, we question the performance of the heuristic. Without using local search, the results are just slightly better than the peak period heuristic, but at the same time, the gap to the exact solution and lower bound is significantly large.

The last results that we will discuss is the set cover heuristic containing a weight factor C_f. Since this process is also intuitive, some extensive analysis is necessary. We want to consider three factors to influence the value of C_f, namely the number of trips in a route n_f, the difference between departure and arrival time and the difference of requested demand between two trips i, j. Since we only consider routes with $n_f = 1$ and $n_f = 2$, we prefer routes with $n_f = 2$ such that train units between two trips can be 'combined' within one route. Furthermore, it makes sense to prefer a low transfer time for sequencing trips, which can be mathematically presented as $s_j - f_i - t_{ij}$. A less significant, but yet important factor is the difference in the demand between two trips: $|d_j - d_i|$.

Since our goal is to minimise the number of used train units, we do not want our penalty factor C_f to approach the size of c^k. Furthermore, we want to penalise the number of trips in a route n respectively, the difference in transfer time and the difference in demand the most. Therefore, we analyse the data such that we can redefine, give a higher weight, square or root one of the penalty factors. We do this with the help of Figure 4. Here, the x-axis represents all significant routes for instance 1 whenever $n_f = 2$.



(a) The weight factor when we consider $s_j - f_i - t_{ij}$

(b) The weight factor when we consider $|d_j - d_i|$

(c) The weight factor when we consider $10(s_j - f_i - t_{ij}) + \sqrt[3]{|r_j - r_i|}$

Figure 4: Motivation for the selection of C_f.

We start by considering the difference in transfer time, as can be seen in Figure 4a. The values of this factor differ between 0 and 85, where a value close to 0 is better. The values of the difference in demand are somewhat more fluctuating, since the values differ between 2 and 1,132, as shown in Figure 4b. Because we want to lower the significance of the difference in demand and emphasise the significance in transfer time, we experiment such that the difference in transfer time would be the deciding factor. Finally, we end up with the following formula for these two factors: $10(s_j - f_i - t_{ij}) + \sqrt[3]{|r_j - r_i|}$, as can be seen in Figure 4c. Since the importance of having two trips in a route instead of only one is even more essential, we want to give C_f such a high value whenever $n_f = 1$, such that it will exceed C_f whenever $n_f = 2$. Putting this all together, we come up with the following definition for the weight factor C_f:

$$C_f = \begin{cases} 10(s_j - f_i - t_{ij}) + \sqrt[3]{|r_j - r_i|} & \text{if } n_f = 2, \\ 1000 & \text{if } n_f = 1. \end{cases} \tag{22}$$

If we use this expression to solve the integer linear program of equations (18)-(21), we end up with the results listed in Table 6. The results approach the (exact) objective value of the integer linear program. Furthermore, the solution time is for all instances just a few seconds, which is also very promising. Even when we try to improve the solution using the local search technique, the objective value decreases even further and in the case of instance 1 even matches the exact solution. The other instances have an percentage gap to the exact solution of not even 1%.

Table 6: The results of the set cover heuristic containing a weight factor C_f as prescribed

Inst.	Local Search?	Composition	Total Train Units	Total Costs (in millions)	Solution Gap (in %)	LB Gap (in %)	Time (in s)
1	*Without*	[121, 65, 32]	218	50.74	0.65	1.20	1.9
	With LS	[121, 65, 31]	217	50.41	-	0.54	201.6
2	*Without*	[126, 73, 36]	235	54.73	3.58	4.77	2.1
	With LS	[125, 73, 33]	231	53.51	0.89	2.43	775.1
3	*Without*	[146, 83, 48]	277	65.19	2.52	4.79	3.0
	With LS	[143, 83, 47]	273	64.17	0.91	3.15	3291.2
4	*Without*	[317, 199, 114]	630	148.34	1.39	2.28	7.2
	With LS	[311, 201, 113]	625	147.01	0.49	1.36	18,045.6

9 Conclusion

In this thesis, we analysed the train unit assignment problem, and we introduced different solution approaches to solve this problem. We started with an integer linear program which could solve this problem exactly. The computational time was decent, although it seems to increase exponentially for bigger instances. Then we considered the peak period heuristic proposed in Cacchiani et al. (2019). Their paper came up with a lower bound and an algorithm represented as the upper bound. Their lower bound was based on the minimisation of train units over a subset of all trips. This so-called set of incompatible trips S was the largest set of trips which cannot be sequenced with each other. This lower bound generated excellent results, while the lower bound is close to the exact solution. Therefore, we used this lower bound as a comparison for all the other heuristics.

The upper bound continued with this incompatible set S. For the other set of trips, the so-called uncritical trips, it computed the cheapest set of train units for a specific trip. Although their paper came up with excellent results, the results for our specific data set, where the layout of the train stations was similar to a star graph, the peak period heuristic performs miserably. The outcomes were way too high compared to the exact solution and the lower bound. Furthermore, the execution time was even worse than the computational time of the integer linear program.

Afterwards we introduced the set cover heuristic, which is similar to the actual set cover formulation. The only difference is the fact that we do not consider all possible trips, but only a small subset of so-called significant trips $\bar{F}$. There is a possibility to even emphasise the significance of a route f by adding a weight factor C_f to it. While there is no general description or procedure for this heuristic, intuition is necessary. Even without adding a weight value to the routes, the performance of the set cover heuristic was better than the peak period heuristic. Although the results were not yet close to the exact solution and the lower bound, the set cover heuristic produced better results in significantly less time than the peak period algorithm.

When we add a weight factor C_f to each route in $\bar{F}$, the outcomes become even closer to the optimal solution, while the execution time is still excellent. The percentage gap between the outcome of this heuristic and the exact solution extends to a maximum of a few percentages. Another benefit is the execution time: this algorithm is solved within a few seconds, even for larger instances.

Lastly, we presented the local search algorithm, whose goal was to improve any given solution. It accomplished this by checking (for every trip) whether there is a more cheaper train unit composition than the current one. If a cheaper composition has been found, it starts again by searching for another cheaper train unit composition. Although this process is time-consuming, as the results did confirm, the heuristic performance was magnificent. It decreases the total costs of the peak period heuristic to an acceptable level. However, its best application was in combination with the set cover heuristic, where it approaches and even matched the exact solutions.

However, there are some suggestions for further research. Firstly, our thesis is in conflict with the paper of Cacchiani et al. (2019) regarding the peak period heuristic. It seems the usage of different data sets affects the performance of that heuristic. It would be interesting to replicate and possibly confirm the effectiveness with a comparable data set as used in their paper. However, the other way around, it is of interest to examine the performance of the set cover heuristic for alternative data sets, especially for a more complex lay-out of train stations. Furthermore, we can make the TUAP more complex by

including additional constraints, such as taking the order of the train units into account for more realistic transferring of those carriages. Other possibly ideas are the introduction of the railway shunting problem or considering rescheduling in case of possibly delays.

In conclusion, the peak period heuristic does not appear to give acceptable results for this specific data set. However, the set cover heuristic gave excellent results, especially in combination with an introduced weight factor and the local search technique. Although the local search technique is time-consuming, the results approach or even match the exact solution.

References

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*.

Cacchiani, V. and Toth, P. (2012). Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737.

Cadarso, L. and Marín, Á. (2011). Robust rolling stock in rapid transit networks. *Computers & Operations Research*, 38(8):1131–1142.

Cadarso, L. and Marín, Á. (2014). Improving robustness of rolling stock circulations in rapid transit networks. *Computers & Operations Research*, 51:146–159.

Caprara, A. (2015). Timetabling and assignment problems in railway planning and integer multicommodity flow. *Networks*, 66(1):1–10.

Caprara, A., Fischetti, M., and Toth, P. (2002). Modeling and solving the train timetabling problem. *Operations research*, 50(5):851–861.

Cordeau, J.-F., Toth, P., and Vigo, D. (1998). A survey of optimization models for train routing and scheduling. *Transportation science*, 32(4):380–404.

Fang, W., Yang, S., and Yao, X. (2015). A survey on problem models and solution approaches to rescheduling in railway networks. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2997–3016.

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.

Ghoseiri, K., Szidarovszky, F., and Asgharpour, M. J. (2004). A multi-objective train scheduling model and solution. *Transportation research part B: Methodological*, 38(10):927–952.

Goossens, J.-W., van Hoesel, S., and Kroon, L. (2006). On solving multi-type railway line planning problems. *European Journal of Operational Research*, 168(2):403–424.

Haahr, J. T., Wagenaar, J. C., Veelenturf, L. P., and Kroon, L. G. (2016). A comparison of two exact methods for passenger railway rolling stock (re) scheduling. *Transportation Research Part E: Logistics and Transportation Review*, 91:15–32.

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

UIC (2018). Railway statistics 2017. `https://uic.org/IMG/pdf/uic-statistics-synopsis-2017.pdf`.

A Appendix

In this chapter, we briefly describe the included programming code and files. We explain the MATLAB-files in chronological order. We thereby show the parents and children relations by the usage of bullet points. A nested list is a collection of children of a parent. We represent the name of the classes as italic words and its description follows immediately. Each MATLAB-folder consists of a so-called dataMatlab.mat file, which has some minor changes to the original given data set. The departure and arrival times are noted in minutes, where 0 minutes corresponds to 0:00, such that 1:00 corresponds to 60. The locations of the train stations are not represented by letters, but by numbers. Lastly, the last column, which shows the train line of the trip, is changed to the maximum length of that train track.

A.1 Integer Linear Program

The integer linear program is solved by MATLAB, but also by Aimms. Therefore, both files are available in the ZIP-file. We will only explain the MATLAB-file, while the Aimms file is pretty straightforward.

- *mainModel*: This is the main model which solves the formulation;
  - *createArcsMatrixLowerBound*: This class computes a so-called adjadency-matrix, which represents the existing arcs. This is similar to the formulation of a_{ij} in Chapter 5.1.1;
  - *formulation*: This class computes the required matrices for the cplexmilp class and returns the outcome;
    - *cplexmilp*: This is the commercial solver CPLEX and solves mixed-integer linear programs;

A.2 Peak Period Heuristic

This is the peak period heuristic which is introduced by Cacchiani et al. (2019).

- *TOTALMODEL* This is the main model which solves the heuristic;
  - *createArcsMatrixLowerBound*: This class computes a so-called arcs-matrix, which represents the existing arcs. This is similar to the formulation of a_{ij} in Chapter 5.1.1;
  - *LowerBoundFirstModel*: This class computes the formulation introduced in Chapter 5.1.1;
    - *cplexmilp*: This is the commercial solver CPLEX and solves mixed-integer linear programs;
  - *changeXmatrix1to2Lower*: This class transfers the output of the first stage for usage for the second stage of the lower bound;
  - *LowerBoundSecondModel*: This class computes the dual formulation of the formulation introduced in Chapter 5.1.2;
    - *cplexmilp*: This is the commercial solver CPLEX and solves mixed-integer linear programs;
  - *changeXmatrixAfter2Lower*: This class changes the output of the second stage of the lower bound, such that it can be used for the computation of the upper bound;
  - *AssignmentCritical*: This class is the main model which is responsible for the assignment of the critical trips as described in Chapter 5.2.2.1;
    - *computeFSEL*: This class computes the set of all the subsets of train units with a sum of the capacities $\geq r_j$ and $\leq l_j$;
    - *APkModel*: This class computes the costs of a given train unit composition, also known as the $AP(k)$-model, described in Chapter 5.2.2.2. It uses the MATLAB-function *matchpairs*;
    - *changeTripOrderFirst*: This class changes the trip order in the set of critical trips whenever the number of iterations is equal to n_c;

        * *changeTripOrderSecond*: This class changes the trip order in the set of critical trips whenever the number of iterations is equal to $2n_c$;
    - *AssignmentUncritical*: This class is similar to the class *AssignmentCritical*, but this class is used for the assignment of the uncritical trips as described in Chapter 5.2.2.3;
        * *computeFSEL*: This class computes the set of all the subsets of train units with a sum of the capacities $\geq r_j$ and $\leq l_j$;
        * *APkModel*: This class computes the costs of a given train unit composition, also known as the $AP(k)$-model, described in Chapter 5.2.2.2. It uses the MATLAB-function *matchpairs*;
    - *feasibilityPhase*: This class executes the feasibility phase as described in Chapter 5.2.3;
        * *computeFSEL*: This class computes the set of all the subsets of train units with a sum of the capacities $\geq r_j$ and $\leq l_j$;
        * *APkModel*: This class computes the costs of a given train unit composition, also known as the $AP(k)$-model, described in Chapter 5.2.2.2. It uses the MATLAB-function *matchpairs*;
    - *endAdjustments*: This class updates the set of critical and uncritical trips;

A.3 Set Cover Heuristic

Here we describe the MATLAB-files corresponding to the set cover heuristic as explained in Chapter 6.

- *mainModel*: This class is the main model which solves the heuristic. The letters a, b, c, d represent the used weight factor as described in Chapter 8.3;
    - *makeMatrix*: This class determines the set of significant trips $\bar{F}$ and their corresponding weight factors C_f;
    - *setCoverModel*: This class solves the set cover formulation as described in Chapter 6.1. In case the weight factor must be ignored, the word 'fMatrix' in line 8 must be deleted;
        * *cplexmilp*: This is the commerical solver CPLEX and solves mixed-integer linear programs;
    - *changeXmatrix*: This class combines all the used routes computes by the *setCoverModel* and computes its costs;
    - *APkModel*: This class computes the costs of a given train unit composition, also known as the $AP(k)$-model, described in Chapter 5.2.2.2. It uses the MATLAB-function *matchpairs*;

A.4 Local Search

The local search technique is not a particular map or file in the ZIP-file. It is the end of the *TOTALMODEL* of the peak period heuristic and *mainModel* of the set cover heuristic.

- *selectionProcedure*: This class runs for h times, where h is the number of iterations used. It remembers the best solution and its composition. It also randomly selects a trip;
    - *selectingComposition*: This class considers every possible composition for a trip and determines the cheapest combination of train units;
        * *computeFSEL*: This class computes the set of all the subsets of train units with a sum of the capacities $\geq r_j$ and $\leq l_j$;
        * *APkModel*: This class computes the costs of a given train unit composition, also known as the $AP(k)$-model, described in Chapter 5.2.2.2. It uses the MATLAB-function *matchpairs*;
- *APkModel*: This class computes the costs of a given train unit composition, also known as the $AP(k)$-model, described in Chapter 5.2.2.2. It uses the MATLAB-function *matchpairs*;