ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis in Econometrics en Operations Research

Constructing a more efficient railway stock planning

Name student: Stephan Kroon
Student ID number: 455100

Supervisor: R. N. van Lieshout
Second assessor: T. A. B. Dollevoet

Date final version: 7 July 2019

Abstract

After a timetable has been made in which all trains run, enough seats need to be present for all travelers on each trip. We are interested in buying new train units and are therefore solving a train unit assignment problem (TUAP). This problem is NP-hard and therefore difficult to solve. We solve it by first constructing a lower bound to this problem using Cacchiani et al. (2010) and Cacchiani et al. (2019) by solving the problem to the peak period, a set of incompatible trips. Then, the heuristic based on Cacchiani et al. (2019) is used for finding a solution to the original problem using the found lower bound. The heuristic itself is also being modified to possibly improve their results and lower the computational time in three different ways. One is changing the way that train units are sorted. Another one is fixing the assignment of the train units of all trips in set S, a set which contains incompatible trips. The last modification also fixes the assignment of all trips in set S and also uses a tabu-search technique adapted from de Werra and Hertz (1989). In the results, four instances are used. In all instances, the lower bound is never found in one heuristic. However, good results are found in three instances. Moreover, two modifications lead to better results for three instances, while also having a lower computational time. And therefore, the two modifications can be used in real-world instances to lower the computational time of the existing algorithm by a lot while also obtaining better results than the original heuristic.

Contents

1 Introduction

Trains run using an already made timetable. In such a timetable, no information is given about the train unit compositions of each train trip. Furthermore, on each trip enough seats should be available for all passengers to prevent crowded trains and to keep the passengers satisfied. Since platforms are not always long enough, a maximum length is given for each trip. We are interested in buying new trains for a timetable like this, and assigning those train units to trips. Train units are expensive and therefore train units perform multiple trips in sequence. However, no information is given about the trip sequencing in a timetable. It is therefore useful to use a method which is able to find combinations for all trips to perform trip sequencing and to reduce the overall total costs of buying train units. However, passenger demand and length constraints should not be violated.

The TUAP is solved by first computing a lower bound to the problem, based on Cacchiani et al. (2010). This lower bound is based on a subset of the original problem, a set of incompatible trips. After that, a peak period heuristic is used to solve the whole problem using this lower bound, developed in Cacchiani et al. (2019). This heuristic is modified three times to potentially reduce the total costs and the computational time. One modification is sorting train units differently. Another modification is fixing the train units that are in the set of incompatible trips. The last modification extends the previous modification by using a tabu search technique described in de Werra and Hertz (1989).

Four instances are used, which contain between 200 and 1010 trips. None of those instances are able to find a combination which is equal to the obtained lower bound. However, two modifications lead to lower computational times and better results than the original peak period heuristic. In particular, the costs of the biggest instance are lowered by almost 5% in comparison with the original heuristic while also being much faster in solving it. Therefore, the two modifications, fixing set S with or without tabu search, can be used in real-world instances.

To solve this problem, we first describe a more precise problem statement including some notation in Section 2. Then, we come up with already done literature about this subject in Section 3. After that, we describe our used methods in Section 4, which contains the methods to find a set of incompatible trips (Section 4.1) and the corresponding lower bound (Section 4.2). It also contains a heuristic which is related to the already found lower bound (Section 4.3). This heuristic is modified in Section 5 to find results with lower costs or results with similar costs that are obtained much faster. After that, the results are given in Section 6. The results of the instances are given at first (Section 6.1) and after that the other obtained results using the described methodology. We end with a conclusion and discussion in Section 7.

2 Problem statement

After a timetable for the arriving and departing trains has been made, train unit are assigned to this timetable. Assigning train units to this timetable is solving a train unit assignment problem (TUAP) and is NP-hard.

The given timetable consists of different trips, which are denoted as set $T = \{1, 2, \ldots, n\}$. Every trip t has a departure d_t and arrival time a_t ($\forall t \in T$). They also have a passenger demand, p_t, which must be met for every trip. Furthermore, all trips have a maximum length which are different for each line, due to the fact that some train stations have a shorter platform than other train station, which are denoted as l_t ($\forall t \in T$). Different train unit types can be bought, which are denoted in set $U = \{1, 2, \ldots, m\}$. Every train unit type, b ($b \in U$), has its own cost (c_b), number of seats (s_b) and length (q_b).

The goal is to buy a combination of train units which have the lowest total cost, while the seat demand is met. For each trip, multiple train units can be assigned to a trip to meet the seat demand. It is even possible to combine different train unit types and there is no maximum amount of combined train units for a trip. However, the combination of train units should never exceed the maximum length given for each line since an arriving train can not be longer than any platform on its trip.

A train unit can perform multiple trips in sequence if and only if the arrival time on a train station plus 5 minutes is earlier than the departure time of the same station. Those 5 minutes are used for shunting operations. If two different trips can not be performed in sequence, they are considered incompatible. This, however, means that dead heading is not allowed.

3 Literature

A lot of research has been done about the railways. Multiple method to solve a TUAP have been developed. Cacchiani et al. (2010) proposed a 3-step method to find a solution with low costs. The first two steps, finding a lower bound to this problem, are used in our methods. The last step uses column-generation. The results of this last step are promising. Cacchiani et al. (2013) use the same method to find a lower bound to this problem. The last step is different: a Lagrangian heuristic. This method turns out to be faster.

Similar problems are also being solved. Lin and Kwan (2014), for instance, proposed a two-phase approach for the train unit scheduling problem (TUSP). The difference between a TUAP and TUSP is that a TUSP also takes operational requirements into account when solving the problem. The first phase in their method is to model the problem as a fixed-charge multicommodity flow problem (FCMF) and solve it using a branch-and-price approach. The second phase is to model the multidimensional matching problem as a mixed integer problem (MIP) and solve it with a column-and-dependent-row generation. Lin and Kwan (2016) developed a branch-and-price approach for solving the integer multicommodity flow problem. This resulted in schedules being better than manual schedules.

In a timetable, employees should also be assigned to train units. Abbink et al. (2005) developed a method for crew scheduling for the dutch railway operator NS. The new schedule reduces the personnel costs by 1.2% per year, which could be even more.

One of the most important steps is building a timetable. Ghoseiri et al. (2004) solved a train scheduling model using an objective function including both train fuel consumption and passenger-time. Lowering the fuel consumption is advantageous for the railway company and passenger-time for the travellers. This problem is solved using a two-phase approach.

4 Methodology

Solving a TUAP is NP-hard. To solve this problem, we first find a set of incompatible trips, the peak period, and the corresponding lower bound based on the methods described in Cacchiani et al. (2010), in Sections 4.1 and 4.2. After that, we describe a heuristic developed in Cacchiani et al. (2019) in Section 4.3 to find a feasible solution to the original problem.

4.1 Determine set of incompatible trips

In order to compute a lower bound, we find a set of incompatible trips, which form the peak period. This peak period corresponds with different and incompatible trips. Therefore, trips in the peak period are assigned to different train units.

Consider the undirected graph $G = (V, E)$, which contain all trips as vertices. When a trip can be done in sequence by the same train unit, vertices i and j are connected with edge (i, j). All i vertices ($i \in T$) should be covered with at least p_i paths. Those paths are the required seats for trip i. This means that we solve the minimum flow problem with demands to find the set of incompatible trips.

To solve this problem, we construct a directed graph $G' = (V', A)$ with the same vertices as in graph G, however the edges in graph G become directed arcs, A, in graph G'. Furthermore, we add 2 new vertices: A source, σ, and a sink, τ. A vertex i without an incoming edge, receives an incoming edge: The source σ to vertex i. A vertex j without an outgoing edge, gets an outgoing edge: vertex j to sink τ. To furthermore solve this problem, we minimize the outgoing flow of the source. The flow of vertices are noted as f_k with $k \in E'$. Also, the flow entering vertex i, $i \in T$ and the flow leaving vertex i, $i \in T$, are denoted as δ^- and δ^+ respectively. The formulation of solving the minimum flow problem is:

$$\textbf{min} \sum_{s \in \delta^+_{(\sigma)}} f_s \tag{1}$$

$$\textbf{s.t.} \sum_{\alpha \in \delta^-_{(j)}} f_\alpha = \sum_{\alpha \in \delta^+_{(j)}} f_\alpha \qquad \forall j \in V' \setminus \{\sigma, \tau\} \tag{2}$$

$$\sum_{\alpha \in \delta^-_{(j)}} f_\alpha \geq p_i \qquad \forall j \in V' \setminus \{\sigma, \tau\} \tag{3}$$

$$f_\alpha \geq 0 \qquad \forall \alpha \in A \tag{4}$$

The objective, equation 1, is to minimize the flow leaving the source. Constraint 2 ensures that all incoming flow of all vertices, except the source and sink, equals to all outgoing flow of those vertices. Constraint 3 ensures that all vertices, except the source and sink, get at least enough seats. All flow must be positive (constraint 4). Since all seat demands are integer, we get an integer flow, because the constraint matrix is totally unimodular.

Let set S be the set with all trips that are in the peak period. After solving this problem, we take a look at the dual variables of constraint 3. The set is build by the selected vertices of the dual variables of this constraint. In particular, all dual variables that are ≥ 1 are selected to be in set S, all trips that are in the peak period. Those trips are used to determine the lower bound.

4.2 Determine lower bound

After we find all trips that build a maximum-weight stable set, we construct the corresponding lower bound by calculating the number of train units needed for each trip. All trips in the found set are incompatible and therefore trip sequencing is not possible. However, this lower bound is not guaranteed feasible in the original problem because the used set is only a subset of the original problem.

Denote variable n_{jk} as the total number of train unit types k which are assigned to trip j with $j \in S$ and $k \in U$. The lower bound is obtained when solving the IP problem as given in model

(5)-(8). This solution is a lower bound because all restrictions are modelled as hard constraints for the peak period, a subset of the original problem.

$$\textbf{min} \sum_{k \in T} \sum_{j \in S} c_k w_{jk} \tag{5}$$

$$\textbf{s.t.} \sum_{k \in T} s_k w_{jk} \geq p_j \qquad j \in S \tag{6}$$

$$\sum_{k \in T} q_b w_{jk} \leq l_j \qquad j \in S \tag{7}$$

$$w_{jk} \in \mathbb{N} \qquad k \in T, j \in S \tag{8}$$

The objective, equation 5, is to minimize the number of seats needed. Constraint 6 ensure that enough seats are present for each trip. For all incompatible trips, the length of all train units combined should not exceed the maximum length of that trip, which is ensured by constraint 7. Constraint 8 ensure that the used train units are integer and at least 0, as train units can not be cut and can not be negative.

4.3 Peak Period Heuristic

After computing the lower bound, we only have information about the minimum required train units. It is therefore unknown which train units are assigned to which trip in the original problem. We then need to solve the original problem with use of this obtained lower bound. With the use of all train unit types needed in the lower bound, we use a method which tries to find a combination to only assign the used train units found in the lower bound. If a feasible solution is found, which means that all trips are assigned using only these train units, then the lower bound is found. If this does not give a feasible solution, a best solution is found by assigning all trips to additional train units.

For the heuristic, we can use two elements that are calculated before: the value of the lower bound (all different train unit types needed) and the set of incompatible trips (set S). If we find a combination in the heuristic which has the same costs as the lower bound, the found solution is optimal. To find this combination, we use, in particular, the total number of different train unit types used in the lower bound. Let $\bar{d}_b$ ($b \in U$) be the number of train unit types used. If this combination gives a feasible solution, then the algorithm can be executed. To find this combination, we also use the second element, the set of incompatible trips determined in Section 4.1. Those trips match the peak period and, therefore, are considered critical trips. Only trips that are hard to assign end up in the set of critical trips. Consequently, the heuristic starts assigning train units to those trips.

The heuristic is an iterative process which consists of 3 different phases: the initialization phase (Section 4.3.1), the constructive phase (Section 4.3.2) and the feasibility phase (Section 4.3.3). After the initialization phase has been finished, the constructive and feasibility phase alternate, until a feasible solution has been found with costs equal to the lower bound or the maximum iterations have been reached or the heuristic runs out of time. The algorithm about the peak period heuristic is given below. We also give some additional information about assigning trips (Section 4.3.4), the assignment problem (Section 4.3.5) and updating sets C and N (Section 4.3.6). For the algorithm given below, we use $n_{iter} = 20$.

Algorithm 1 Peak Period Heuristic

```
Start Initialization Phase
h ← 1
while not done do
    Start Constructive Phase
    if Feasibile Solution has been found in Constructive Phase then
        Return feasibile solution found
        break
    Start Feasibility Phase
    if Better solution has been found in Feasibility Phase then
        Update best solution found
    if Heuristic running over 12 hours or h ≥ n_iter then
        Return best solution found
        break
    h ← h + 1
    Update critical and uncritical tripsets
```

4.3.1 Initialization phase

In the initialization phase, we use the two elements: the lower bound and set of incompatible trips. All trips are either in the set of the critical trips or in the sets of uncritical trips. Let set C be all critical trips and set N be all uncritical trips, with $C \subseteq T$, $N \subseteq T$ and $U = T \setminus C$. In both sets, the trips are initially ordered based on their departure time. That changes, however, for set C during the Constructive phase (Section 4.3.2). Moreover, the best solution costs are initialized as ∞ and we use $n_c = 10$.

4.3.2 Constructive phase

The constructive phase tries to find a combination such that the already found lower bound is a feasible solution to the problem. It only uses the train units $\bar{d}_b$ ($b \in U$). If a combination leads to a feasible solution. The heuristic can stop as it found the lower bound and so the optimal solution to this problem. When assigning train units to trips, we first assign train units to trips that are in set C and after that in set N. If no combination has been found that gives a feasible solution, then some trips remain uncovered. The uncovered trips are being assigned during the Feasibility phase.

4.3.3 Feasibility phase

If all trips are covered, this means only $\bar{d}_b$ train units are used, then the found solution is optimal. However, if not all trips have been covered, the feasibility phase is being executed. As we have to buy trains, we do not have restrictions on the total number of trains needed. This ends up always finding a feasible solution during this phase if at least one train unit combination exist for each trip that does not violate the length or demand constraint. Furthermore, the feasibility phase keeps track of the best found solution in this phase. This solution is only replaced by another solution if a better solution has been found. This best feasible solution is returned if the maximum iterations has been reached or when running for more than 12 hours.

4.3.4 Assigning trips

During the phases, trips have to be assigned. Assigning trips is the most important step during all the three phases. Therefore, we provide the algorithm for assigning trips in Algorithm 2, which can be found below.

Algorithm 2 Assigning trips

```
1: uncov ← all to be assigned trips, bool ← true, i ← 0
2: while i < n_c and uncov is not empty do
3:     uncov ← all to be assigned trips
4:     for All to be assigned trips do
5:         Get all feasible train unit combinations for the current trip
6:         Sort all train unit combinations based on their total seats
7:         while Combinations can be tested and current trip is not assigned do
8:             Consider the first combination
9:             Solve Assignment Problem for all train units with this combination
10:            if feasible combination has been found or no maximum train units are used then
11:                Current trip is assigned with the current combination
12:            else
13:                Remove current combination
14:    if Critical Trips are assigned then
15:        i ← i + 1
16:        if i ≥ n_c and not all trips are assigned and bool then
17:            i ← 0, bool ← false
18:        Change the order in which all trips will be assigned in the next iteration
19:    else
20:        i ← n_c
21:    if not all trips are covered and i < n_c then
22:        Remove all assigned combinations from current solution
```

In line 1, the set *uncov* is made which contains all to be assigned trips, a Boolean variable is set to true and the iteration counter i is set to 0. In lines 3, the *uncov* set is set back to its original. In line 4, all to be assigned trips are visited in the order they are given. In lines 5 and 6, train unit combination are sorted based on their seat demand. It is sorted on seats to prevent over-covering a trip. In lines 7 till 13, the combinations are visited one by one until a feasible combination has been found or all combinations have been visited. In line 9, the assignment problem is solved to test whether or not the current combination is feasible, for all train unit types, which is described much more in detail in Section 4.3.5.

In line 10, the current combination is only accepted if the combination is feasible or when assigning without maximum number of train units. The maximum number of train units being used is not used during the Feasibility phase, but is used during the Constructive phase. The current combination is removed if the combination is not feasible and the maximum number of train units is used. Then, try the next combination if any is left. Otherwise, visit the next trip that should be assigned and leave the current trip uncovered. In line 14 till 17, when assigning to critical trips only, the iteration counter is incremented by 1. In case that some trips are left uncovered and the iteration counter is equal to n_c and the Boolean variable is true, the iteration counter, i, is set back to 0. In addition, the Boolean variable is set to false. In line 18, the order in which the trips are being visited in the next iteration is based on the Boolean variable. If the Boolean variable is true: all uncovered trips $\in S$ are assigned first in the next iteration, then all other trips $\in S$. After that, all uncovered trips $\notin S$, and finally all covered trips $\notin S$. If the Boolean variable is false, the order is the same except that all uncovered trips $\notin S$ are considered before all covered trips $\in S$. In lines 19 and 20, if not assigning to critical trips, set i to n_c. In lines 21 and 22, remove the assigned combinations from the current solution if not all trips are covered and this is not the last iteration.

4.3.5 Assignment Problem

In line 9 of Algorithm 2, the assignment problem is solved. This problem determines how many train units of a specific type are needed. Therefore, a graph $G(V'', A')$ is build for each train unit

type, $b \in U$, containing all trips that are done by train unit b, with V'' all trips train unit b has to do and A' all arcs between every (i,j)-pair $(i \in U, j \in U)$. If train unit b is assigned more than once for a specific trip, then for every trip that has to be done by train unit b, a new vertex is created. For instance, trip i needs 3 train units of the current type. Then this graph is build with 3 vertices of the current trip. Denote the costs of arc (i,j) $((i,j) \in A')$ as $g(i,j)$. The costs of $g(i,j)$ are set to 0 if trip j can be done after trip i during a day, $(i,j \in T)$. The arc (i,j) is set to 1 otherwise. Denote binary variable y_{ij}, $(i,j \in T)$ which is 1 if arc (i,j) is used and 0 otherwise. To find the minimum train units needed for this problem, we solve the following assignment problem:

$$\textbf{min} \sum_{(i,j)\in A''} g_{ij} y_{ij} \tag{9}$$

$$\textbf{s.t.} \sum_{i \in V''} y_{ij} = 1 \qquad j \in V'' \tag{10}$$

$$\sum_{j \in V''} y_{ij} = 1 \qquad i \in V'' \tag{11}$$

$$y_{ij} \geq 0 \qquad i \in V'', j \in V'' \tag{12}$$

The objective, equation 9, minimize the number of train units needed. Equations 10 and 11 ensure that all trips are entered and left by a train unit once, respectively. Equation 12 ensure that y_{ij} is binary, the binary restriction is not needed because of the constraint matrix being totally unimodular.

4.3.6 Updating sets C and N

Using the three phases, which are described above, the critical trips are considered in the order given by the previous iteration, while the uncritical trips are ordered based on their departure time. In addition, the set of critical trips, set C, is updated at the end of the constructive phase with all unassigned trips that are not in set C at the moment. The unassigned trips are added to set C after all elements that are already in this set. This is important for the main execution since critical trips are ordered based on the previous iteration.

5 Modifications to the existing heuristic

To improve the existing heuristic, three modifications are tested. The first two are given below and the last one is given in Section 5.1. That one is based on a tabu-search technique described in de Werra and Hertz (1989).

The first modification is changing the way the train units are sorted in line 6 of algorithm 2. In the current heuristic, train units are sorted based on their total seats. In this problem, the goal is to minimize the total costs. The price of a train unit is not only depended on the number of seats, but also depended on, for instance, the length of a specific train unit and the material of a train unit. This can lead to train unit combinations with more seats while it costs less. Therefore, we modify line 6 of Algorithm 2 from sorting all train unit combinations based on their total seats to sorting all train unit combinations based on their total costs.

The second modification is using a third element that is computed before: the assignment of the train units of the trips in set S (Equations (5)-(8)). The lower bound can only be found as solution to the original problem if the assignment of train units of the trips belonging to set S are fixed to that solution. Therefore, all trips that do not belong to set S are put initially in the set of uncritical trips, and the trips that do belong to set S are assigned to the found solution (and are not put in the set of critical trips). This means that set C is initialized as: $C = \{\emptyset\}$. The assignment of all other trips are then built around the assignment of trips in set S. Furthermore, line 18 of algorithm 2 needs a change with this modification and is therefore modified to: randomize the order for the next iteration and assign all trips $\in$ *uncov* before the other trips.

5.1 Updating sets C and N with tabu search

The third modification is the way sets C and N are updated, with the use of the second modification. In particular, this modification is extended. This extended use a tabu search method. Initialize set *tabu*, which contains all trips that are tabu to become critical again. Trips that are in set *tabu* can not become a critical trip for the current and two additional iterations. This will prevent trips becoming critical and uncritical, alternating. We use a combination of removing trips being critical and a tabu search technique to prevent a large amount of trips being critical: being critical should be something more special, rather than being assigned first. Trips are removed from set C if they are not left uncovered after the constructive phase. With the use of this tabu search technique, all trips get a chance to become critical. A maximum of $|S|$ trips are in set C during the whole heuristic. To do this, we use algorithm 3 including a tabu search technique.

Algorithm 3 Update sets C and N with tabu search

```
1:  Update tabu
2:  newC ← {∅}
3:  for all trips t ∈ C do
4:      if t ∈ uncov then
5:          Add trip t to newC and remove trip t from uncov
6:      else
7:          Add trip to t to tabu and add trip t to N
8:  for all trips t ∈ uncov \ tabu do
9:      if |newC| ≥ |S| then
10:         break
11:     Add trip t to newC
12: C ← newC
```

In line 1, update *tabu*: All trips that are in *tabu* for 3 iterations are removed from *tabu*. In line 2, the set *newC* is initialized empty. In lines 3 till 7, all trips in the current set C are being visited. All trips that remain uncovered are put in *newC* and removed from *uncov*. All covered trips are added to *tabu* for the current and 2 additional iterations and to N. In lines 8 till 11, if *newC* is full, the forloop stops and every uncovered trips t, $t \notin tabu$, are added to *newC*. In line 12, C becomes the temporarily set *newC*.

6 Results

The results about the used instances are given at first in Section 6.1. After that, the results obtained about the maximum-weight stable set and the lower bound are presented in Section 6.2. After that, the results about the original heuristic (Section 6.3) and the modifications to the heuristic (Section 6.4) are presented. Then, we present some information about the varying costs during each iterations in Section 6.5. We end the results with the different train unit combination that form the best solution in Section 6.6 of all methods.

6.1 Instances

The railways have three different type of trains. Each with different characteristics, which are shown in Table 1. The costs per seat are the cheapest for type OC (€460 costs/seats) and the most expensive for type OH (€527,78 costs/seats). The costs per meter train are also the cheapest for type OC (€2300 costs/meter), but the most expensive for type OT (€2640 costs/meter). However, OT has the most seats per meter train: 5,12 seats per meter, while OH has the least: 4,8 seats per meter.

Table 1: All available train unit types with their characteristics

Train unit type	Costs/year (€)	Seats	Length (meters)
OC	230,000	500	100
OH	190,000	360	75
OT	330,000	640	125

The railways have 12 different lines in which train units will be assigned to. Each line has a different maximum length in meters. The lowest maximum length is 200 meter, while the highest maximum length is 400 meter. This means that a maximum of 5 train units can be combined to perform a trip with a maximum length of 400 meter consisting of either 5 OH train units or 4 OH and 1 OC train units.

In the provided data set, we are given four instances. Each instance contains the following information about each trip: The number of trip, the departure and arrival times, the starting- and end locations, the demand μ and σ, and the line. We are given two different types of demand. Demand μ is the minimum amount of seats that should be available during each trip and demand σ can be added to demand μ to guarantee that all passengers can sit on a seat during each trip. However, demand σ is optional and only satisfying passenger demand μ is sufficient. Furthermore, the 4 instances have 200, 318, 564 and 1010 trips respectively and the first train will depart at 5:02 and the latest will arrive at 13:05. Moreover, we are given 13 different stations where trains can start and end.

6.2 Maximum-weight stable set and lower bound

Table 2 shows the minimum flowing leaving the source for all different instances and the number of trips that build the maximum-weight stable set (set S). The flow corresponds with seats: 1 flow is 1 seat. This minimum flow is therefore the minimum required seats. In addition to this, the time to solve for each instance is given. This problem is solved using CPLEX.

Table 2: Results about the maximum-weight stable set

Instance	Flow	#Trips	Time (sec)
1	95,295	110	0.00
2	98,815	115	0.06
3	116,039	136	0.22
4	271,100	349	0.84

Table 3 shows the total costs for each instance belonging to the trips that build the maximum-weight stable set. For all instances, the required seats in the lower bound and the computation times are given. This problem is solved using AIMMS.

Table 3: Results about the lower bound

Instance	Costs (€)	Seats	Time (sec)
1	50,910,000	101.520	0.27
2	52,780,000	105.320	0.39
3	61,780,000	123.320	0.56
4	144,800,000	289.900	1.02

The maximum-weight stable sets are build between 24.11% and 55% of all trips in the original problem. The increase in seats between the flow and the lower bound is always between 6 and 7%. The computational times are low, but increasing if the number of trips increase.

6.3 Heuristic

The heuristic solved the problem for each instance and those results can be found in table 4. The best found solution is given in costs and seats. Also, the computational times are given. Some instances ran out of time as the heuristic needed more than 12 hours to solve it. Those instances were forced to terminate when their iteration was done. The computational time is given in bold, if was forced to terminate. Moreover, the gap in costs and seats are given to compare with the found lower bound. The difference between costs and seats is that those numbers are not necessarily the same, because the costs per seats, when only comparing costs with seats, is not the same for each train unit type as described in Section 6.1. Furthermore, this problem is solved using CPLEX.

Table 4: Results about the heuristic without a modification

Instance	Costs (€)	#Seats	Time (hours)	n_{iter}	Gap costs	Gap seats
1	56,470,000	112,800	3.23	20	9.85%	10.00%
2	62,650,000	124,840	8.03	20	15.75%	15.63%
3	79,540,000	158,900	**12.81**	9	22.33%	22.39%
4	178,370,000	357,080	**14.43**	2	18.82%	18.81%

None of the instances found their corresponding lower bound. Furthermore, instances 3 and 4 ran out of time, more details about the consequences of that can be found in Section 6.5. The gap in costs and seats are increasing for the first 3 instance and are decreased for the last instance. This might happen, because the minimum number of seats is a bit low for instance 3 compared with instance 4 in the lower bound. However, a big gap is used for

The computational times increase exponentially when the number of trips increase. The last iteration of instance 4 was **at least** over 2.5 hours, but seems to be more than 7 hours if each iteration has the same computation time. That is at least 2 times the computation time of instance 1. The computation time increase can be explained by the fact that the last instance contains more trips in set S than trips in the first instance. All trips in set S are also in set C, which has a much longer computational time than set N, because it assigns all trips up to 20 times during 1 iteration of the constructive phase. Furthermore, the computational time of the assignment problem increases by a lot if more trips needs to be assigned.

6.4 Modifications to the existing heuristic

Tables 5, 6 and 7 contain the information about the outcomes about the 3 modifications that are made to the existing heuristic. The tables contain the same information as that of the heuristic,

including the time given in bold if it does run out of time. In addition, when an instance found a better solution than the heuristic without modifications, then the gap costs and seats are given in bold.

Table 5: Results about the heuristic with sorting on seats replaced by sorting on costs

Instance	Costs (€)	#Seats	Time (hours)	n_{iter}	Gap costs	Gap seats
1	58,840,000	117,740	16.34	20	13.48%	13.78%
2	64,330,000	128,280	**12.44**	19	17.95%	17.90%
3	79,870,000	159,160	**16.09**	6	22.65%	22.52%
4	198,450,000	397,480	**29.24**	2	27.03%	27.07%

Sorting the train unit combinations based on their costs instead of their seats lead to worse results than the original heuristic. In particular, the costs are increased between 0.41% and 11.26% when compared with the original heuristic. Also, the computational time is increased by a lot. Instance 4 was more than twice as slow using this modification. This improvement is therefore not successful.

Table 6: Results about the heuristic with set S being fixed

Instance	Costs (€)	#Seats	Time (hours)	n_{iter}	Gap costs	Gap seats
1	54,470,000	108,660	1.17	20	**6.54%**	**6.57%**
2	62,010,000	123,420	5.36	20	**14.88%**	**14.67%**
3	82,020,000	163,880	**12.43**	13	24.68%	24.75%
4	173,230,000	344,280	**18.95**	3	**16.41%**	**15.80%**

Fixing set S allows the heuristic to become more quickly, which can easily be seen when comparing the computational times with the original heuristic. Furthermore, only instance 3 did not manage to find a better solution with this modification than the heuristic. This method has costs which are around 3% lower than the costs of the normal heuristic. Instance 3 has costs which are 3% higher. Therefore, this heuristic is good to use, especially for bigger instances. Even when the computational time of instance 4 is larger than the normal heuristic, this method is able to get an additional iteration.

Table 7: Results about the heuristic with set S being fixed and the use of tabu search

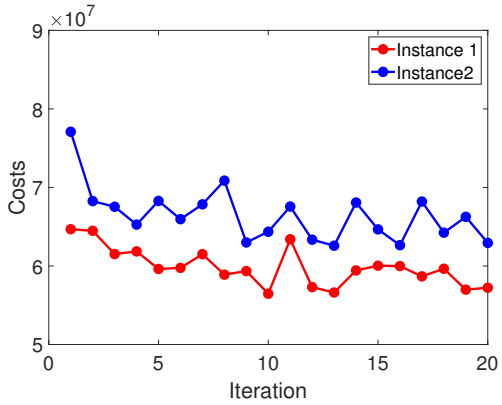
Instance	Costs (€)	#Seats	Time (hours)	n_{iter}	Gap costs	Gap seats
1	54,950,000	109,520	0.48	20	**7.35%**	**7.30%**
2	62,510,000	124,560	1.85	20	**15.57%**	**15.45%**
3	79,890,000	159,820	8.11	20	22.67%	22.84%
4	169,750,000	339,720	**17.22**	3	**14.70%**	**14.67%**

The results of using tabu search are a bit worse for small instances when compared without using tabu search. Those results, however, are still slightly better than the results of the original heuristic. For bigger instances, tabu search turned out to get better results than without tabu search, including much shorter computational times. Tabu search is therefore also a lot faster than the original heuristic. Furthermore, the costs of instance 4 are much better than the original heuristic (4.83% lower). Only instance 3 did not get a better result, however it found a similar solution. Therefore, this heuristic is good, because it is faster while also getting better results.

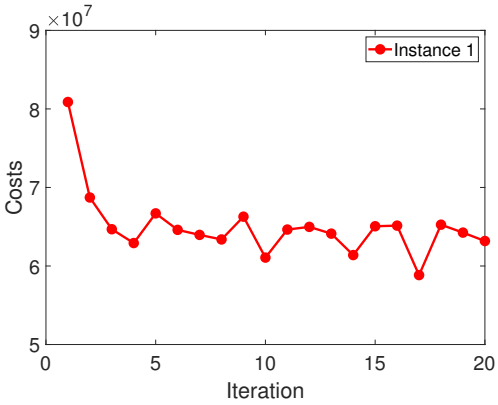
The difference between the gap in costs and seats do not differ a lot, meaning that if the costs are close to value of the lower bound, then the seats are also close to its minimum value. Moreover, none of the two gaps is always lower than the other. This also holds for the heuristic in which sorting on seats is replaced by sorting on costs.

6.5 Costs during each iteration

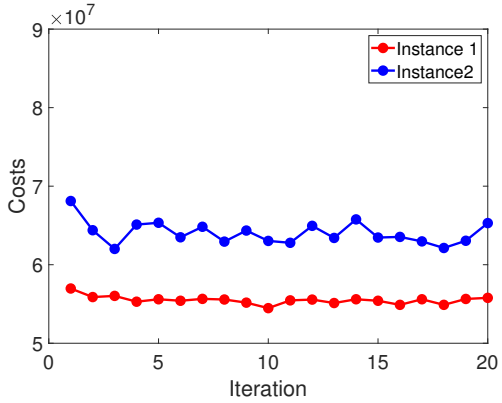
Some instances ran out of time. To find out if that is a big problem, we consider all instances that did not run out of time. The graphs of the original heuristic and 3 modifications can be found below, in Figure 1.



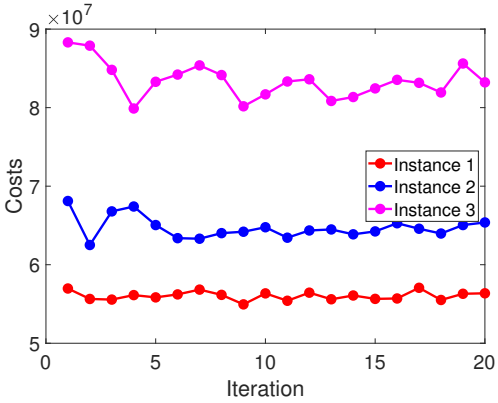
(a) Costs for each iteration for the original heuristic



(b) Costs for each iteration for modification 1



(c) Costs for each iteration for modification 2



(d) Costs for each iteration for modification 3

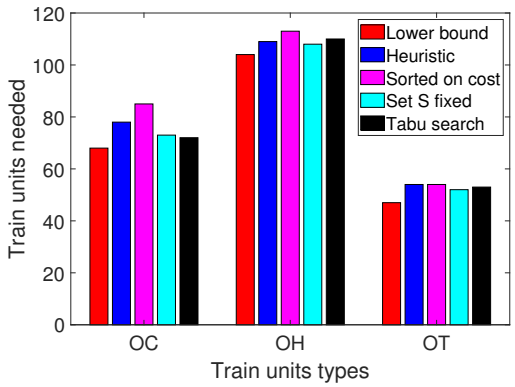
Figure 1: Costs after each iteration

The standard heuristic had its minimum at iteration 10 and 13 for instances 1 and 2 respectively. Therefore, the heuristic is likely to get better results for instances 3 and 4 if the heuristic would run longer. It is therefore somewhat problematic that it did not run more iterations. For the first modification, instance 1 found its minimum at iteration 17. Since instance 2 has 19 iterations, it is likely that it found a minimum. However, instances 3 and 4 are likely to find a combination with lower costs if they run longer. Modification 2 found the minimum costs at iteration 10 and 3 for, respectively, instance 1 and 2. Therefore, instance 3 did find its lowest costs, but instance 4 could possibly find lower costs if it performed some more iterations. The last modification, only did not finish all iterations for instance 4. Instances 1, 2 and 3 found their minimum costs at iteration 9, 2 and 4 respectively. If this heuristic did at least 10 iterations, it is likely to find a better minimum.

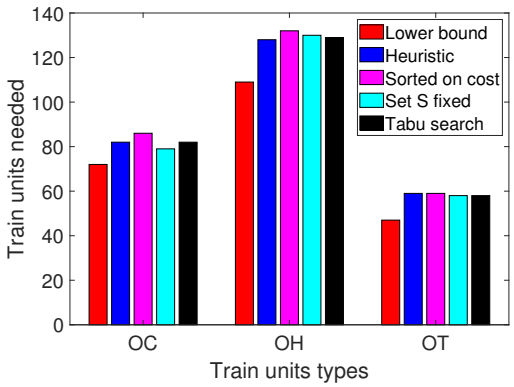
In all instances of fixing set S with and without using tabu search, the costs stabilize at a higher costs than the minimum, after finding its minimum. This is only compared with instances 1 and 2 of the original heuristic and the first modification. The other two heuristics fluctuate a lot, even after finding a minimum.

6.6 Train Unit Combinations

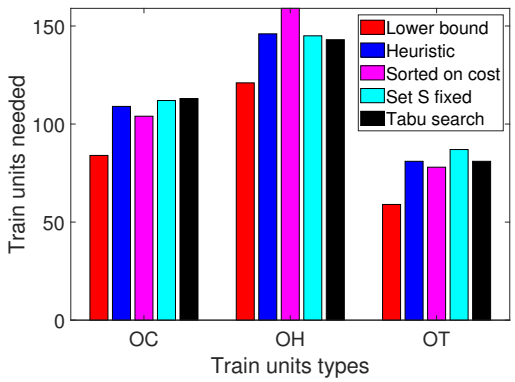
Some train units are more preferred than others. To find out if all train units are used and if one is preferred over others, we consider the combinations of the best results. The results about the used train units can be found below, in Figure 2. The figures contain the combination for the lower bound, the original heuristic, the modification with sorting on costs, the modification with set S fixed and the modification with set S fixed and tabu search method.



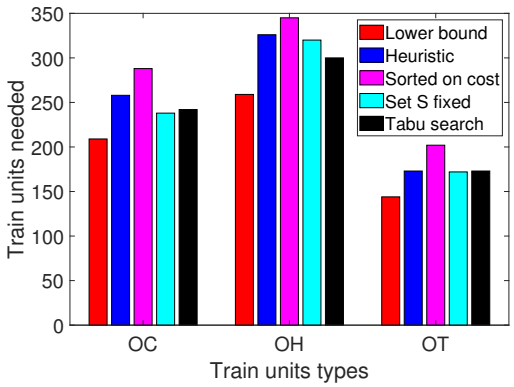
(a) Different combinations for instance 1



(b) Different combinations for instance 2



(c) Different combinations for instance 3



(d) Different combinations for instance 4

Figure 2: Train unit combinations for the best found solution

Train unit type OH seems to be the most used train unit. This seems logical since the lower bound also use that type the most. Furthermore, this is the sequence in increasing costs, length and seats. Therefore, smaller trains with less seats are preferred over longer trains with more seats. Another noticeable thing is that the distribution among all train unit types are pretty similar for each instance. That distribution has the same pattern for the lower bound, the heuristic and all modifications to the heuristic.

7 Conclusion and discussion

We are interested in a method that provides the assignment of train unis with the lowest possible costs. This assignment should have enough seats for each trip, but the combination of train units must not exceed the maximum platform length. Different train units can be bought with different characteristics, which can all be combined to perform one trip. This problem, a TUAP, is solved based on earlier research about this topic. We also tried to improve the existing heuristic.

The existing heuristic found an assignment for only 1 instance that is never improved by any modification. Moreover, those improved results are obtained much faster than the results obtained using the original heuristic if at least set S is fixed. The costs are reduced by almost 5% for the biggest instance, with the use of tabu search. Therefore, a modification with at least set S fixed is recommended to use. It is much better for bigger instances in combination with tabu search: lower costs, while also being much faster than when set S is fixed only. Therefore, using set S fixed in combination with tabu search is very effective for real-world instances with over thousand trips because it is fast and has very good results.

Concluding, for small instances ($<$500 trips), the heuristic with only set S fixed is recommended, while it is recommended to also use tabu search for bigger instances ($>$500 trips).

Instance 3 always has gaps over 20%, therefore sorting train units, based on costs or seats, is not effective for that instance. Even instance 4 is able to get a gap below 15%. Consequently, a new method should be developed, which may also give better results for other instances as well.

Further research can be done about the effect of shunting operation which takes more time. That could be beneficial, because a timetables become more robust to some small delays. Research can also be done about the number of iterations a trip can not become critical, as this influence the algorithm. Also, the fixed size of set C can be analyzed: more or less than the cardinality of set S. And, at last for the last modification, the influence of randomizing the order in which trips can be assigned to set C instead of departure time can be analyzed.

References

E. Abbink, M. Fischetti, L. Kroon, G. Timmer, and M. Vromans. Reinventing crew scheduling at netherlands railways. *Interfaces*, 35(5):393–401, 2005.

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

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

V. Cacchiani, A. Caprara, and P. Toth. An effective peak period heuristic for railway rolling stock planning. *Transportation Science*, 2019.

D. de Werra and A. Hertz. Tabu search techniques. *Operations-Research-Spektrum*, 11(3):131–141, 1989.

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

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

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

A Programming code

The zip-file contains 5 folders. I will shortly explain (1 sentence max for each file) what everything does:
Calculating subset contains 5 files. *DirectedGraph* stores the information about the graph that is built. *DirectedGraphArc* stores the information about all arcs in the graph. *Location* stores the information about all trips, including the demand, arrival, departure, from and to locations. *Main* is the main program to execute the code, which needs all input from a specific instance and give the dual values as output. *Model* is the main program for CPLEX to solve the minimum flow problem with demand.
Step 3 without modification also contains 5 files. *DirectedGraph*, *DirectedGraphArc*, *Location* have exactly the same purposes as in "Calculating subset". However, it is used for the assignment problem instead. *Main* is the main program which contains the algorithm for the peak period heuristic, assigning trips and updating sets C and N, which give the results about the trips and costs as output. *Model* is the main program for CPLEX to solve the Assignment Problem.
Main Modification 1, **Main Modification 2** and **Main Modification 3** contain respectively the main programs for the first, second and third modification. The other programs are exactly the same as used in "Step 3 without modification".