Demand-to-Train Allocation in a Hub-and-Spoke Network

Sven van den Berg (343536)

Erasmus School of Economics
Erasmus University Rotterdam

October 3, 2014

Abstract

Several planning and scheduling problems are involved within rail freight transportation planning. One of these problems is the demand-to-train allocation problem, which is the assignment of demands to trains. In this thesis a method is presented to solve this demand-to-train allocation problem for a hub-and-spoke network. The problem is divided into two subproblems: the blocking problem and the block-to-train assignment problem. Hereby, demands are first combined into blocks, after which these blocks are assigned to trains. A heuristic is presented to solve the blocking problem, while we also investigate the case where we do not create blocks and solve the demand-to-train allocation problem directly. The block-to-train assignment problem is formulated as an IP model. For this problem a Lagrangian heuristic and a greedy heuristic are created to solve the problem. The IP model and heuristics were tested on several problem instances with different problem sizes, in case of blocking and in the case where the demand-to-train allocation problem is solved directly. For all problem instances it was better, in terms of both solution value and computation time, to first create blocks instead of solving the problem at once.

Keywords: Railroad scheduling; Freight transportation; Blocking problem; Assignment problem; Integer Programming; Lagrangian Relaxation; Heuristics

Erasmus University Rotterdam
Erasmus School of Economics
Supervisor:
Dr. Dennis Huisman
Coreader:
Dr. Twan Dollevoet

Ab Ovo International
Business Unit Advanced Planning and Scheduling
Supervisors:
Drs. Peter Koot
Drs. Dieter Veldhuis

Contents

1 Introduction

Rail freight transportation is an important mode of freight transport. In rail freight transportation demands are transported from their origin to their destination, by means of a locomotive and a certain number of wagons, over a railroad network. Two other very important modes of freight transport are road transport (small demands and short distance) and sea transport (large demands and large distances). Rail freight transport is, with regard to the demand size and travel distance, in between the other two modes. A major difference between road and/or sea transport and rail transport is that the physical infrastructure is already there (sea transport) or is mainly used by cars (road transport), while in rail transport the infrastructure is especially build for the rail transport itself.

Real-life rail networks are very large, complicated networks where, daily, a large amount of demands is transported from their origin to their destination. This makes the planning of the transport of demands very difficult, but also very important (a good planning can save a lot of money). There are, due to the size of real-life problems, no efficient techniques to find the optimal overall rail freight transportation planning for real-life problems. Heuristics are used to find efficiently very good, near-optimal solutions for the different planning problems involved with rail freight transportation planning.

Rail freight transportation planning has, in general, three main levels: strategic, tactical and operational (Assad [3]). The most important difference between these three main levels is the planning horizon. Strategic decisions are involved with long term transportation planning. The tactical level consists of decisions with medium-term planning horizons, and it focuses especially on the optimal use of resources. Finally, the operational level is associated with short-term decisions. These three different levels are related to each other. To put it simply, the strategic decisions are input for the tactical decisions and the tactical decisions are input for the operational decisions.

As stated earlier, rail freight transportation planning consists of several planning and scheduling problems. The most important planning and scheduling problems for railroads are: empty car distribution problem, the blocking problem, train scheduling problem, block-to-train (BTA) assignment problem, locomotive scheduling problem and crew scheduling problem. The empty car distribution problem ensures that there are always enough wagons available at each location to transport the demands. In this problem empty wagon moves are created that restore the balance of empty wagons at all locations. These empty wagons moves are demands of empty wagons, and thus, treated as demands in the rest of the overall problem. A very important problem for railroads to be solved is the blocking problem. In this problem individual demands are combined into blocks. A *block* is a set of demands which are temporarily joined during their trip between a common origin (block origin) and destination (block destination) (Campbell [6]). Mathematically, the blocking problem is a multicommodity flow, network design and routing problem (Ahuja et al. [2]). In multicommodity network design (MCND) problems, multiple commodities, such as individual demands, must be routed between different points of origin and destination on the available arcs (Yaghini and Akhavan [22]). Next, the train scheduling problem determines the origins, destinations, routes and days of operation for trains. As the blocks and trains are created, the blocks should be assigned to the trains, and this is done in the BTA problem. Then, locomotives (locomotive scheduling problem) and crews (crew scheduling problem) should be assigned to the trains. All these individual problems should be solved in such a way that the costs related to the problems are minimized.

In some real-life rail networks both freight and passenger transport are routed over the same network. In these cases, passenger trains are mostly given priority. This means that the route of a freight train can only be scheduled in such a way that it does not influence the schedule of passenger trains. This is a huge restriction for the train scheduling problem.

This thesis is based on a project of Ab Ovo for a European railway company. The type of the network described in this thesis is based on the network of this European railway company. Passenger trains are also moving on the same network as the freight trains, and therefore, a network with also passenger trains moving on it is considered. It is assumed that a set with possible train schedules which are not in conflict with the passenger trains is known beforehand (for example, using historical data). As a set of trains is known beforehand, the train scheduling problem is not considered in this thesis. The problem discussed in this thesis is to choose a subset of the set with possible train schedules and assign the demands to these trains. This problem is called the demand-to-train allocation problem and is a combination of two of the problems earlier discussed in this chapter: the blocking problem and the BTA problem. We present an exact formulation for the BTA problem and two different heuristics for each of the subproblems. So, first the demands are combined into blocks and then these blocks are assigned to trains from the set with possible train schedules. Then, automatically a subset of the trains is chosen (all trains with demand(s) assigned to it).

The thesis is organized as follows: In Chapter 2 the problem is described in detail. The first part of this chapter is about the rail network and in the second part both subproblems are described. Chapter 3 reviews literature about both subproblems: the blocking problem and the BTA problem. The mathematical model of the BTA problem is described in Chapter 4. For both subproblems a heuristic to solve the problem is presented in Chapter 5. In Chapter 6 the results of the IP formulation and the heuristics are shown and compared. Finally, a conclusion and some points with room for improvement are given in Chapter 7.

2 Problem Description

In this chapter the demand-to-train allocation problem is described. The problem we discuss is based on a project of Ab Ovo for a railway company in Europe, as stated in Chapter 1. Therefore, some details of the problem are especially for the problem of this railway company in Europe. The different types of locations, demands and trains are good examples of details of the problem that are specifically part of this problem.

In the first three sections of this chapter all the components of the network will be described. The fourth section is about the network itself after which the demand-to-train allocation problem is described in the last section. The first three sections are divided as follows: Section 2.1 describes the different types of locations that are present in the network, Section 2.2 describes the different types of demands and in Section 2.3 the different types of trains are described.

2.1 Locations

At locations, wagons can be stored, attached to trains or detached from trains. We distinguish two different types of locations: service locations and base locations. The combination of these two locations can be seen as a hub-and-spoke network. For some types of demands the network is indeed treated as a hub-and-spoke network. This hub-and-spoke network is explained in Section 2.4. The two types of locations are described below:

- Service locations

  Service locations are small locations where the origins and destinations of the demands are. Customers are connected via railroad tracks to service locations. Demands are first shunted from their customers to the origin (service location), after which the demands are further transported via the rail network to their destination (service location). Then, the demands are shunted from the destination to the customer. Industries with customers of a railroad company are, among others: mining, logging, iron ore, and paper. Some examples of customer locations are: paper mills (paper), forests (logging), mines (mining) and ports (container transport). Every service location belongs to exactly one base location.

- Base locations

  Base locations are larger, centralized locations which consist of a lot of railroad tracks next to each other. Base locations are locations where a lot of operations are performed, such as storing wagons (at the railroad tracks) and *reclassification*, which is the reallocation of demands among trains. This is done at the railroad tracks, where the demands are attached to and detached from trains. Every base location is connected with several service locations.

Reclassification at base locations is necessary when, for example, two demands with different destinations are combined on one train. This situation is shown in Figure 1. In this figure a_1 and a_2 are service locations of base location A_{base}, locations B_{base} and C_{base} are two other base locations. At service location a_2 two demands with different destinations, b_1 and c_4, are both loaded on a train and transported to base location A. At the same time a demand for location c_8 is transported by another train from service location a_1 to base location B. In this simple example, the most easy solution is via a reallocation of the demand at base location A. At this location (A) both demands that need to be transported to a service location of C (demands c_4 and c_8) are combined and put on a train to base location C. The demand for service location b_1 is put on a train to base location B, as shown in Figure 1. Now both trains can go immediately to the base location of the destination of all demands loaded on the train.

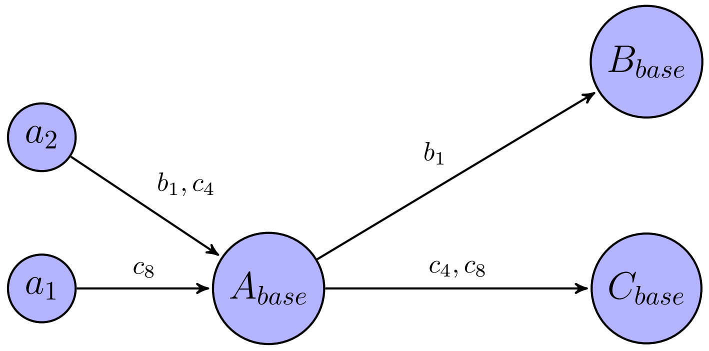


Figure 1: Example of a reallocation of demands among trains at a base location

2.2 Demands

Railway companies make transport agreements with their customers. These agreements consist of one or more subcontracts. In these subcontracts the demand of a certain product, that the customer wants to transport between two fixed locations, is stated. This can be a yearly demand, but it is also possible that in the contract it is stated that the customer wants to transport a certain amount of a product every Tuesday, for example. Individual daily demands are a result of these contracts. A single demand is defined by its origin, its destination, its weight, its length, the number of wagons, a pick up time-slot, a delivery time-slot and the type of demand. The first five characteristics of a demand speak for themselves, but the last three should be explained: A pick-up time slot is a time-slot in which the demand should be picked up at its origin and a delivery time-slot is a time-slot in which the demand should be delivered at its destination. The different types of demand are explained below:

- Customer demand

  These demands are very large, single demands of one customer that are transported directly from their origin to their destination. These demands are so large that, due to costs, they are not combined with other demands on one train. If the number of wagons needed to transport a single demand is above a certain number, the demand is indicated as a customer demand.

- Customer in circular demand

  Like the customer demands, the customer in circular demands are very large, single demands of one customer. The only difference between those two demands is that for customer in circular demand the wagons used to transport the demand should also be transported back to their origin location. The reason that wagons are directly transported back to their origin is that between some pairs of locations there is only transport from one location to another, and not vice versa.

- Wood demand

  Wood demands are also transported directly from their origin to their destination, because special handlings are needed for the transport of wood. Wagons with wood demands are picked up in forests on small, simple railroad tracks and these wagons are delivered at, and can be parked in, a wood factory. There, inside the factory, the trunks are lifted from the wagons. Due to the special handlings involved with the transport of wood demand, these are treated as a separate type. At most two wood demands can be combined on one train, but only if the origin and destination of both demands is the same.

- Liner demand

  In contrast to the other demands, liner demands are not transported directly from their origin to their destination. A liner demand is smaller than a customer demand and can therefore be combined together with other liner demands on one train. This can result in several stops and a reallocation of demand among trains at base locations. It is therefore quite normal that a single liner demand is transported via several trains from its origin to its destination.

- Empty wagons demand

  These demands consist of a number of empty wagons that should be transported from one location to another location. These moves are needed for the re-positioning of wagons. The flows of incoming and outgoing wagons at most locations are not equal. Therefore, empty wagon moves are needed to ensure that locations have enough wagons available to transport the demands.

In this thesis we mainly focus on the demand-to-train allocation problem for the liner and empty demands, as for these demand types it is allowed to assign more than just one or two demands to a single train. This makes the problem for these two demand types the most difficult, and thus, the most interesting to investigate.

2.3 Trains

In this thesis a train is a scheduled route where several wagons can be moved by a locomotive. A railway company should apply for a train schedule to the infrastructure manager. The infrastructure manager determines whether the train schedule is assigned to the railway company. Passenger trains are mostly given priority over freight trains. Railway companies should apply for train schedules a long time in advance. A company applies for many train schedules, so that, even if a lot of these requests are refused, all demands can be transported from their origin to their destination. This will result in a set of possible time schedules for freight trains that is known beforehand. Train routes are only performed if at least one demand is assigned to a train.

Assigning demands to trains is restricted, because trains have a maximum length, a maximum weight and a maximum number of wagons that can be assigned. Demands can be assigned to trains as long as these bounds are not exceeded, and the trains are in the right time window. For a pick up of a demand at its origin this means that the train should be in the right time window with respect to the pick up time-slot of the demand.

Another characteristic of a train is its type. There are different types of trains, and each of them is able to transport one or more types of demand. The different types of trains are listed and explained below:

- Customer train

  A customer train is a train that is able to transport both types of customer demands. Such a train transports a single customer demand (as explained in Section 2.2) directly from origin to destination. These trains consist of only one (customer) demand.

- Wood train

  Wood trains are only used for the transport of wood demand. These trains also transports the demand(s) directly from origin to destination. In contrast to customer trains, wood trains can consist of more than one demand. At most two demands can be combined on one wood train.

- Liner train

  These trains are used for the transport of liner demand and empty wagon demand. They pick up the demands from service locations, which belong to one base location, and transport these demands to their base location. Liner trains are also used to deliver the demands, which arrived at a base location, to its service locations. In short, liner trains transport demands between a base location and its service locations. Liner demands that originates from service locations are always first transported to their base location, after which the demands can be reclassified at the base location. In contrast to the first two train types, liner trains can consist of many demands, as long as the maximum length, weight and number of wagons are not exceeded.

- Base train

  Base trains are used to transport liner demands and empty wagon demands between base locations. A demand is picked up by a base train at the base location that belongs to its origin, and then further transported, possibly via other base locations and by other base trains, to the base location that belongs to its destination. As with liner trains, base trains can consist of many demands as long as the upper bounds on length, weight and number of wagons are not exceeded.

2.4 Network

The transport of liner demands can be seen as a hub-and-spoke network, where the base locations are the hubs and the service locations as spokes. Transport between hubs and their spokes is done by liner trains and transport between hubs is done by base trains. The flow of all the other demands is very simple, just directly from origin to destination, and is therefore not further explained in this section.

Figure 2 shows the flow of liner demands in a simple hub-and-spoke network with service and base locations as hubs and spokes, respectively. The flow of liner demands from service locations of base location B to their destinations is shown in this example. The liner train starts at service location b_3 and moves via b_2 and b_1 to base location B while picking up the demands at all these locations. At base location B the demands are unloaded and loaded on two different base trains, one base train for the demands (a_1 and a_3) for the service locations of base location A and another for demands c_2 and c_3 for base location C. When these base trains arrive at their destination (A and C), the demands are unloaded and loaded on a liner train that will deliver the demands at their destinations.

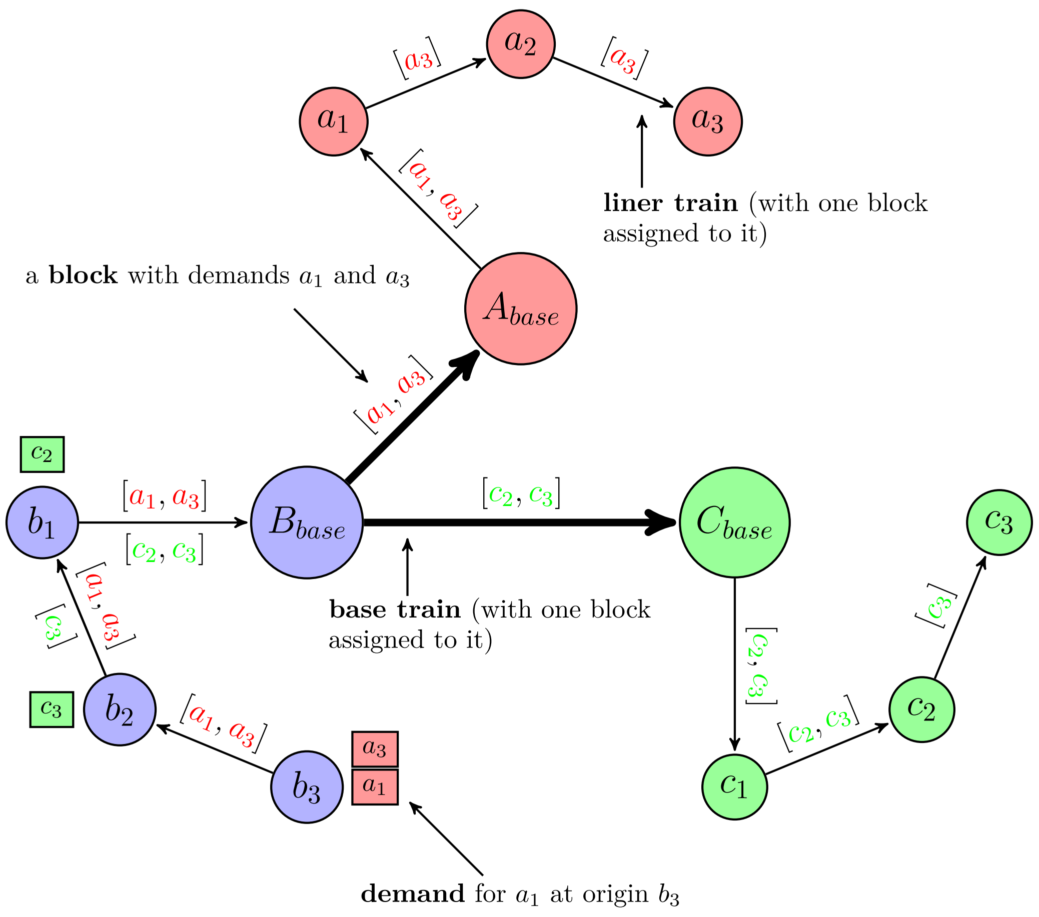


Figure 2: Example of a flow of liner demands through the hub-and-spoke network with base and service locations as hubs and spokes, respectively

2.5 Demand-to-Train Allocation Problem

The goal of the demand-to-train allocation problem is to assign demands to trains in such a way that all demands are transported from their origin to their destination and the total cost are minimized. Costs involved with this problem are, on the one hand, all costs with respect to the transport and handling of demands, and on the other hand, penalty costs if demands arrive too late at their destination. These penalty costs ensure that most of the demands are delivered on time.

As explained in Section 2.2 there are different types of demands, and these demands have to be assigned to trains to ensure that they are transported from their origin to their destination. The problem consists of all demands of an 'average' week; a week that is representative for all the weeks in a year. This week is called a *modelweek*. One week is chosen as the length of the planning period, because train schedules are usually repeated every week (Jha et al. [11]).

The decisions that should be taken for the demand-to-train allocation problem can be divided into two subproblems: (i) combining demands into blocks (blocking problem) and (ii) choose a subset of all possible trains and assign blocks to these trains (BTA problem). These two subproblems are described in the next two subsections.

2.5.1 Blocking Problem

In the blocking problem demands are combined into blocks. The idea behind combining demands into blocks is that in the second subproblem there will be less possibilities to assign the demands to the trains. This will decrease the computation time for the block-to-train assignment problem. On the other hand, it is likely that the solution is worse as the solution space is limited. Here it is important to find a good balance between the computation time and the quality of the solution. This balance will be different for every problem. For small problems the computation time is no problem and individual demands can be treated as a block of one demand, while for very large problems many blocks are needed to ensure that the computation time is acceptable.

As explained in Section 2.2 the problem we discuss has large demands (customer demands) that cannot be combined with other demands. This means that these demands are a block, consisting of one demand. The other types of demand can be combined into blocks. These demands should be combined into blocks in such a way that the total block distance, which is the travel distance from the origin to the destination of a block, and the number of reclassifications, which is the number of times a demands is changed from block, is minimized.

It is not possible to assign all demands to one block, as blocks have a maximum weight, maximum length and a maximum number of wagons that can be assigned. Demands can be assigned to blocks as long as these bounds are not exceeded. Each block has an origin and a destination, and all demands in a block are transported from the origin to the destination of that block. At the destination of a block the demands can be reclassified into new blocks and transported further. In the second subproblem these blocks will be assigned to trains.

See Figure 2 for an example of a block; demands a_1 and a_3 are combined into a block with origin b_3 and destination a_1. At location a_1 demand a_1 is unloaded and demand a_3 is reclassified into a new block from a_1 to a_3, where it is delivered. The number of reclassifications of demand a_1 is zero, as it is only assigned to one block (from its origin to its destination). Demand a_3 is reclassified once, namely at location a_1 where it is reclassified into a new block. The other two demands shown in the example (c_2 and c_3) are together in a block from b_1 to c_2. Demand c_3 is also assigned to a block from b_2 to b_1 and to a block from c_2 to c_3. This demand is reclassified twice, at b_1 and c_2.

2.5.2 Block-to-Train Assignment Problem

As the blocks are created, they should be assigned to trains. This is done in the block-to-train assignment (BTA) problem. As stated in Section 2.3, the possible schedules of the trains are known beforehand. If all the blocks are assigned to trains, we know which trains are chosen (the trains with blocks assigned to it) and which trains are not chosen (the trains without blocks assigned to it). The blocks should be assigned to trains in such a way that a certain cost function is minimized. Costs involved with this cost function are, among others: fixed costs for using a train, variable costs for the transport of the blocks of demands and (penalty) costs when a demand arrives too late at its destination.

To illustrate this problem with an example, take again a look at Figure 2. Five trains (three liner trains and two base trains) are chosen and blocks $[a_1, a_3]$, $[c_3]$ and $[c_2, c_3]$ are assigned to the liner train from the service locations of B to base location B, block $[a_1, a_3]$ is assigned to the base train from B to A, etcetera. When solving the BTA problem the blocks are given (created in the blocking problem), and these should be assigned to trains. In the example of Figure 2 it was in the BTA problem, for example, also possible to assign block $[a_1, a_3]$ to the base train from B to C and to a base train between C and A. Then, the base train from B to A is not chosen. These kind of decisions are involved with the BTA problem.

3 Literature Review

From the previous chapters it has become clear that this thesis deals with a demand-to-train allocation problem, which consists of two subproblems: combining demands into blocks and assigning these blocks to trains. In the literature most papers focuses only on one of the subproblems. Therefore, the literature of both subproblems is discussed separately. Section 3.1 reviews literature about blocking problems. In Section 3.2 we discuss some papers about BTA problems.

3.1 Blocking Problems

One of the first papers with a model for blocking problems was Bodin et al. [5], in 1980. They formulated the problem as a nonlinear, mixed-integer programming (MIP) model. The model is a multicommodity flow problem with additional side constraints, such as yard and block capacity constraints. In 1986, Van Dyke [21] presented a heuristic to improve an existing blocking plan. The heuristic is based on an iterative procedure that tries to improve the current blocking plan by solving a series of shortest-path problems on a network where the arcs represent blocks. In Keaton [12] and Keaton [13] a Lagrangian relaxation approach is presented to solve a combined problem of blocking, train scheduling and block-to-train assignment. The problem is solved very efficiently when considering no limits on the train size, but then the solution will consist of overloaded trains. Afterwards, heuristic adjustments are needed to get a feasible solution. When taking into account the limits on train size, Keaton found out that it was impossible to find good lower bounds with their approach. A simulated annealing approach is presented in Huntley et al. [10], in 1995, to solve the blocking problem. Newton [18] models the blocking problem as a directed network budget design problem with constraints on yard and block capacity, and restrictions on legal blocking paths. He indicates paths as legal paths if the number of handlings on a path does not exceed a certain upper bound and if the path is in the set with all possible routings, for a certain commodity. A branch-and-price algorithm is presented to solve this problem. In Newton et al. [19] a branch-and-price algorithm is developed to solve the blocking problem. New paths are generated by solving a shortest path problem. The model consists of the same type of constraints as in Newton [18]. In 1998, Gorman [8] developed a genetic search algorithm and a tabu-enhanced genetic search to construct operating plans, including, among others, both blocking and block-to-train assignment. The tabu-enhanced genetic search performed clearly better than the genetic search algorithm. In Kwon et al. [14] a method is developed to improve an existing blocking plan and block-to-train assignment. They use a column generation approach to solve the problem. Barnhart et al. [4] use a Lagrangian heuristic to solve the blocking problem. The problem is divided into two subproblems, whereby the storage requirement and computational effort are reduced. In contrast to a lot of other papers Ahuja et al. [2] presented an arc-based (instead of path-based) multicommodity network design (MCND) model. The problem is solved with a very large-scale neighborhood (VLSN) search algorithm. The results of this algorithm were great when they applied it to data from several major railroads. In 2011, Yue et al. [25] presented an Ant Colony algorithm to solve the blocking problem. In the same year, Yaghini et al. [23] also proposed an Ant Colony algorithm. The objective function of the model of Yaghini et al. [23] only consists of variable cost for the flow of the demands, while Yue et al. [25] also takes into account reclassification cost. The difference between these two models in terms of constraints is that Yaghini et al. [23] has constraints on the capacity of a block and the number of blocks that can originate at a location, and Yue et al. [25] not. Yue et al. [25] has one constraint that is not in the model of Yaghini et al. [23], and this is a constraint on the capacity, in wagons, of a railroad line. Both models have a constraint on the reclassification capacity at a location. In 2012, Yaghini et al. [24] proposed a genetic algorithm to solve the blocking problem. They used the same model as Yaghini et al. [23].

This review of the literature of blocking problems shows that, since 1980, researchers presented a lot of different approaches (Lagrangian relaxation, branch-and-price, genetic search, column generation, VLNS, etcetera) to solve the blocking problem. It also shows that there are several objectives (variable flow cost and classification cost) and constraints (block capacity, yard capacity, maximum number of reclassifications at a location, etcetera) for this problem, which are mostly not combined altogether in a model. Most models consist of both objectives and just one, two or three (capacity) constraints.

3.2 Block-to-Train Assignment Problems

The BTA problem is mostly combined with the train scheduling problem (determine routing and frequency of trains), often referred to as routing and makeup models. As one of the tasks of these models is to assign blocks to trains, these models need the blocking policy as input. First, papers that consider the problem separately are discussed. Then, also papers that consider the BTA problem in a combined problem, are discussed.

One of the papers which discussed the BTA problem separately is Nozick and Morlok [20]. The objective of the model consists of variable cost for the movement of the demands and some cost for repositioning. A heuristic was developed to solve the model. The heuristic is an iterative procedure where the LP relaxation of the model is solved and thereafter the fractional values are rounded. Another paper that discussed the BTA problem separately is Jha et al. [11]. In Jha et al. [11] a space-time network, which was developed by Ahuja et al. [1] for locomotive scheduling on a train network, is described over which the BTA is formulated as a flow problem. Two multicommodity network flow formulations are provided for the BTA problem: an arc-based and a path-based formulation. Jha et al. [11] stated that the path based formulation is better than the arc based formulation. They use a planning period of one single day, and the model consists of only one capacity constraint, on the maximum number of wagons that can be assigned to a train. For the path-based formulation all paths are enumerated beforehand. To restrict the total number of paths for a block, the length of a valid path between an origin and destination is limited to a specific percentage of the shortest path between that origin and destination. A greedy heuristic and a Lagrangian heuristic are used to solve the path-based formulation of the problem.

Next, some papers with the BTA problem within a combined overall problem (mostly makeup and routing problems), are discussed. In 1986, Crainic and Rousseau [7] proposed a multicommodity service network design model for the makeup and routing problem. A column generation approach is proposed to solve the model. Haghani [9] proposed, in 1989, a formulation and a heuristic for a combined problem, consisting of train routing, makeup and empty car distribution. As stated in the previous section, Keaton [12] and Keaton [13] presented a Lagrangian heuristic to solve a combined problem with, among others, block-to-train assignment. Marin and Salmerón [15] and Marin and Salmerón [16] proposed and analyzed three heuristics, including simulated annealing and tabu search, for the routing and makeup problem. Capacity constraints on both trains and yards were considered. The simulated annealing heuristic turned out to give the best results, but required more time to solve the problem than the other heuristics. Gorman [8] developed a genetic search algorithm and a tabu-enhanced genetic search for a combined problem with, among others, block-to-train assignment and Kwon et al. [14] developed a column generation approach to improve an existing blocking plan and block-to-train assignment, as stated in the previous section. In the paper of Gorman [8], the tabu-enhanced genetic search performed clearly better than the genetic search algorithm. Newman and Yano [17] proposed a heuristic, based on Lagrangian relaxation and Benders decomposition, to solve the problem.

The literature of BTA problems consists of two different types of papers: Papers that consider the problem separately and papers that consider the BTA problem in a combined problem. Most of the papers in the literature consider the BTA problem in a combined problem. Mostly the BTA problem is combined in routing and makeup problems, but sometimes BTA problems are also combined with, among others, the empty car distribution and the blocking problem. As with the literature of blocking problems, the literature of BTA problems also shows several different approaches to solve the problem. Among others, Lagrangian relaxation, column generation, simulated annealing, tabu search and genetic search approaches are presented in the papers.

For the block-to-train assignment problem we decided to base it on the paper of Jha et al. [11]. One of the reasons to choose for this paper is that their space-time network is created in an efficient way and makes it easy to model the problem as an IP formulation. We also choose for a path-based formulation, as they stated that the path-based is better than the arc-based formulation. Other reasons to choose for this paper are that the algorithm of this paper was tested on large problem instances and showed good results, and it is a clear approach which makes it easier to implement.

4 Mathematical Model: Block-to-Train Assignment

In this chapter the IP formulation of the BTA problem is presented. First, in Subsection 4.1, a space-time network is described over which the BTA problem can be formulated as a flow problem. This space-time network is based on the space-time network from Ahuja et al. [1] and Jha et al. [11]. Thereafter, an IP formulation, based on the IP formulation of Jha et al. [11], is presented in Subsection 4.2. This formulation is path-based. In Subsection 4.3 an algorithm is described to enumerate paths for each block.

4.1 Space-Time Network

The space-time network presented in this subsection is based on the space-time network from Ahuja et al. [1] and Jha et al. [11]. In this thesis the same terminology, as used in both papers, is used to describe the space-time network.

The space-time network is denoted as $G = (N, A)$, where N are the nodes and A the arcs. All nodes correspond to a combination of a train, a location on the route of the train and the arrival or departure time of the train at that location. All arcs correspond to a possible flow of blocks (blocks can be assigned to arcs). In this network the route of a train consists of several trips. A *trip* is the movement of a train between two consecutive locations at which the train stops and where demands can be loaded and unloaded. In the space-time network a trip is represented by a directed arc (*train-arc*) from a *train-departure node*, which corresponds to the departure of the train at the origin location of the trip, to a *train-arrival node*, which corresponds to the arrival of the train at the destination location of the trip. A graphical representation of a trip in the space-time network is shown in Figure 3. If a block is assigned to a train-arc, this means that the block is transported by this train from the departure of the trip to the arrival of the trip. Blocks can be assigned to a train as long as the maximum number of wagons, weight capacity and length capacity of the train, corresponding to the train-arc, are not exceeded.

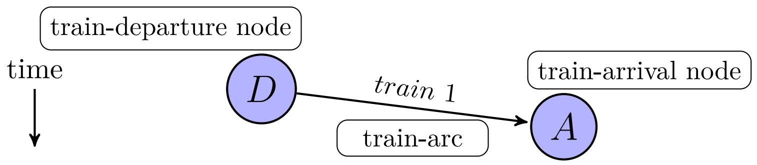


Figure 3: Graphical representation of a trip

So far, only the nodes and arcs of individual trips are explained. A train route consists of several trips, and therefore, consecutive trips should be linked to each other. This is done by *connection arcs.* A connection arc is a directed arc from the train-arrival node of a certain train at a certain location to the train-departure node of the same train at the same location. An example is shown in Figure 4. In this example train 1 departs from location A, stops at location B and moves further to location C. The arc between the arrival (B_{arr}) at location B and the departure (B_{dep}) from location B is a connection arc, and ensures that both trips (A to B and B to C) are connected. If a block is assigned to a connection arc, it means that the block is not unloaded at this location and stays on the train. As with train-arcs, blocks can be assigned to connection arcs as long as the three different capacities of the train, corresponding to the connection arc, are not exceeded.

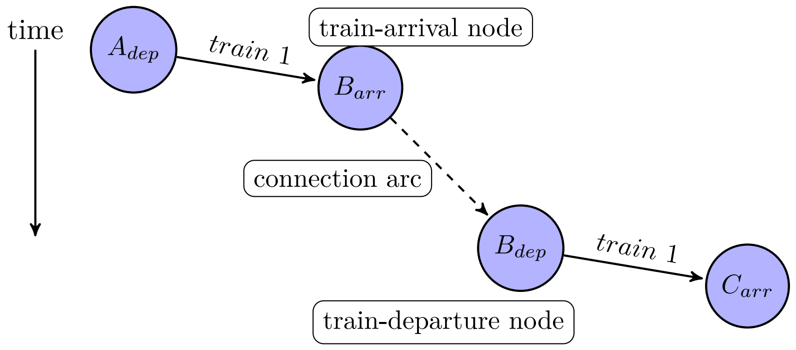


Figure 4: Graphical representation of a connection arc

The space-time network is still not complete, because there are no nodes and arcs for the loading and unloading of blocks. Therefore, a new set of nodes is used, called *ground nodes.* For each train-arrival node a *ground-arrival node* is created with the same location and train attributes. The time at this node is the time at which the unloading activities are finished. The train-arrival node is then connected with a directed arc to the ground-arrival node. This arc is called an *arrival-connection arc.* For each train-departure node a *ground-departure node* is created with the same location and train attributes. The time at this node is the time at which the loading activities start. The ground-departure node is then connected with a directed arc to the train-departure node. This arc is called a *departure-connection arc.* A graphical representation of these nodes and arcs is shown in Figure 5, where A en D are the train-arrival and train-departure nodes of the same train (and thus also connected with a connection arc).

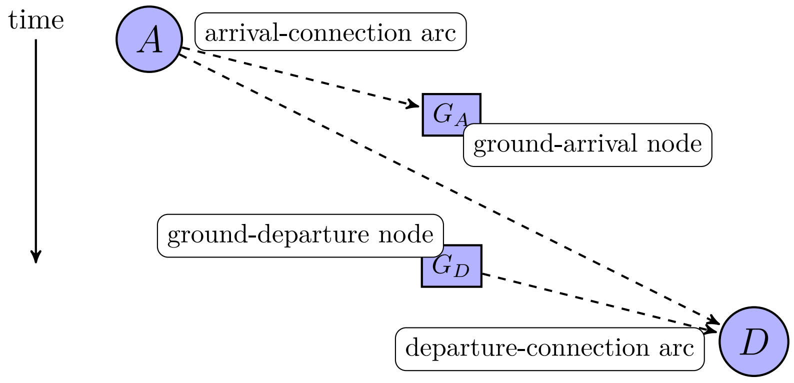


Figure 5: Graphical representation of a ground-arrival and ground-departure node, and an arrival-connection and a departure-connection arc

With these nodes and arcs explained so far, it is not possible for blocks to change trains. Therefore, the ground nodes of a certain location should be connected with each other. These arcs are called *ground arcs*. A ground node is connected via a ground arc with the ground node corresponding to the next arrival or departure, at the same location, in time. So, all ground nodes of a certain location are ordered on their time attribute and then each ground node is connected with the next node. The last ground node of a certain location is connected with the first ground node of that location, to allow a block to be assigned to a train in the next period. If a block is assigned to a ground arc, this means that the block stays at the location during the time between the two ground nodes. In Figure 6 an example of a part of the space-time network is shown (with the ground arcs included).

In the example in Figure 6 the incoming and outgoing trains of a certain location are shown. First, train 1 arrives at the location and the blocks assigned to the train can be unloaded via the arrival-connection arc or stay on the train via the train-connecting arc. The next three events at this location are: the departure of train 1, the arrival of train 2 and thereafter the arrival of train 3. For train 3 this location is its destination, whereby it has no train-connecting arc at this location. The next event is the departure of train 2. Blocks that were unloaded from train 1 can be assigned to this train via the ground arcs between the arrival of train 1 and the departure of train 2. The last events are the arrival and departure of train 4 and the departure of train 5. The origin of train 5 is this location, whereby there is no train-arrival node for train 5 at this location, and thus, also no train-

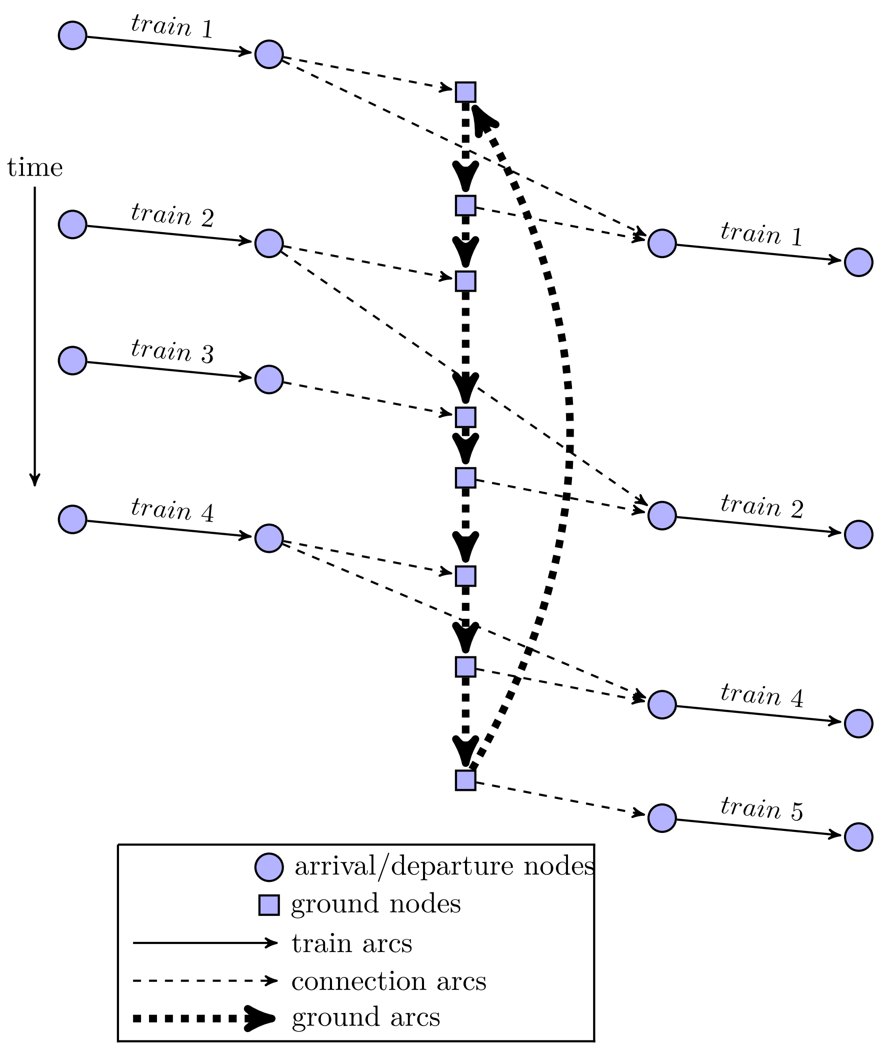


Figure 6: Example of a part of the space-time network

connecting arc of train 5 at this location. The curved ground arc between the departure of train 5 and the arrival of train 1 allows a block to take a train in the next period.

As the space-time network is described, the BTA problem with respect to this space-time network can be defined. The idea of the space-time network with respect to the BTA problem is that in this network the blocks are transported from an *origin-node* to a *destination-node* by means of trains.

Therefore, the origin-nodes and destination-nodes of a block should be defined. The ground-departure nodes at the origin location of a block can be chosen as origin-nodes, as at these nodes the decision is made whether the block is assigned to that train or not. However, it is not a good idea to take a ground-departure node, corresponding to a train-departure on Friday, as an origin-node for a block if the block is released four days earlier on Monday, because this means that the block has to wait at least four days at the location before it is assigned to the next train. Therefore we introduce a *maximum waiting time*. Then, all ground-departure nodes, at the origin location of a block, that are at most the maximum waiting time later than the release time of the block, are the origin-nodes for that block. For example, if the maximum waiting time is one day and the release time of a block at a certain location is Monday at 10.00, then all ground-departure nodes at this location between Monday 10.00 and Tuesday 10.00 are possible origin-nodes.

The destination-nodes of a block are all ground-arrival nodes at the destination location of the block, as we do not know beforehand with which train the block will arrive at its destination. For example, a block has as destination the location whose part of the space-time network is shown in Figure 6. Then, the destination-node set of that block are all the ground-arrival nodes, which are in this example the ground-arrival nodes of trains 1, 2, 3 and 4. The BTA problem is then to find a path from an origin-node of a block to a destination-node of that block.

4.2 IP Formulation

The BTA problem defined in Subsection 4.1 is formulated as an IP problem in this section. The formulation is path-based, which means that first for every block all possible paths from the origin-nodes to the destination-nodes of that block should be enumerated. The enumeration of these paths is described in Subsection 4.3. The decision involved with this path-based formulation is to assign at most one path to each block. As stated above in the introduction of Chapter 4, the IP formulation is based on Jha et al. [11]. Our IP formulation has the same constraints as in Jha et al. [11] plus some extra constraints. The extra constraints in our formulation are: a constraint on the maximum weight of a train, a constraint on the maximum length of a train, a constraint that ensures that a block can be assigned to a path from the moment that all demands in the block have arrived at the origin location of the block and a constraint that determines whether a train is used or not. The objective function is also slightly different from the objective function in Jha et al. [11] as we have, next to the costs of the paths that are chosen, also fixed costs when a train is used. Besides, we also allow a block to be not transported as we create an 'empty'-path with high penalty costs for each block. Our IP formulation is shown below:

Sets:

B	set of all the blocks created in the blocking problem, indexed by b
A	set of all arcs (train, connection and ground arcs), indexed by a
D_b	set of all demands that are in block $b \in B$, indexed by d
T	set of all trains, indexed by t
P_b^e	set of all paths for block $b \in B$ including the 'empty'-path (e) for block $b \in B$, indexed by p.
P_b	set of all paths for block $b \in B$ excluding the 'empty'-path for block $b \in B$, indexed by p. $P_b \subset P_b^e$
P_{ba}	set of all paths for block $b \in B$ with arc $a \in A$ included, indexed by p. $P_{ba} \subseteq P_b$
P_{dpb}^e	set of all paths (including the 'empty'-path) for the previous block (before block $b \in B$) of demand $d \in D_b$, with an end time earlier than (or exactly equal to) the start time of path $p \in P_b^e$
P_t	set of all paths with at least one trainleg of train $t \in T$ included, indexed by p

Parameters:

f_a	maximum number of wagons that can be assigned to arc $a \in A$
g_a	maximum total length of blocks that can be assigned to arc $a \in A$
h_a	maximum total weight of blocks that can be assigned to arc $a \in A$
n_b	number of wagons in block $b \in B$
l_b	length of block $b \in B$

w_b	weight of block $b \in B$
c_p	cost of assigning path $p \in P_b^e$ to the corresponding block
c_t	fixed cost of train $t \in T$

Decision Variables:

x_p	1 if path $p \in P_b^e$ is assigned to the corresponding block, 0 otherwise
y_t	1 if train $t \in T$ is used, 0 otherwise

$$min \sum_{b \in B} \sum_{p \in P_b^e} c_p x_p + \sum_{t \in T} c_t y_t \tag{4.1}$$

s.t.

$$\sum_{p \in P_b^e} x_p = 1 \qquad \forall b \in B \tag{4.2}$$

$$\sum_{b \in B} \sum_{p \in P_{ba}} n_b x_p \leq f_a \qquad \forall a \in A \tag{4.3}$$

$$\sum_{b \in B} \sum_{p \in P_{ba}} l_b x_p \leq g_a \qquad \forall a \in A \tag{4.4}$$

$$\sum_{b \in B} \sum_{p \in P_{ba}} w_b x_p \leq h_a \qquad \forall a \in A \tag{4.5}$$

$$x_p \leq \sum_{z \in P_{dpb}^e} x_z \qquad \forall b \in B, \ \forall p \in P_b^e, \ \forall d \in D_b \tag{4.6}$$

$$x_p \leq y_t \qquad \forall t \in T, \ \forall p \in P_t \tag{4.7}$$

$$x_p \in \{0, 1\} \qquad \forall p \in P_b^e, \ \forall b \in B \tag{4.8}$$

$$y_t \in \{0, 1\} \qquad \forall t \in T \tag{4.9}$$

Next, we will describe the objective function and each of the constraints:

- The objective function (4.1),

$$min \sum_{b\in B}\sum_{p\in P_b^e} c_p x_p + \sum_{t\in T} c_t y_t,$$

minimizes the sum of the costs of all paths that are assigned to a block and the fixed costs for all trains that are used.

- Constraints 4.2,

$$\sum_{p\in P_b^e} x_p = 1 \qquad \forall b \in B,$$

ensure that each block is assigned to exactly one path.

- The next three constraints (4.3, 4.4 and 4.5),

$$\sum_{b\in B}\sum_{p\in P_{ba}} n_b x_p \le f_a \qquad \forall a \in A$$

$$\sum_{b\in B}\sum_{p\in P_{ba}} l_b x_p \le g_a \qquad \forall a \in A$$

$$\sum_{b\in B}\sum_{p\in P_{ba}} w_b x_p \le h_a \qquad \forall a \in A,$$

ensure that the total number of wagons, total length and total weight, respectively, of all blocks assigned to an arc do not exceed the maximum number of wagons, length and weight of that arc.

- Constraints 4.6,

$$x_p \leq \sum_{z \in P^e_{dpb}} x_z \qquad \forall b \in B, \;\; \forall p \in P^e_b, \;\; \forall d \in D_b,$$

ensure that a block can only be assigned to a path if all demands in the block have arrived at the origin location of the block before the start time of the path.

- The next constraints (4.7),

$$x_p \leq y_t \qquad \forall t \in T, \;\; \forall p \in P_t,$$

ensure that a train ($t \in T$) is used (y_t is set to 1) if there exists a path which is assigned to its block and which includes at least one trainleg of train $t \in T$.

- Constraints 4.8,

$$x_p \in \{0,1\} \qquad \forall p \in P^e_b, \;\; \forall b \in B,$$

ensure that a path is either assigned to its corresponding block or not.

- Finally, constraints 4.9,

$$y_t \in \{0,1\} \quad \forall t \in T,$$

ensure that a train is either used or not.

As mentioned earlier in this section, an 'empty'-path with high penalty costs is created for each block. An 'empty'-path has no arcs and it indicates that the block is not transported. So, if a block is assigned to an 'empty'-path it means that the block is not transported and the high penalty costs associated with the 'empty'-path are incurred. The start and end time of these 'empty'-paths should be set to a date earlier than the start of the planning period. This is needed for constraints 4.6. In these constraints a block can be assigned to a path if all demands have arrived at the origin location of the block before the start time of the path. If the previous block (before this block), of one of the demands in this block, was assigned to an 'empty-path', it means that the demand will never arrive at this location, and thus this block can not be assigned to a path. Therefore, we set the start and end time of the 'empty'-paths to a date earlier than the start of the planning period, as the constraint will then recognize the 'empty'-path as a path that finishes earlier than the start time of the current path. Hereby, the block has not to wait on demands that will never arrive at the origin location of the block. These constraints are only created for all paths $p \in P_b$ and not for the 'empty'-paths, as it is always possible to assign a block to its 'empty'-path, independent of the previous and next blocks of the demands in this block.

The cost of a path, c_p, is the summation of the cost of all arcs (travel costs) included in the path plus train-switching costs for each demand in the block if the block has to switch to another train during the path. As the formulation is path-based and the paths are created for each block separately (P_b), penalty costs for demands that arrive too late can also be assigned to a path, if the destination of the block is the same as the destination of at least one of the demands in the block. If then, for these demands, the end time of the path is later than the end time of the time-slot of the demand, penalty costs for delivering these demands too late are assigned to the path.

The formulation can be extended with more constraints. For example, constraints on the capacities at locations. We did not include such constraints, as we assume that locations have very large capacities which will not be exceeded. Adding location capacity constraints to this formulation, can be done by observing which arcs correspond to the demands present at a location at a certain time and restrict that these arcs together may not exceed the maximum location capacity of that location. This should be done for every location at any time (try to find time-periods in which no demands depart or arrive from a location and create one constraint for the whole time-period).

4.3 Path Enumeration

Before the IP model can be used, the possible paths for each block should be created. A path enumeration algorithm is described in this subsection. As the number of paths that can be created for a block between its origin-nodes and its destination-nodes can become very large, we only consider paths with a duration which is less than or equal to the duration of the shortest path multiplied with a certain value.

The algorithm starts with creating all paths for the latest origin node of a block. Thereafter the paths for all other origin nodes (in descending order of time) are created. To ensure that we can create a path from the origin node to one single other node, a *unique destination-node* is created. This unique destination-node has incoming arcs from all destination-nodes of the block. Below the algorithm is shown and thereafter explained:

```
for each origin node (descending ordered on time) do
    create a label with null predecessor at the origin-node (say $o_i(b)$);
    make a set $S = \{o_i(b)\}$;
    while $S \neq \emptyset$ do
        remove a label $l$ from $S$;
        for each outgoing arc $a$ of the node associated with label $l$ do
            if total duration to reach head node of arc $a$, via the path
            represented by $l$ and $a$, does not exceed the maximum
            allowed duration of a path for block $b$ then
                if head node of arc $a$ has at least one label of the
                previous origin node then
                    create a special label at the head node of $a$
                else
                    create a new label at the head node of $a$;
                    if head node of arc $a$ is not the unique
                    destination-node of block $b$ then
                        add the new label to $S$;
                    end
                end
            end
        end
    end
    Create the paths for this origin node ($o_i(b)$)
end
```

Algorithm 1: Enumerate all valid paths for a block

The algorithm makes use of labels. A label is assigned to every node visited in a path. The information kept by a label consists of: the node to which it is assigned, the label of the predecessor, the time that it takes to reach this node from the origin-node, the origin node of the path and whether the label is a *special label*. A special label is a label at a node which has also at least one label of the next origin-node in time assigned to it. The algorithm is performed for every origin node of every block. The origin nodes of a block are ordered in descending order of time, which means that paths for origin nodes later in time are earlier created.

It starts with assigning a label to an origin-node with no predecessor. Thereafter, this label is added to a set S. Then, in the while-loop one of the labels (l) from set S is chosen to be explored further (label l is removed from set S). Then, new labels are created at the head nodes of the outgoing arcs of the node associated with label l, if the total duration constraint is not exceeded. Such a label is a special label if the node has also at least one label of the previous origin node $o_{i-1}(b)$ (i.e. the next origin node in time) assigned to it. In this case the new label is not added to set S, because it does not need to be explored further as we know all paths from this node to the unique destination-node through the previous origin node. If, on the other hand, the node has not a label of the previous origin node assigned to it, a normal label is created at this node. Thereafter, the label is added to set S if the node is not the unique destination-node (as the path is then finished). The while-loop is repeated until set S is empty, which means that no labels should be explored further.

The last step for an origin node is to create the paths. All labels of the current origin node at the unique destination-node correspond to a path from the origin node to a destination node. As each label is connected with its previous label, we can find the exact path corresponding to the label. Besides, all special labels of the current origin node correspond to at least one path. To create paths from this special label we need to find all paths from the previous origin node that consists of one of the outgoing arcs of the node corresponding to the special label. Then, for all these paths the first part of the path, until the special label node is reached, is replaced by the path corresponding to the special label. In this way, we create paths from the current origin node whose last part is exactly the same as the last part of a path of the next origin node in time. As this can result in paths with a total duration that exceeds the maximum allowed duration, we only create the paths that do not exceed this duration.

Using the special labels makes the algorithm faster. If we did not use these special labels, we should continue labeling until all paths exceeds the maximum duration or have arrived at the destination. Hereby, a lot of 'duplicate' labels are created. When using the special labels we reuse the information about previous created paths, and we do not create these 'duplicate' labels.

Figure 7 shows an example of the algorithm. In this network we need to create all the paths for a block from the location at the left to the location at the right. The red and blue ground-node at the left correspond to two possible origin nodes of the block. The algorithm starts with the latest origin node in time (blue). The first label (l_1) is created at this node with *null* predecessor and is added to set S. Then a label from set S is chosen (l_1) to explore further. From this label we will create a new label l_2 at the train-departure node. Then l_1 is removed from set S and label l_2 is added to set S. Hereafter, in the next three iterations we create labels l_3, l_4 and l_5. Then, from l_5 we have two possible outgoing arcs and we create both labels l_6 and l_7 and add them to set S. Hereafter, the algorithm will find labels l_8, l_9 and l_{10}. Labels l_{10} is not explored further as it corresponds to the unique destination-node. Then, from label l_7 the other labels (l_{11} - l_{14}) are found. After label l_{14} is found set S becomes empty and this means that all paths are found for the current origin node. Two labels reached the unique destination-node, which means that two paths were found. Label L_{10} corresponds to the path $l_9 \rightarrow l_8 \rightarrow l_6 \rightarrow l_5 \rightarrow l_4 \rightarrow l_3 \rightarrow l_2 \rightarrow l_1$, which is the path via train 2 and train 4. In the same way, the other label (l_{14}) corresponds to the path via train 2 and train 5.

After the algorithm found the paths for the blue node, it starts with the red origin node. In the same way, labels $l_a - l_e$ are found for the red node. Then, label l_e has two outgoing arcs: one arc to the train-departure node of train 3 (where label l_f is created) and another arc to the ground-arrival node of train 2. This last node has a label (l_4) from the previous (blue) origin node assigned to it, and thus a special label (L_g) is created at this node. The special label is not explored further. Hereafter, labels l_h, l_i and l_j are created. Then, set S is empty, which means that all paths are found. For the red node, one label (l_j) reached the unique destination-node. This label corresponds to the path $l_i \rightarrow l_h \rightarrow l_f \rightarrow l_e \rightarrow l_d \rightarrow l_c \rightarrow l_b \rightarrow l_a$ (train 1 and train 3). For this red node one special label (L_g) is created, so there may be more valid paths for this origin node. L_g is created at the ground-arrival node of train 2. This node has one outgoing arc, namely the ground-arc to the ground-departure node of train 4. Then, all paths of the previous origin node which contain this arc correspond to possible paths for this origin node. In this example, both the path corresponding to l_{10} and the path corresponding to l_{14} contain this arc. As this special label reached its node via another way ($L_g \rightarrow l_e \rightarrow l_d \rightarrow l_c \rightarrow l_b \rightarrow l_a$) than the paths corresponding to l_{10} and l_{14}, we are only interested in the last part of the path starting at the node of the special label. For l_{10} this means the path $l_9 \rightarrow l_8 \rightarrow l_6 \rightarrow l_5 \rightarrow l_4$ and for l_{14} this means the path $l_{13} \rightarrow l_{12} \rightarrow l_{11} \rightarrow l_7 \rightarrow l_6 \rightarrow l_5 \rightarrow l_4$. If we then add these paths to the path from the origin node to the node of the special label, we get the paths: $l_9 \rightarrow l_8 \rightarrow l_6 \rightarrow l_5 \rightarrow l_G \rightarrow l_e \rightarrow l_d \rightarrow l_c \rightarrow l_b \rightarrow l_a$ (train 1 and train 4) and $l_{13} \rightarrow l_{12} \rightarrow l_{11} \rightarrow l_7 \rightarrow l_6 \rightarrow l_5 \rightarrow l_G \rightarrow l_e \rightarrow l_d \rightarrow l_c \rightarrow l_b \rightarrow l_a$ (train 1 and train 5). Finally, we check if the paths created via the special label are valid with respect to the maximum allowed duration. If this is not the case, these paths should be removed. Assume that in this example the duration of both paths is below the maximum allowed duration.

Now, the algorithm is finished as we have found all the paths. For the blue origin node paths [train 2 → train 4] and [train 2 → train 5] were found. For the red origin node path [train 1 → train 3] was found directly and paths [train 1 → train 4] and [train 1 → train 5] were found using the paths of the blue origin node (i.e. via a special label).

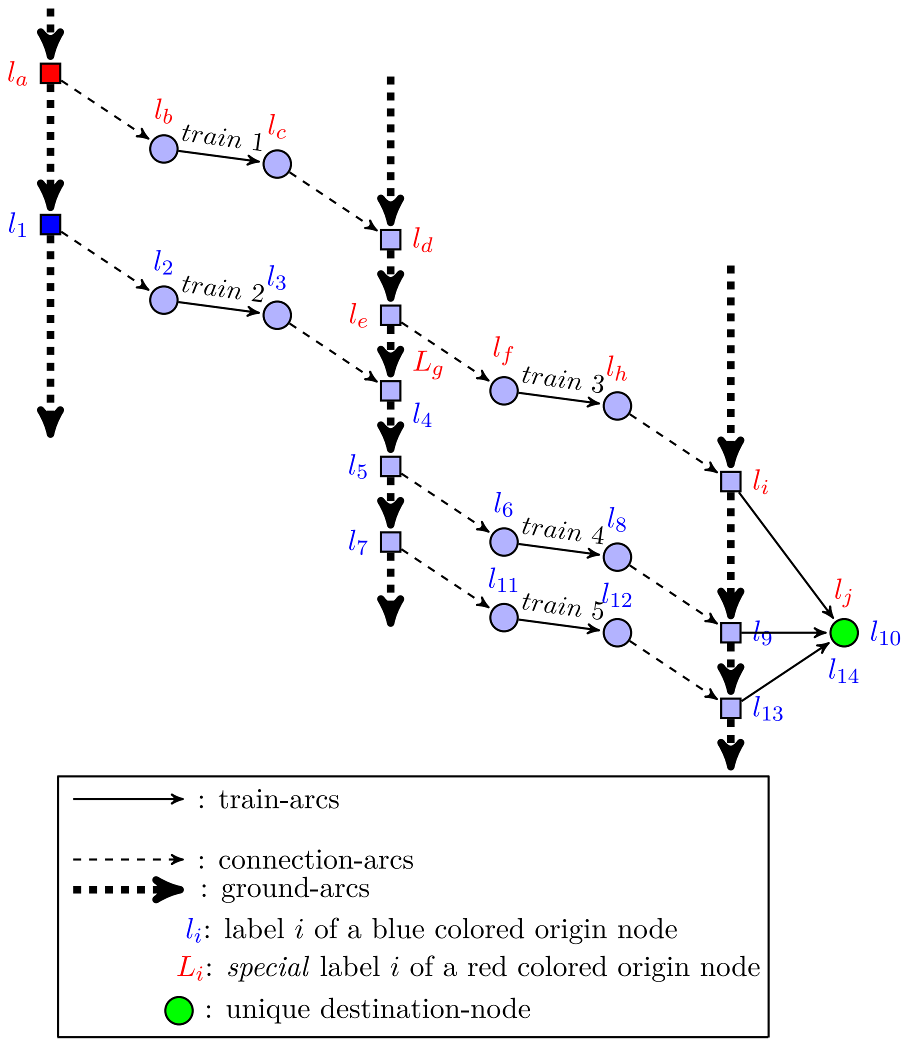


Figure 7: Example of the path enumeration algorithm

5 Heuristics

In this chapter we present several heuristics for each subproblem. A heuristic, based on the characteristics of a hub-and-spoke network, for the blocking problem is described in Section 5.1. In Section 5.2 a Lagrangian heuristic, based on Jha et al. [11], and a greedy heuristic are described to solve the BTA problem.

5.1 Blocking Heuristics

In this section we will describe some heuristics to solve the blocking problem. As stated in Subsection 2.5.1, the amount of blocks created in the blocking problem has an influence on the computation time and quality of the solution for the block-to-train assignment problem. It is important to find a good balance between the computation time and the quality of the solution. Therefore, we propose several heuristics to solve the blocking problem, which will differ in the amounts of blocks created.

5.1.1 No Blocking

This is the most easiest heuristic as it will create for each demand a separate block. Actually, it is not even a heuristic, as it just means that the demand-to-train allocation is solved directly. If the computation time of this demand-to-train allocation problem is acceptable and the enumeration of the paths is not restricted that much (i.e. a high multiplier for the maximum allowed path duration), this method will likely give the best solution.

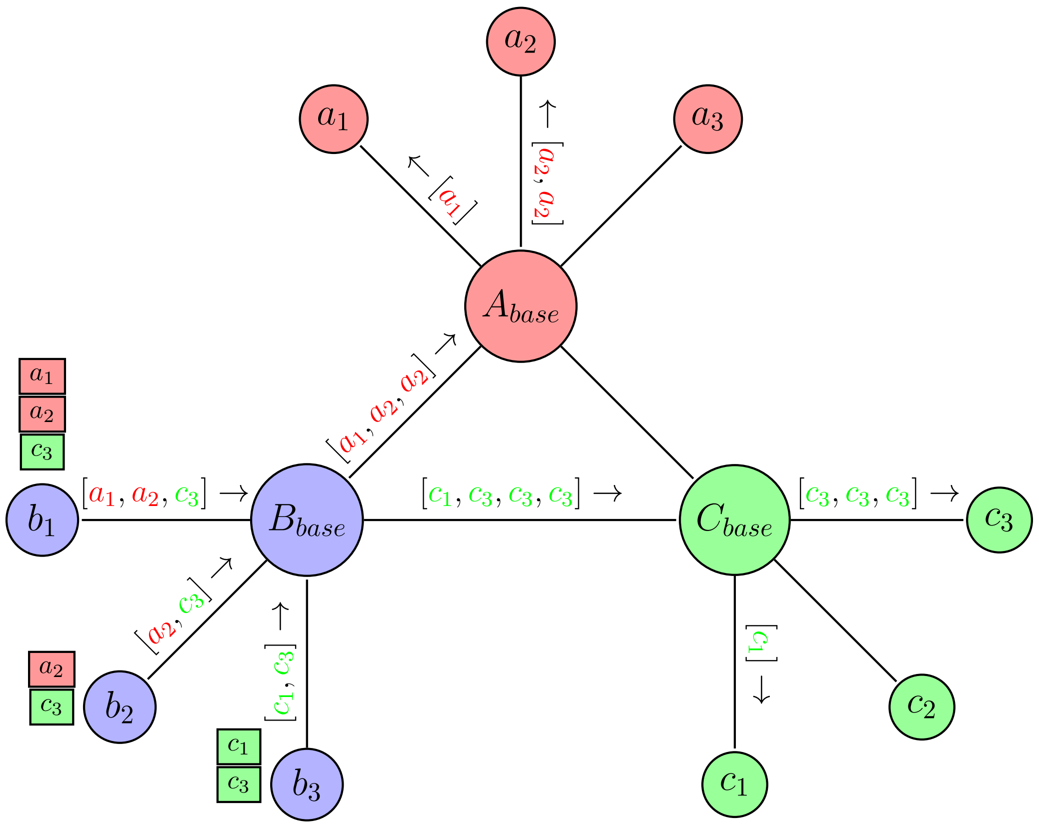
In the problem we discuss, the customer (and customer in circular) demands cannot be combined into blocks, and thus, these demands are always treated as separate blocks. As wood demands can be combined into blocks of at most two demands, combining them into blocks will not decrease the amount of blocks very much with respect to this method. Therefore, this method is also very suitable for wood demands. Liner and empty-wagon demands can be combined into blocks of several demands, whereby deriving a heuristic that will create less blocks is very interesting for these demands. In the next subsection a heuristic is presented where demands are combined into blocks.

5.1.2 'Hub-and-Spoke'-heuristic

This heuristic makes use of the characteristics of the hub-and-spoke network (Section 2.4). It is especially created for the liner and empty-wagon demands, as these make use of the hub-and-spoke network. In such a hub-and-spoke network all demands will pass the hubs corresponding to its origin and destination. The idea of this heuristic is that a demand is assigned to: a block from its origin to the base location corresponding to the origin (spoke $\rightarrow$ hub), a block from the base location corresponding to the origin to the base location corresponding to the destination (hub $\rightarrow$ hub) and to a block from the base location corresponding to the destination to its destination (hub $\rightarrow$ spoke). Then, demands with the same origin can be combined in blocks from that origin to its base location. Demands with the same origin hub and destination hub can be combined on these blocks and demands with the same destination can be combined from the destination hub to the destination.

Figure 8 shows an example of demands that are combined into blocks according to the three different block types as used in this heuristic. All demands at a service location are combined in a block to its base location. In Figure 8, for example, demands a_1, a_2 and c_3 at service location b_1 are combined in a block to the corresponding base location B. All demands at base locations are combined in a block to the base location corresponding to the destination. In Figure 8, for example, demand a_1 and both demands for location a_2 at base location B, are combined in a block to base location A. In the last part of the route of demands, demands are combined in a block from the base location, corresponding to the destination, to the destination. For example, all demands for c_3 at location C are combined in a block from C to c_3.

When combining demands into blocks we should also take into account that the release times of demands in one block should be relatively close to each other. Combining a demand that originates on a Monday in one block with a demand that originates four days later on Friday is not a good idea, as the Monday demand has to wait four days until it is transported. To indicate which demands can, and which demands can not, be combined into a block there is a limit on the time between the earliest and the latest *earliest arrival-time* of the demands in one block. This limit is called *earliest arrival-time difference limit* (EAT-limit). For the spoke $\rightarrow$ hub blocks the earliest arrival-time is just the release time of the demand. For the hub $\rightarrow$ hub and hub $\rightarrow$ spoke blocks the earliest arrival-time is the earliest possible time that the demand can arrive at the origin of the block. Hereby, the previous



All lines between two nodes correspond to blocks between these nodes. The arrow next to a block indicates the direction of the block.

Figure 8: An example of blocks in the 'Hub-and-Spoke'-heuristic

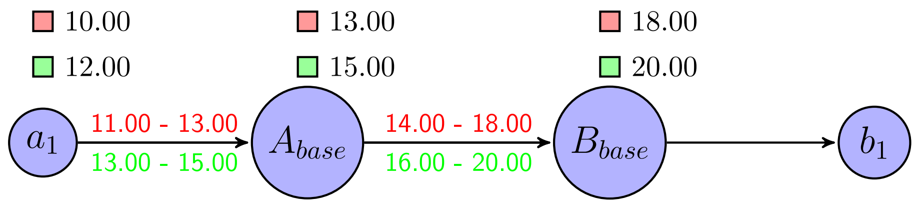
blocks should be taken into account as these affect the earliest arrival-time. Besides, all paths of the previous blocks should be known as the earliest arrival time of a demand at a base location is obtained from the earliest path of its previous block to this base location.

The order of the steps of this heuristic is as follows:

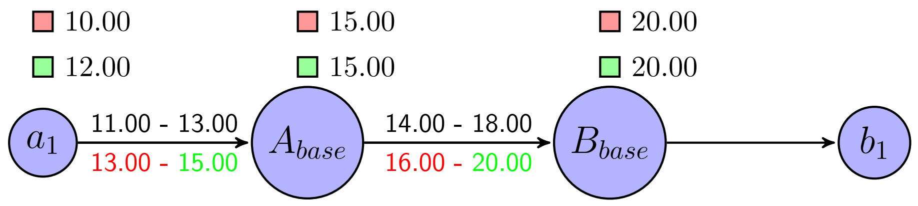
1. Create all spoke $\rightarrow$ hub blocks
2. Create all paths (see section 4.3) for the spoke $\rightarrow$ blocks
3. Create all hub $\rightarrow$ hub blocks
4. Create all paths for the hub $\rightarrow$ hub blocks
5. Create all hub $\rightarrow$ spoke blocks
6. Create all paths for the hub $\rightarrow$ spoke blocks

In Figure 9 an example is shown where the earliest arrival-time of a demand depends on the block it is assigned to. In the first case in the figure both demands are a separate block and the earliest arrival-time of the red demand at base location A is 13.00 (first possible train). The earliest-arrival time of the red demand at base location B is 18.00, as the earliest arrival-time at base location A is 13.00 and the first train from A to B after 13.00 is the train from 14.00 until 18.00. In the second example both demands are combined in one block from a_1 to A (recognize that this is only possible if the EAT-limit is at least two hours). The earliest arrival-time of the red demand at base location A is 15.00, as it can be assigned to a train from a_1 to A after the release of the second demand (at 12.00). Then, the first train is the train from 13.00 until 15.00. The first train from A to B after 15.00 is the train from 16.00 until 20.00, whereby the earliest arrival-time of the red demand at B is 20.00. So, this example shows that the earliest arrival-times of the red demand are different when it is combined in one block together with the green demand.

Both demands as a separate block:



Both demands in one block:



The two colored rectangles above the locations are a red demand and a green demand. The times next to the demands indicate the earliest-arrival time of the demand at that location. The times above and below the lines indicate the departure and arrival times of trains. A colored train time indicates that the demand with that color is assigned to the train.

Figure 9: An example of earliest-arrival times of demands

The goal of the heuristic is to create as few as possible blocks, while not exceeding the capacities of a block (weight, length and number of wagons) and the EAT-limit. The heuristic creates all blocks for a pair of locations. It should be applied to each spoke $\rightarrow$ hub, each hub $\rightarrow$ hub and each hub $\rightarrow$ spoke pair to create all blocks. Below the heuristic is described:

Steps of the 'Hub-and-Spoke'-heuristic:

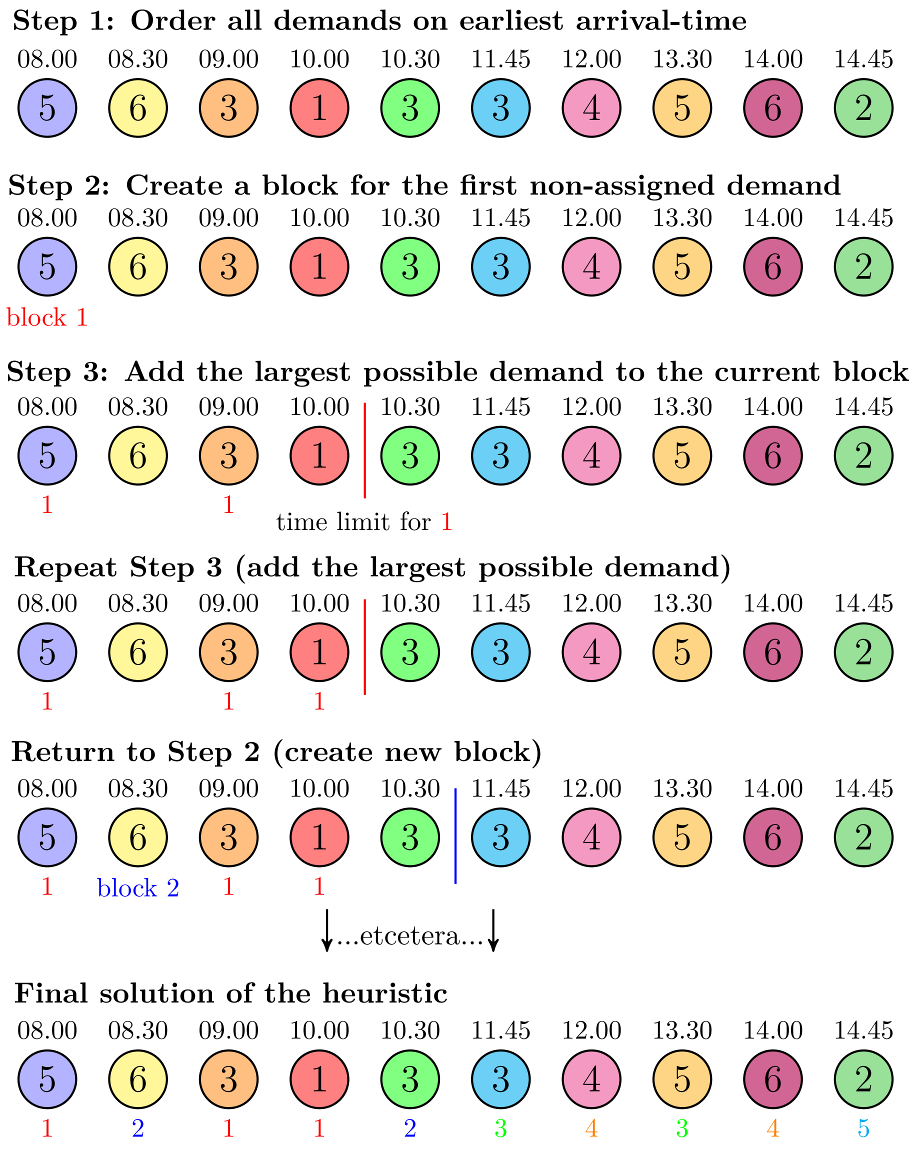
1. Order all the demands at the origin of this pair of locations on earliest arrival-time.

2. Pick the first demand, from the ordered list, that is not assigned to a block and create a new block for it. If all demands have already been assigned to a block, skip the third step and GO TO STEP 4.

3. Find, among all demands not assigned to a block and whose release is not too long (EAT-limit) after the release of the first demand in the block, the largest demand that fits into the current block and assign it to the block. If a demand has been found and added in this step, REPEAT STEP 3. If no demand has been found in this step, RETURN TO STEP 2.

4. The heuristic is finished, as all demands are assigned to a block.

The first step of the heuristic is to order all demands at the origin according to the earliest arrival-time. Hereby, it is easy to find the earliest non-assigned demand (step 2) and to see which demands can, according to the EAT-limit, be assigned to the current block (step 3) as all these demands will be next to each other in the list. In the second step a new block is created for the first non-assigned demand in the list. Then, in the third step the largest demand, which fits in the block and can be combined with the other demands in the block, is added to the block. This step is repeated until no more demands can be added to the block. Hereafter, we return to the second step and create a new block for the first non-assigned demand. Then again demands are assigned to this block in the third step, etcetera. The heuristic is finished when all demands are assigned to a block. In the third step we add the largest possible demand to a block, because the goal is to create as few as possible blocks and this means that the blocks should be as full as possible. If we want to get the blocks as full as possible it seems a good idea to assign in each step the largest possible demand to the block.



Step 1: Order all demands on earliest arrival-time
08.00 08.30 09.00 10.00 10.30 11.45 12.00 13.30 14.00 14.45
5 6 3 1 3 3 4 5 6 2
Step 2: Create a block for the first non-assigned demand
block 1
Step 3: Add the largest possible demand to the current block
1 1
time limit for 1
Repeat Step 3 (add the largest possible demand)
1 1 1
Return to Step 2 (create new block)
1 block 2 1 1
...etcetera...
Final solution of the heuristic
1 2 1 1 2 3 4 3 4 5

*All nodes in this example correspond to a demand. The number in the node is the weight of the demand, and the time above the node is the earliest arrival-time of the demand. Here, in this example, the EAT-limit is **two hours** and the capacity of a block is **ten**.*

Figure 10: Example of the 'Hub-and-Spoke'-heuristic

Figure 10 shows the steps of the heuristic with an example. The example consists of ten demands with each a corresponding earliest arrival-time and weight. The maximum weight of a block is ten and the EAT-limit is two hours. In the first step the demands are ordered with respect to their earliest arrival-time. Hereafter, a new block is created for the first demand in the order. Then, in the third step we first investigate which demands can be added to the current block. The earliest-arrival time of the first demand is 08.00 and, as the EAT-limit is two hours, all demands with an earliest arrival-time no later than 10.00 can be assigned to the current block. The maximum capacity of a block is ten and the weight of the first demands is five which results in a remaining capacity of five. The three demands that are within the EAT-limit have weights six, three and one. The demand with weight three is the largest demand that fits into the block and is thus assigned to this first block. Hereafter, the third step is again repeated and we will find that the demand with weight one can also be added to the block. Then, no more demands can be added to this first block and the heuristic returns to the second step and creates a new block for the first demand in the order that is not assigned so far (in Figure 10 the yellow demand with weight six). Then the third step is performed for the second block, etcetera. The final solution of the example is also shown in Figure 10. The heuristic creates for this example five blocks with total weights: nine, nine, eight, ten and two.

In our problem the blocks have a maximum number of wagons, maximum weight and maximum length. Hereby, we need a rule to find the largest demand, because if, for example, one demand has more wagons than another, but the other demand is heavier, it is not very clear which demand is the 'largest' demand. Many different rules can be created for finding the largest demand. The most obvious rules are enumerated below.

Some rules for finding the largest demand:

1. Pick the demand with the most number of wagons
2. Pick the heaviest demand
3. Pick the longest demand
4. Largest demand = $\max_{\forall\, demands} \{ \frac{nr\ of\ wagons}{max\ nr\ of\ wagons} + \frac{weight}{max\ weight} + \frac{length}{max\ length} \}$

The first three rules just choose one of the three capacity categories to determine the largest demand. The last rule combines all three capacity categories. The number of wagons, weight and length of a demand are divided by the maximum number of wagons, weight and length of a block. Each of these three expressions represent the percentage of the total capacity (of the corresponding category) in a block that will be used by this demand. These three expressions are summed up, and then the largest demand is the demand with the largest value for this expression.

These are the four most obvious rules. For each individual problem it is the best to investigate which of the capacities is the most restricting, and use that capacity to determine whether a demand is larger than another demand. This is also what we did in Chapter 6. For the problem instances discussed in Chapter 6, the weight was the most restricting capacity. Therefore, we have chosen rule 2 (take the heaviest demand as largest demand) for the 'Hub-and-Spoke'-heuristic for these problem instances.

5.2 Block-to-Train Assignment Heuristics

This section present heuristics to solve the IP formulation of the BTA problem as described in Chapter 4. The first heuristic is a Lagrangian heuristic and is, like the BTA model, also based on Jha et al. [11]. In this approach all the capacity constraints are relaxed and for each capacity constraint a penalty is assigned for each unit capacity violation on an arc. The relaxed problem is then solved. If the solution of the relaxed problem contains arcs with violated capacity, the corresponding Lagrangian multipliers are increased and the problem is solved again. In each iteration a lower bound and an upper bound on the solution of the BTA problem are calculated. The solution value of the relaxed problem with violated arc capacities provides a lower bound on the optimal solution. An upper bound is created via a heuristic which creates a feasible solution for the BTA problem from the solution of the relaxed problem. The heuristic terminates if for a certain number of consecutive iterations the solution does not improve.

The second heuristic is a greedy heuristic. This heuristic is created to get a good solution within minimal computation time. The Lagrangian heuristic is described in subsection 5.2.1 and the greedy heuristic is described in subsection 5.2.2.

5.2.1 Lagrangian Heuristic

In this section the Lagrangian heuristic is explained. The heuristic consists of three steps, which are explained in the next three subsections. First, the procedure to create a lower bound is described, then the procedure to get an upper bound (i.e. feasible solution) is described and in the last subsection the procedure of updating the Lagrangian multipliers, at the end of each iteration, is described.

Obtaining a Lower Bound

In each iteration of the heuristic a lower bound is obtained by solving the relaxed problem. In the relaxed problem all capacity constraints (location capacity, maximum number of wagons, maximum length and maximum weight) are relaxed and penalties are assigned for capacity violation. The decision to relax all capacity constraints is based on the Lagrangian relaxation approach of Jha et al. [11]. We also tested the case where we also relax constraints (5.4), but this gave worse results than our current approach. Therefore, we decided to only relax the capacity constraints. For the constraints on the number of wagons (4.3) a penalty $q_a \geq 0$ is assigned to each unit capacity violation on every arc $a \in A$. For the length (4.4) and weight (4.5) capacity constraints a penalty $r_a \geq 0$ and $s_a \geq 0$, respectively, is assigned to each unit capacity violation on every arc $a \in A$. These capacity constraints are then integrated into the objective function with their corresponding Lagrangian multipliers. The formulation of the relaxed problem is shown below:

$$\begin{aligned} min \ \sum_{b \in B} \sum_{p \in P_b^e} c_p x_p \ + \ \sum_{t \in T} c_t y_t \ + \ \sum_{a \in A} \Bigg(& q_a (\sum_{b \in B} \sum_{p \in P_{ba}} n_b x_p - cn_a) \\ & + r_a (\sum_{b \in B} \sum_{p \in P_{ba}} l_b x_p - cl_a) \ + \ s_a (\sum_{b \in B} \sum_{p \in P_{ba}} w_b x_p - cw_a) \Bigg) \end{aligned} \tag{5.1}$$

$$\sum_{p \in P_b^e} x_p = 1 \qquad \forall b \in B \tag{5.2}$$

$$x_p \leq \sum_{z \in P_{dpb}^e} x_z \qquad \forall b \in B, \ \forall p \in P_b^e, \ \forall d \in D_b \tag{5.3}$$

$$x_p \leq y_t \qquad \forall t \in T, \;\; \forall p \in P_t \tag{5.4}$$

$$x_p \in \{0,1\} \qquad \forall p \in P_b^e, \;\; \forall b \in B \tag{5.5}$$

$$y_t \in \{0,1\} \qquad \forall t \in T \tag{5.6}$$

The above IP model is solved to optimality (using CPLEX) in each iteration for given Lagrangian multipliers q, r and s to obtain a lower bound on the BTA problem. After each iteration the Lagrangian multipliers are updated in order to obtain a less violated solution in the next iteration. This process of updating the Lagrangian multipliers is described in Subsection 5.2.1.

Obtaining an Upper Bound

The next step in an iteration is to create a feasible solution (upper bound) from the infeasible solution of the relaxed problem. As the infeasible solution of an iteration is known, the violated arcs can be identified and all the blocks assigned to these violated arcs are unassigned. Hereafter, no arc capacities are violated, but there are some blocks that are not assigned to any arc. Finally, the blocks that are not assigned to any arc, are assigned to arcs. This is done in the following way: The blocks are first ordered in descending order with respect to the weight of a block. Then, the first block in the order is assigned to the least-cost path (including fixed train costs) with enough spare capacity for this block and such that the order of the blocks is not violated. The fixed train costs are only added to a path for trains that are in the path and not used so far. In this way, we try to assign these blocks to trains that have already been used so far. Hereafter, the second block in the order is assigned to the least-cost path with enough spare capacity for that block and such that the order of blocks is not violated. Then the third block in order is assigned, etcetera (see Algorithm 2 for a description of the algorithm).

When all blocks are assigned to a path, we have created a feasible solution. As a block can only be assigned to a path if it does not violate the order of the blocks of all demands in the current block, we might get a lot of non-assigned blocks (i.e. blocks which are assigned to its 'empty'-path). Therefore, we want to improve this feasible solution. As we might get a lot of non-assigned blocks, we first try to assign these paths to another, cheaper path. To assign such a block to another path, we need to assign the previous and/or next blocks of demands in the current block to other paths such that the current block is able to start earlier and/or finish later. Thereby, there may be 'new' paths available for the current block that do not violate the order of blocks of demands in the current block anymore. Then, the current block is assigned to one of its possible paths, and all previous and next blocks of demands in the current block can be rescheduled again to their cheapest possible path. This might result in a cheaper schedule in which the current block is assigned to a path. If it was not possible to improve the schedule, we keep the current schedule. This is done for each path to which the current block can be assigned after the rescheduling of the previous and next blocks of demands in the current block. Hereafter, we repeat the same steps, as we did for these non-assigned blocks, but now for all blocks, to try to achieve an even better schedule. The resulting solution is the upper bound of the current iteration. The best (lowest) upper bound found in all iterations is kept as upper bound on the BTA problem. The global idea of creating a feasible solution from the infeasible solution is described above. Below, the algorithm is described in detail:

```
obtain the infeasible solution of the relaxed problem;
for all violated arcs in the infeasible solution do
    $\boxed{\text{Unassign blocks on an overcapacitated arc}}$ (see Algorithm 3 below)
end
order all blocks $b \in B$ in descending order w.r.t. their weight;
for each block $b$ in the above order do
    order the paths $p \in P_b^e$ in ascending order w.r.t. their costs plus fixed
    costs for all non-used trains in the path;
    for each path $p$ in the above order do
        if the capacities and the order of the blocks of all demands in block
        $b$ are not violated when assigning block $b$ to path $p$ then
            assign block $b$ to path $p$;
            update the available train capacity of train arcs on path $p$;
            exit for
        end
    end
end
for all non-assigned blocks $b \in B$ do
    $\boxed{\text{Create the largest possible gap for } b}$ (see Algorithm 4 below)
    for all paths $p \in P_b$ to which block $b$ can be assigned do
        $\boxed{\text{Create the largest possible gap for } b}$ (see Algorithm 4 below)
        assign block $b$ to path $p$;
        assign the previous and next blocks of demands in $b$ to their
        cheapest possible paths;
        if this solution is better than the best solution so far then
            keep this solution;
        else
            set solution back to the best solution;
        end
    end
end
Repeat the above for-loop again, but now for all blocks;
```

Algorithm 2: Obtain a feasible solution from the infeasible solution of the relaxed problem

```
if there exists a block $b$ such that the capacities are met if $b$ is removed from
the current arc then
    Remove the smallest block $b$ from the current arc such that the
    capacities are met;
else
    Remove the largest block $b$ from the current arc
end
```

Algorithm 3: Unassign blocks on an overcapacitated arc

```
for all demands $d$ in block $b$ do
    if block $b$ is not the first block of demand $d$ then
        assign the previous block (before $b$) of demand $d$ to the earliest
        possible path
    end
    if block $b$ is not the last block of demand $d$ then
        assign the next block (after $b$) of demand $d$ to the latest possible
        path
    end
end
```

Algorithm 4: Create the largest gap for a block

Updating the Lagrangian Multipliers

At the end of each iteration the Lagrangian multipliers are updated in order to obtain a less violated solution for the relaxed problem in the next iteration. We use Subgradient Optimization to update the Lagrangian multipliers. The Lagrangian multipliers for the three different capacity constraints at iteration i are indicated as q_a^i, r_a^i and s_a^i. The idea is that the Lagrangian multipliers for arcs whose capacities are violated are increased to make these arcs more unattractive and that Lagrangian multipliers for arcs whose capacities are not violated decrease to make these arcs more attractive. For example, in iteration i we have, among others, multiplier r_a^i. If the total length of all blocks assigned to arc a is larger than the capacity (i.e. the constraint is violated), the Lagrangian multiplier is increased to make this arc more unattractive in the next iteration. So, $r_a^{i+1} > r_a^i$. If the total length of all blocks assigned to arc a is not larger than the capacity (i.e. the constraint is not violated) and $r_a^i > 0$, the Lagrangian multiplier is decreased to make this arc more attractive. So, $r_a^{i+1} < r_a^i$. The following formulas are used to update the Lagrangian multipliers at iteration i:

$$q_a^{i+1} = q_a^i + \theta_q^i \left(\sum_{b \in B} \sum_{p \in P_{ba}} n_b x_p - c n_a \right) \tag{5.7}$$

$$r_a^{i+1} = r_a^i + \theta_r^i \left(\sum_{b \in B} \sum_{p \in P_{ba}} l_b x_p - c l_a \right) \tag{5.8}$$

$$s_a^{i+1} = s_a^i + \theta_s^i \left(\sum_{b \in B} \sum_{p \in P_{ba}} w_b x_p - c w_a \right) \tag{5.9}$$

where θ_q^i, θ_r^i and θ_s^i are the step lengths at iteration i. These step lengths are updated using the following formulas:

$$\theta_q^i = \frac{\lambda_q^i (UB - LB_i)}{|| \sum_{a \in A} \sum_{b \in B} \sum_{p \in P_{ba}} n_b x_p - c n_a ||^2} \tag{5.10}$$

$$\theta_r^i = \frac{\lambda_r^i (UB - LB_i)}{|| \sum_{a \in A} \sum_{b \in B} \sum_{p \in P_{ba}} l_b x_p - c l_a ||^2} \tag{5.11}$$

$$\theta_s^i = \frac{\lambda_s^i (UB - LB_i)}{|| \sum_{a \in A} \sum_{b \in B} \sum_{p \in P_{ba}} w_b x_p - c w_a ||^2} \tag{5.12}$$

where λ_q^i, λ_r^i and λ_s^i are scalars used to adjust the size of the change in the step length. A common way to update these scalars is to multiply them with a half, if, for a certain number of iterations, the best solution did not change. In these formulas for updating the step length UB is the best upper bound found in all iterations so far, while LB_i is the lower bound found in iteration i.

5.2.2 Greedy Heuristic

In this section a greedy heuristic to solve the BTA problem is presented. The heuristic starts with the creation of a good feasible solution. Thereafter, it tries to improve the initial feasible solution.

The initial feasible solution is obtained by assigning each block to a possible path with the earliest end time, starting with the block with the earliest possible departure time, then the block with the second earliest possible departure time, etcetera. If it is not possible to assign a block to a path, it is assigned to the 'empty'-path. We have ordered the blocks on earliest possible departure time, because in that way it is ensured that the blocks of a demand are planned in the right order (the first block before the second block, the second block before the third block, etcetera). Otherwise, if we first assign the third block of a demand to its earliest possible path, it is possible that due to capacity reasons the second and/or first block can not be assigned to a path that finishes before the starting time of this third block. Ordering the blocks will results in less non-assigned blocks, and thus, a better initial solution.

Hereafter, we try to assign the blocks to cheaper paths. When creating the initial solution we only focused on time and not on costs, while we want to get the cheapest possible schedule. In this step, each block is assigned to the cheapest possible path, and this is done in the reverse order as when creating the initial solution. Now, we start with the block with the latest, earliest possible starting time, than the second latest, etcetera. This is done, because as we first assigned the blocks to a possible path with the earliest possible departure time, all blocks before the last block are enclosed by its next block, and thus, have not many other paths to which it can be assigned given the current schedule. The last block of a demand has many paths to which it can be assigned as all blocks are assigned to the earliest possible path (so the previous blocks will end early) and the possible paths of the last block are not restricted by the starting time of a next block (as there is no next block). If we have then assigned the last block to a later, cheaper path, the previous block of the demand has also more possible paths to assign to, as it is allowed to end later as the next block starts later. This will, if this step is done for each block, result in a cheaper (and thus better) initial solution.

In the previous steps we did not change the paths of previous and/or next blocks when trying to assign a certain block to a path. In this step we will assign the previous and/or next blocks of demands in a certain block to other paths. This is done, because assigning a block to a slightly more expensive path can have as a result, that it is now possible for other blocks to be assigned to much cheaper paths. This step is exactly the same as the last step of Algorithm 2 (which creates an upper bound from the infeasible solution of the Lagrangian relaxation). For each block we try to get as much as possible paths to which the block is able to be assigned, by assigning the previous blocks of demands in the current block to the earliest possible path and the next blocks of demands in the current block to the latest possible path. If we assign the current block to one of these available paths, we should assign the previous and next blocks of demands in the current block to their cheapest possible path. As in Algorithm 2, this is first done for all paths that are not-assigned to a path, and thereafter for all blocks.

If we have to solve the BTA problem with one block for each demand (i.e. without blocks), the last part of the heuristic is not necessary as none of the blocks has a previous or next block. The heuristic then boils down to the first two for-loops, there we first assign the blocks to the earliest possible path (in the right order) after which we assign them to the cheapest possible path (in the right order). The algorithm of this greedy heuristic (Algorithm 5) is shown below:

```
order all blocks $b \in B$ in ascending order w.r.t. its earliest possible
departure time;
for each block $b$ in the above order do
    assign block $b$ to the earliest possible path;
end
order all blocks $b \in B$ in descending order w.r.t. its earliest possible
departure time;
for each block $b$ in the above order do
    assign block $b$ to the cheapest possible path;
end
for all non-assigned blocks $b \in B$ do
    $\boxed{\text{Create the largest possible gap for } b}$ (see the appendix)
    for all paths $p \in P_b$ to which block $b$ can be assigned do
        $\boxed{\text{Create the largest possible gap for } b}$ (see the appendix)
        assign path $p$ to block $b$;
        assign the previous and next blocks of demands in $b$ to their
        cheapest possible paths;
        if this solution is better than the best solution so far then
            keep this solution;
        else
            set solution back to the best solution;
        end
    end
end
Repeat the above for-loop again, but now for all blocks;
```

Algorithm 5: Greedy heuristic algorithm

6 Computational Results

In this chapter we evaluate the IP model and the heuristics on several instances with different sizes. The different instances are described in Section 6.1. Hereafter, in Section 6.2, the results and the computation times of the IP model and the heuristics are compared with each other. Besides, the results of the BTA problem without blocks and with blocks, created via the blocking heuristic described in this thesis, are compared in this section. Finally, in Section 6.3, we perform a sensitivity analysis. We investigate the effect of the EAT-limit, the effect of the maximum path duration multiplier and the effect if we limit the number of paths per origin node of a block.

All tests are performed on a laptop with 8 GB RAM memory and an Intel(R) Core(TM) i7-4800MQ cpu @2.7GHz. The IP model and all heuristic are programmed in the Quintiq software package and we use CPLEX 12.5 to solve the IP formulation to optimality.

6.1 Data

In this section we will describe the four different instances. As the liner and empty demands are the most difficult to plan (hub-and-spoke network) and these demands can be combined into blocks, we investigate only instances with liner and/or empty demands. For the other two demand types, solving the problem is slightly different (and easier). For customer demands we only have to solve the demand-to-train allocation directly, as it is not allowed to combine these demands into blocks. Wood demands can be combined into blocks with at most two demands per block, and each wood demand can be in at most one block. This is because, wood demands can only be combined in a block together with another wood demand with the same origin and destination. Table 1 below shows the properties of the instances.

Instance	Demands	Trains	Base locations	Service locations
1	10	20	3	5
2	30	50	3	7
3	100	100	4	11
4	500	250	7	25
5	1000	400	7	25

Table 1: Properties of the different problem instances

As the table shows, the sizes vary from ten demands and twenty trains to 1000 demands and 400 trains. In the smaller instances all base locations have approximately the same number of service locations and there is no big difference in the incoming and outgoing flow of demands between all locations. In instances four and five we varied the number of service locations per base location and we created large differences in the incoming and outgoing flow of demands per location. In real-life there may be locations with only incoming or outgoing flow. For example, forests with only outgoing flow of wood. To imitate these properties of real-life problems we created, in the largest problem instance, some locations with only an outgoing flow of demands and some locations with only an incoming flow of demands. The network of the fourth and fifth instance are exactly the same, only the number of demands and trains are different. For all instances, the maximum allowed duration of a path for a block is 1.5 times the duration of the shortest path of the block, and we use an EAT-limit of eight hours (when creating the blocks).

The penalty costs for blocks which are not delivered are set very high with respect to the traveling and fixed train costs such that a solution with i non-delivered demands is always cheaper than a solution with $i + 1$ non-delivered demands. The most important task for the algorithm is thus to minimize the amount of non-delivered demands. Also, the penalty costs for demands which arrive too late are set very high (but lower than the penalty costs for a non-delivered block) with respect to the traveling and fixed train costs, such that it is always better to ensure that a demand arrives on time, even if that means that the demand has to travel a longer distance. The second most important task for the algorithms is thus to minimize the number of demands which arrive too late. The sum of the traveling costs and fixed train costs is the least important task of the three for the algorithm to minimize. Therefore, the tables with results in the next section have three columns: a column for the number of non-delivered demands, a column for the number of demands which arrive too late and a column with the sum of the traveling costs and fixed train costs.

6.2 Results

This section will evaluate and compare the results of the BTA problem for the IP model (with and without blocking) and the heuristics on several instances with different sizes. First, we compare the results of the IP model and the heuristics if no blocks are created. These results are shown below in Table 2.

<u>**Explanation of the column names:**</u>

Time: total computation time,
ND: number of blocks which are not delivered,
Late: number of demands which are delivered too late,
Costs: sum of traveling costs and fixed train costs

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	190	10	1sec	1	0	11,645
2	684	30	5sec	0	0	29,168
3	2078	100	58sec	2	0	77,927

(a) IP model

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	190	10	1sec	1	0	11,650
2	684	30	5sec	0	0	29,168
3	2078	100	64sec	4	0	81,076

(b) Lagrangian heuristic

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	190	10	1sec	1	0	11,685
2	684	30	3sec	0	0	31,011
3	2078	100	57sec	4	0	79,530

(c) Greedy heuristic

Table 2: Results of the IP model and heuristics without blocking

Only the results of the first three instances, and thus not of the fourth and fifth instance, are shown in Table 2. This is, because we were not able to solve the largest two instances with the model without blocking within reasonable amount of time (it takes at least fifteen hours to create all the paths for these instances). As both heuristics also need to have all the paths before the algorithm can start, we have also no solution of both heuristics for the largest problem instance.

From the results of the three instances shown in Table 2 we can see that the computation time of the IP model and both heuristics are about the same. For the first two problem instances the results of the heuristics did not differ that much from the result of the IP model, while the results of both heuristics for the third problem instance are (much) worse than the result of the IP model for the third problem instance. Via the IP model we found a solution with two demands which are not delivered, while both heuristics come up with a solution with four demands not delivered. On the other hand, the sum of the travelling and fixed train costs for both heuristics do not differ that much from the costs of the IP model.

Next, we compare the results of the IP model and heuristics without blocking, which we discussed in the previous paragraph (see Table 2 for these results), with the results of the IP model and heuristics when we create blocks via the 'Hub-and-Spoke'-heuristic (see Table 3 below for these results).

Explanation of the column names:
Time: total computation time,
ND: number of blocks which are not delivered,
Late: number of demands which are delivered too late,
Costs: sum of traveling costs and fixed train costs

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	59	18	<1sec	0	0	10,874
2	306	62	1sec	0	0	24,979
3	1,182	181	26sec	0	0	69,132
4	11,882	672	176sec	0	0	248,601
5	23,842	941	483sec	1	0	361,494

(a) IP model

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	59	18	<1sec	0	0	10,874
2	306	62	1sec	0	0	25,039
3	1,182	181	23sec	1	0	72,395
4	11,882	672	3206sec	1	0	250,647
5	23,842	941	11459sec	5	0	368,595

(b) Lagrangian heuristic

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	59	18	<1sec	0	0	11,870
2	306	62	<1sec	0	0	27,972
3	1,182	181	13sec	0	0	81,205
4	11,882	672	475sec	0	0	259,225
5	23,842	941	1966sec	2	0	384,067

(c) Greedy heuristic

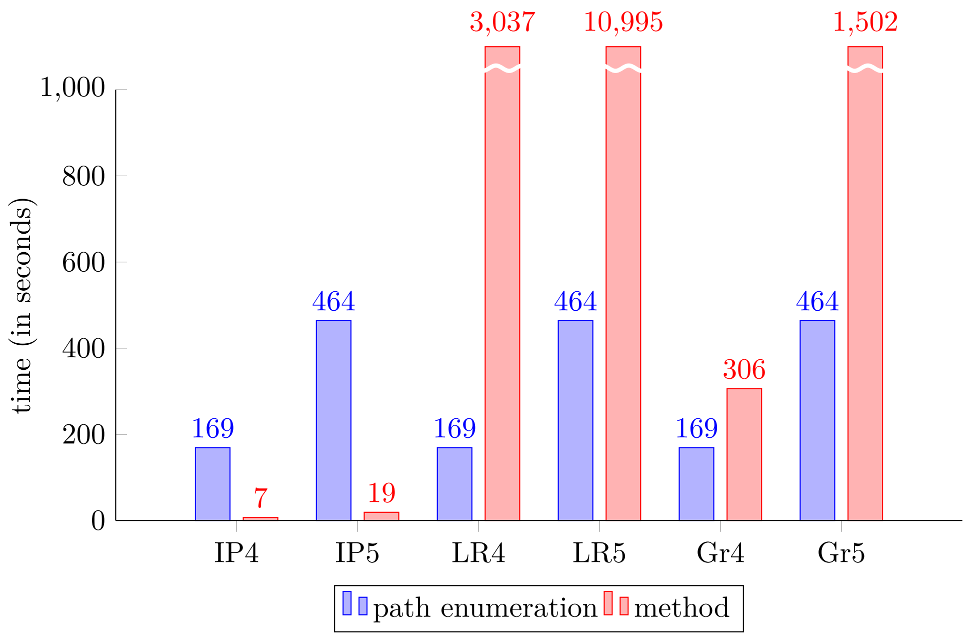
Table 3: Results of the IP model and heuristics when blocks are created

First, we observe that we are able to solve the largest two problem instances when we create blocks, and that the computation time of the IP model is very low (less than three minutes for the fourth instance and approximately eight minutes for the fifth instance), if we keep in mind that it was not able to solve these instances when we do not create blocks. When we observe the results of both tables, we see that we get much better results for the IP model as well as for both heuristics in case of blocking. For all instances the costs are lower when blocks are created, while in most instances there are even less demands not delivered than in the case where no blocks are created. The fact that it is better to create blocks seems, first and foremost, strange, but can be explained by the fact that we restricted the maximum allowed path duration. This was in the first place done to restrain the duration of the path-enumeration, and also to restrain the duration of the IP model and the heuristics. A consequence of this is that paths with a long waiting time between two different trainlegs may not be considered as these may exceed the maximum allowed path duration. In the case where blocks are created, it is possible to have a long waiting time between two consecutive blocks of a demand. This gives more flexibility and is the reason that in our case creating blocks gives better results. When we do not consider the maximum allowed path duration for the problem without blocking, its results will be at least as good as in the case where we do create blocks. This is, because then the solution space of the problem with blocking is a subset of the solution space of the problem without blocking. However, the results of both tables shows that, even if we restrict the maximum duration of a path, the problem without blocking can not be solved within reasonable amount of time. Therefore, for our approach, it is better to first create blocks.

An unexpected observation from Table 3 is that the computation time of the IP model is much shorter than the computation time of both heuristics, for the fourth and fifth instance. This is strange, as the main reason to create a heuristic is to get good results within less computation time than the IP model. The IP model gives the best results and has the shortest computation time for these instances, whereby it is the best to solve instances of these sizes with the IP model. However, we do not know the behavior of the IP formulation for much larger problems, so it is still useful to investigate the results of the heuristics as they might be needed if we want to solve much larger problems.

If we compare the results of the different methods when creating blocks (Table 3) we see that the costs of the solutions of the Lagrangian heuristic are in all instances relative close to the costs of the solution of the IP model, but for the larger instances one extra block (with respect to the solution of the IP model) was not delivered. The costs of the solutions of the greedy heuristic are always higher than the costs of the solutions of the Lagrangian heuristic, but for the larger instances the solutions of the Lagrangian heuristic have one extra block which is not delivered. If the penalty costs for not delivering a block are very high, the greedy heuristic seems to give the best results, but if these penalty costs are relative low the Lagrangian heuristic seems to give the best results. If we compare the computation times of both heuristics, we see that for the larger instances the greedy heuristic is much faster than the Lagrangian heuristic. Therefore, we prefer the greedy heuristic over the Lagrangian heuristic.

To take a further look into the computation times of the methods, we have splitted them up in the computation time of the path enumeration algorithm and the computation time of the method itself. Figure 11 shows these computation times for the IP model and both heuristics, for the largest two instances. This figure shows that the IP formulation is solved within very little time: seven seconds for the fourth instance and just nineteen seconds for the largest instance. These computation times are much shorter than we had expected for the IP formulation to solve instances of these sizes. Compared to the time needed to solve the IP formulation, the computation times of the methods itself, of both heuristics, are very long. The Lagrangian heuristic itself is approximately 434 (579) times slower than solving the IP model of the fourth (fifth) instance. The greedy heuristic itself is approximately 44 (79) times slower than solving the IP model of the fourth (fifth) instance. So, the methods of the heuristics take way too long. We investigated the reason for these large computation times and found out that most of the time is spend with checking whether a block can be assigned to a path or not. In both heuristics this action is done very often, as we repeatedly try to assign blocks to other paths to get a cheaper schedule. Every time we want to try to assign a block to another path we have to check whether that is possible with respect to all capacities and with respect to the order of the blocks of a demand. This is an essential part of the method as it ensures that we end up with a feasible solution. Therefore, it is not possible to leave this part out of the method or to adjust it, to achieve less computation time.



The names below the bars represent the method (first two letters) and the problem instance (number).

Figure 11: Splitted computation times

6.3 Sensitivity Analysis

In this section we check how sensitive the model reacts to changes in different parameters. In Subsection 6.3.1 we investigate the effect of the EAT-limit. As creating blocks with different EAT-limits will result in different blocks and a different total amount of blocks, this may have an effect on the solution of the BTA problem. Next, in Subsection 6.3.2 we investigate the effect of different maximum allowed path duration multipliers. As creating paths with different multipliers will result in different amount of paths, this may also have an effect on the solution of the BTA problem. Finally, we investigate how the solution and the computation time of the IP model change, if we only select the ten cheapest paths at each origin node of a block.

6.3.1 Effect of Earliest Arrival Time limit

In this section the effect of different EAT-limits on the BTA problem is investigated. In Section 6.2 the blocks were created with an EAT-limit of eight hours. In this section we compare these results with the results of the IP model when the blocks are created with EAT-limits of four and twelve hours. The results are shown below in Table 4.

Explanation of the column names:
Time: total computation time,
ND: number of blocks which are not delivered,
Late: number of demands which are delivered too late,
Costs: sum of traveling costs and fixed train costs

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	68	20	<1sec	0	0	10,874
2	341	71	1sec	0	0	24,979
3	1,367	207	100sec	0	0	69,082
4	12,010	697	188sec	0	0	248,594
5	24,211	988	462sec	1	0	361,470

(a) EAT-limit of 4 hours

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	59	18	<1sec	0	0	10,874
2	306	62	2sec	0	0	24,979
3	1,182	181	26sec	0	0	69,132
4	11,882	672	176sec	0	0	248,601
5	23,842	941	483sec	1	0	361,494

(b) EAT-limit of 8 hours

Instance	# Paths	# Blocks	Time	ND	Late	Costs
1	56	17	<1sec	0	0	10,874
2	303	61	1sec	0	0	24,979
3	1,115	168	130sec	1	0	67,968
4	11,870	669	178sec	0	0	248,601
5	23,805	936	448sec	1	0	361,532

(c) EAT-limit of 12 hours

Table 4: Results of the IP model when creating blocks with different EAT-limits

From Table 4 we can see that, in most cases, different EAT-limits do not give very different results for the IP model. As expected, we get more blocks (and thus more paths) if we decrease the EAT-limit, but, as said earlier, this do not result in big changes in the solution value.

This table shows two unexpected observations. The first unexpected observation is that for the third problem instance, in case of an EAT-limit of twelve hours, one demand was not delivered, while in case of both other EAT-limits all demands were delivered. This can be explained by the fact that in case of a large EAT-limit (twelve hours) it is possible that demands are combined in one block which have a relative big difference in release time. This can result in demands which cannot be delivered on time, or, as in this case, blocks that cannot be delivered. The other unexpected observation is that it took the IP model for the third instance in case of an EAT-limit of eight hours just 26 seconds to solve the problem, while in case of both other EAT-limits (four and twelve hours) it took the IP model 100 and 130 seconds to solve the problem. We have no clear reason for this and it seems like a coincidence.

6.3.2 Effect of Maximum Path Duration

In this section the effect of different multipliers, for the maximum allowed path duration, on the BTA problem is investigated. In Section 6.2 the instances where solved with a maximum allowed duration of a path for a block of 1.5 times the shortest path duration. In this section we compare these results with the results of the IP model when using multipliers 1.25, 1.75 and 2. We use for all different multipliers the IP model with blocks created with an EAT-limit of eight hours, as Section 6.2 showed that it is better to create blocks than not to create blocks. The results are shown below in Table 5.

Explanation of the column names:

Time: total computation time,

ND: number of blocks which are not delivered,

Late: number of demands which are delivered too late,

Costs: sum of traveling costs and fixed train costs

Instance	# Paths	# Blocks	Time	Not	Late	Costs
1	51	19	<1sec	4	0	11,409
2	282	66	<1sec	0	0	26,950
3	1,087	181	13sec	0	0	70,292
4	7,460	672	155sec	2	0	252,694
5	18,076	941	388sec	1	0	367,364

(a) Multiplier = 1.25

Instance	# Paths	# Blocks	Time	Not	Late	Costs
1	59	18	<1sec	0	0	10,874
2	306	62	1sec	0	0	24,979
3	1,182	181	26sec	0	0	69,132
4	11,882	672	176sec	0	0	248,601
5	23,842	941	483sec	1	0	361,494

(b) Multiplier = 1.5

Instance	# Paths	# Blocks	Time	Not	Late	Costs
1	59	18	<1sec	0	0	10,874
2	308	62	1sec	0	0	24,979
3	1,241	181	89sec	0	0	67,231
4	12,221	672	185sec	0	0	242,722
5	28,353	941	496sec	1	0	360,037

(c) Multiplier = 1.75

Instance	# Paths	# Blocks	Time	Not	Late	Costs
1	61	18	<1sec	0	0	10,874
2	320	62	1sec	0	0	24,979
3	1,273	181	206sec	0	0	66,568
4	13,742	672	203sec	0	0	240,843
5	32,901	941	546sec	1	0	358,350

(d) Multiplier = 2

Table 5: Results of the IP model with blocking, for different maximum allowed path duration multipliers

In Table 5, as expected, the number of paths increases if the maximum allowed path duration multiplier increases. The biggest difference in the number of paths is the step from a multiplier of 1.25 to a multiplier of 1.5. Thereafter, from 1.5 to 1.75 and from 1.75 to 2, the differences between the number of paths created are smaller.

If we compare the results of the models with different multipliers, we see that the model with multiplier 1.25 gives much worse results than the other multipliers. For instance one and four the solution has more blocks not delivered than the solutions of the other multipliers. Besides, using multiplier 1.25 gives for most instances relatively much higher costs than the other multipliers. This shows that we should not take a too low value for the multiplier. The results of the other three multipliers (1.5, 1.75 and 2) are not very different from each other with respect to computation time and costs. There are just two relative large differences between the results of these three multipliers. The first difference is in the computation time of the third instance. For the other instances the computation times are relative close to each other, but for this third instance the computation time is 26 seconds for multiplier 1.5, 89 seconds for multiplier 1.75 and 206 seconds for multiplier 2, which are relative large differences. The other relative large difference is in the result of the fourth instance. The model with multiplier 1.5 has one more demand not delivered than the models with multipliers 1.75 and 2. There is also a clear difference in costs: 248,601 (1.5) versus 242,722 (1.75) and 240,843 (2).

As the results of multipliers 1.75 and 2 are not much different and the computation time of multiplier 1.75 is for the third instance and also for the fifth instance relatively much shorter than the computation time of multiplier 2, we conclude from these results that 1.75 is a good choice for the maximum allowed path duration multiplier.

6.3.3 Effect of Maximum Number of Paths

In this section we investigate, especially, the effect on the solution value of the IP model if we limit the number of paths per origin node of a block. As we showed in Section 6.2, the computation time to solve the IP formulation is very low for all our instances, whereby we can not decrease this computation time that much for these instances. The idea behind this section is to investigate how (and if) the solution value changes when we limit the number of paths per origin node of a block, as this may be important if we want to solve larger problems which have much larger computation times for the IP formulation.

We decided to select for each origin node of each block only the ten cheapest paths. This is done after we have first created all the paths. The IP model is then solved with the selected paths. We expect that the solution is not much worse than the solution of the IP model with all paths, because we expect that a block is mostly assigned to one of the cheapest paths of one of its origin nodes. We have restricted the maximum number of paths for an origin node of a block for both the IP problem without blocking and with blocking. The results of the IP model, without blocking, with all paths and with at most ten paths per origin node of a block are shown below in Table 6. Again, only the first three instances are shown, as creating all the paths for the fourth and fifth instance can not be done in reasonable amount of time (more than fifteen hours).

Explanation of the column names:
Time: total computation time,
ND: number of blocks which are not delivered,
Late: number of demands which are delivered too late,
Costs: sum of traveling costs and fixed train costs

Instance	# Paths	# Blocks	Time	Not	Late	Costs
1	190	10	1sec	1	0	11,645
2	684	30	5sec	0	0	29,168
3	2078	100	58sec	2	0	77,927

(a) All paths

Instance	# Paths	# Blocks	Time	Not	Late	Costs
1	95	10	1sec	1	0	11,645
2	363	30	5sec	0	0	29,168
3	1370	100	56sec	2	0	77,927

(b) Maximum ten paths per origin node of a block

Again we have chosen to compare the results for the model with a maximum allowed path duration multiplier of 1.5.

Table 6: Results of the IP model without blocking with a maximum number (10) of paths per origin node of a block

The results in both cases are exactly the same. As we expected, only selecting the ten cheapest paths for each origin node of a block do not result in a (much) worse solution, while the number of paths was decreased with approximately 50%. On the other hand, as expected, the computation time for all problem instances is approximately the same as solving the IP formulation is, in terms of computation time, just a very small part of the method.

Next, we compare the results of the IP model with blocking when we select only the ten cheapest paths per origin node of a block with the results of the IP model with blocking with all paths. Table 7 below shows these results:

Explanation of the column names:
Time: total computation time,
ND: number of blocks which are not delivered,
Late: number of demands which are delivered too late,
Costs: sum of traveling costs and fixed train costs

Instance	# Paths	# Blocks	Time	Not	Late	Costs
1	59	18	<1sec	0	0	10,874
2	306	62	1sec	0	0	24,979
3	1,182	181	26sec	0	0	69,132
4	11,882	672	176sec	0	0	248,601
5	23,842	941	483sec	1	0	361,494

(a) All paths

Instance	# Paths	# Blocks	Time	Not	Late	Costs
1	59	18	<1sec	0	0	10,874
2	306	62	1sec	0	0	24,979
3	1,182	181	26sec	0	0	69,132
4	11,882	672	172sec	0	0	248,601
5	23,842	941	472sec	1	0	361,494

(b) Maximum ten paths per origin node of a block

Again we have chosen to compare the results for the model with a maximum allowed path duration multiplier of 1.5 and an EAT-limit of eight hours.

Table 7: Results of the IP model with blocking with a maximum number (10) of paths per origin node of a block

First, we observe from Table 7 that the results do not change when we only select the ten cheapest paths of each origin node of every block. For the first three instances this is not a surprise, as for these instances there is no origin node of a block with more than ten paths. Therefore, no paths can be removed and the instances are thus solved with all paths. For the fourth and fifth instance a lot of paths were removed, such that, for both instances, just slightly more than 40% of all the paths remains. Though, this has no influence on the solution.

As we mentioned beforehand, the decrease in computation time is not that much, as the IP formulation was already solved very fast with all the paths. However, this section has showed that if we have to solve larger problems with a much longer computation time for the IP formulation, only selecting the ten cheapest paths per origin node of a block is a good idea to decrease the computation time.

7 Conclusion and Discussion

In the first section of this chapter we will give our conclusion. Thereafter, in Section 7.2 we will mention some points with room for improvement.

7.1 Conclusion

This thesis presented a way to solve demand-to-train allocation problems in rail freight transportation. This problem is divided in two subproblems: the blocking problem and the block-to-train assignment (BTA) problem. The BTA problem is defined on a space-time network, after which the BTA problem was formulated as an IP model. In addition, heuristics were created to solve both subproblems.

For the blocking problem a heuristic has been created, which creates blocks based on the characteristics of the hub-and-spoke network. This heuristic has been compared with the case where one block is created for each demand from its origin to its destination (which actually means that no blocks are created and the demand-to-train allocation problem is solved directly). When comparing the results of the BTA problem with and without blocks, we saw that the results of the BTA problem with blocks were better. This is explained by the fact that we restricted the maximum path duration. Also, the computation time of BTA problems with blocking was much better. Especially, the time of creating all paths was much better when blocks were created. This was also the reason that, if we do not create blocks, it was not possible to create all paths for the largest two instances. This is caused by the fact that the length of the paths of blocks is much shorter than the length of the paths for a demand. Therefore, we concluded that it is much better to create blocks, and thus to divide the demand-to-train allocation problem into a blocking problem and a block-to-train assignment problem.

We also compared the results of the IP model with the results of the heuristics. Surprisingly, the computation time of the IP model was much shorter than the computation time of both heuristics. This was on one hand due to a very short computation time for solving the IP formulation and, on the other hand, checking whether a block can be assigned to a train (which is done often in the heuristics) turned out to be a time-consuming task, as both the capacities of a train and the right order of the blocks of a demand should be checked. As it turned out that the IP model can be solved very efficiently in little computation time, even for our largest instances, we conclude that it is best to solve the block-to-train assignment problem, which are of a size similar to the size of our instances, with the IP model. As we do not know the behavior of the IP formulation for much larger problems, it is possible that we need a heuristic when we want to solve much larger problems. Therefore, we compared the results of both heuristic and concluded that we prefer the greedy heuristic over the Lagrangian heuristic as the results are better and its computation time is much shorter.

7.2 Discussion

One of the points of this thesis with room for improvement is the creation of the paths for each block. Our path enumeration algorithm enumerates all possible paths for a block which do not exceed the duration of the shortest path times a certain multiplier. This resulted in the fact that, when we do not create blocks, it was not possible (due to computation time) to create all the paths for each demand for our largest two instances. Besides, if we do create blocks, the biggest part of the computation time of the IP model was spend by the path enumeration algorithm and just a very small part by solving the IP formulation itself. If we want to decrease the computation time in the case where we do create blocks and if we want to solve larger instances in case of no blocks, the path enumeration algorithm should be made more efficient. It may be possible to speed up the current path enumeration algorithm with some smart tricks, but the most obvious way to make the path enumeration algorithm more efficient is to create a method which does not create all the paths for a block. As Section 6.3.3 showed, we create a lot of useless paths, and it might therefore be an idea to create a method which creates just a subset of the paths for a block, including the most important paths for the block. A possible approach for creating a subset of all the paths is a K-shortest path algorithm. Another possibility to create the paths more efficiently is via column generation.

Another point to focus on is the greedy heuristic. We preferred this heuristic over the Lagrangian heuristic and, therefore, we might try to adjust the heuristic a little bit to improve the solutions. Its computation time is much shorter than the computation time of the Lagrangian heuristic and for some instances it found a solution with less demands not delivered than the Lagrangian heuristic. Only the sum of the fixed train costs and traveling costs was, for all instances, worse (higher costs) than the costs of the solution of the Lagrangian heuristic. These costs were also the biggest difference between the solution of the greedy heuristic and the optimal solution. We found out that, for all instances, the costs were mainly higher due to higher fixed train costs. This can be explained by the fact that we focused mainly on delivering as much as possible demands (we did this because we assume that the penalty costs for non-delivered blocks are very high). The last step of the greedy heuristic tries to assign blocks to other paths to get a cheaper schedule. Hereby, some blocks are assigned to another path such that one of the trains from its previous path is not used anymore, whereby we get a cheaper schedule, but it might be worth to try to extend the heuristic with a step that focuses especially on decreasing the number of trains used.

Finally, we spoke in this thesis about blocks which are not delivered, but actually it is important to know if demands are not delivered. The problem is then that if we count the number of demands in a non-delivered block and assign costs to it, it is possible that some demands are in multiple blocks which are not delivered. So, then it is possible that the costs for not delivering a demand are counted more than once. Therefore, it is better that, when a demand is in a non-delivered block, the demand is not taken into account in its other blocks. Besides, it should be nice that, when a block can not be delivered because of one of its demands, it is possible to delete that demand from its blocks, whereby the other demands in the block can be delivered. In our approach all demands in a block are not delivered, when it is not possible to transport the block to its destination. However, focusing more on the individual demands in blocks is not so easy.

References

[1] R. K. Ahuja, J. Liu, J. B. Orlin, D. Sharma, and L. A. Shughart. Solving real-life locomotive scheduling problems. Transportation Science 39, 503-517, 2005.

[2] R. K. Ahuja, K. C. Jha, and J. Liu. Solving real-life railroad blocking problems. Interfaces 37, 404-419, 2007.

[3] A. A. Assad. Modelling of rail networks: Toward a routing/makeup model. Transportation Research 14B, 101-114, 1980.

[4] C. Barnhart, P. M. Vance, and H. Jin. Railroad blocking: A network design application. Operations Research 48, 603-614, 2000.

[5] L. D. Bodin, B. L. Golden, A. D. Schuster, and W. Romig. A model for the blocking of trains. Transportation Research 14B, 115-120, 1980.

[6] K. C. Campbell. Booking and revenue management for rail intermodal services. Ph.D. thesis, Department of Systems Engineering, University of Pennsylvania, USA, 1996.

[7] T. G. Crainic and J. M. Rousseau. Multicommodity, multimode freight transportation: a general modeling and algorithmic framework for the service network design problem. Transportation Research 20B, 225-242, 1986.

[8] M. F. Gorman. An application of genetic and tabu searches to the freight railroad operating plan problem. Annals of Operations Research 78, 51-69, 1998.

[9] A. E. Haghani. Formulation and solution of a combined train routing and makeup, and empty car distribution model. Transportation Research 23B, 433-452, 1989.

[10] C. L. Huntley, D. E. Brown, D. E. Sappington, and B. P. Markowics. Freight routing and scheduling at csx transportation. Interfaces 25, 58-71, 1995.

[11] K. C. Jha, R. K. Ahuja, and G. Şahin. New approaches for solving the block-to-train assignment problem. Networks 51, 48-62, 2008.

[12] M. H. Keaton. Designing optimal railroad operating plans: Lagrangian relaxation and heuristic approaches. Transportation Research 23B, 415-431, 1989.

[13] M. H. Keaton. Designing railroad operating plans: A dual adjustment method for implementing lagrangian relaxation. Transportation Science 26, 263-279, 1992.

[14] O. K. Kwon, C. D. Martland, and J. M. Sussman. Routing and scheduling temporal and heterogeneous freight car traffic on rail networks. Transportation Research 34E, 101-115, 1998.

[15] A. Marin and J. Salmerón. Tactical design of rail freight networks. part i: Exact and heuristic methods. European Journal of Operational Research 90, 26-44, 1996.

[16] A. Marin and J. Salmerón. Tactical design of rail freight networks. part ii: Local search methods with statistical analysis. European Journal of Operational Research 94, 43-53, 1996.

[17] A. M. Newman and C. A. Yano. Centralized and decentralized train scheduling for intermodal operations. IIE Transactions 32, 743-754, 2000.

[18] H. N. Newton. Network design under budget constraints with application to the railroad blocking problem. Ph.D. thesis, Industrial and Systems Engineering Department, Auburn University, USA, 1996.

[19] H. N. Newton, C. Barnhart, and P. M. Vance. Constructing railroad blocking plans to minimize handling costs. Transportation Science 32, 330-345, 1998.

[20] L. K. Nozick and E. K. Morlok. A model for medium-term operations planning in an intermodal rail-truck service. Transportation Research 31A, 91-107, 1997.

[21] C. D. Van Dyke. The automated blocking model: A practical approach to freight railroad blocking plan development. Transportation Research Forum 27, 116-122, 1986.

[22] M. Yaghini and R. Akhavan. Multicommodity network design problem in rail freight transportation planning. Procedia - Social and Behavioral Sciences 43, 728-739, 2012.

[23] M. Yaghini, A. Foroughi, and B. Nadjari. Solving railroad blocking problem using ant colony optimization algorithm. Applied Mathematical Modeling 35, 5579-5591, 2011.

[24] M. Yaghini, M. Seyedabadi, and M. M. Khoshraftar. A population-based algorithm for the railroad blocking problem. Journal of Industrial Engineering International 8, 1-11, 2012.

[25] Y. Yue, L. Zhou, Q. Yue, and Z. Fan. Multi-route railroad blocking problem by improved model and ant colony algorithm in real world. Computers and Industrial Engineering 60, 34-42, 2011.