International Journal of Transportation
Vol.2, No.1 (2014), pp.53-76
http://dx.doi.org/10.14257/ijt.2014.2.1.04

Metaheuristic for the Integrated Approach to the Freight Train Routing and Block-to-train Assignment

Martim Moniz[1], Diogo Silva[2] and João Telhada[3]

[1]Graphes et Optimisation Mathématique, Université Libre de Bruxelles
[2]Department of Statistics and Operations Research, Faculty of Science, University of Lisbon
[3]Center of Operations Research, University of Lisbon
[3]joao.telhada@fc.ul.pt

Abstract

Freight railroad transportation is a critical issue in an economic setting with growing concern about pollution and road congestion. With an efficient management, the rail option for freight transportation may lead to more competitive costs. In this article, we address one part of the complex issue of freight railroad transportation, which deals with the definition of train routes and, simultaneously, with the way freight is carried from its origin to its destinations. The integration of these two problems is called Train Design. A new genetic algorithm is proposed to solve this problem, which tries to find a freight processing order so that the iterative dispatch subproblems leads to a lower global cost solution. Some results show the effectiveness of the proposed algorithm.

Keywords: *Railway management; Metaheuristics*

1. Introduction

Nowadays, railway is considered to be a very interesting option for freight transportation, in particular due to the lower CO2 emissions, as well as the lower traffic congestion when compared with road solutions. Additionally, economy growth has leveraged commercial transactions to a larger geographical scale and to higher volumes. These two aspects, and especially the combination of both, grant particular importance to freight railway management.

For freight to be taken from its origin to its destination, at least one train is required. One first problem would be how to route the train and which timetable to choose. However, it wouldn't be economically efficient to assign one train to each freight. One possible alternative, but equally inefficient in economic terms, as well as potentially unfeasible due to technical aspects, would be to take just one train and establish a round trip which would traverse all origin-destination pairs.

The dilemma raised above suggests that a sensible and intelligent approach is advisable and that freight train management is, in general terms, a relevant problem. Nevertheless, it is also a complex one, due to the several and different phases freight railway management comprises. Figure 1 illustrates the various phases of railway management and their hierarchy, as suggested by Cordeau *et al.*, [7].

The first step, called blocking, consists in analyzing different goods, by comparing their origin, destination and other relevant characteristics, such as size and weight, and then grouping them so that they are transported together, from one particular yard to another, as one block. However, this does not necessarily mean that all the freight in a block has the same final destination. In fact, due to reclassification done at intermediate yards, certain goods may be part of more than one block along its trip. Bodin *et al.*, [5] wrote one of the first papers about the blocking problem, while more recent references include Ahuja *et al.*, [1], Newton *et al.*, [16], Barnhart *et al.*, [4] and Yaghini *et al.*, [19].

As shown in Figure 1, after the blocking process, one next step may consist in train routing, where pre-established routes are chosen, according to anticipated freight flow patterns. This problem can be intertwined with the train timetable problem, resulting in the so-called train scheduling problem, which has been extensively studied in recent years ([2, 14, 20, 15, 10]). Figure 1 highlights the aggregation of these two subproblems with a dashed contour.

Using as input the solutions given by the blocking and train routing phases, decisions can then be made about each block's itinerary. This problem is designated as block-to-train assignment ([14, 17, 13]). The block's trip plan must include information regarding which trains transport the block, and the different yards from where the block is either picked up from or set off.

The final phase of this process usually respects crew scheduling, where several constraints are considered so that an assignment of crews to the train schedule is feasible.

Train routes may be decided at a strategic level, either because that is within the organization's guidelines, or because altering that decision in a regular basis may have significant costs or logistic problems associated with it. When block flow patterns are reasonably stable, this strategic level decision happens to suit well. In these cases, train routes may be decided based on some forecast of block flows. When blocks are dispatched, their routes have to match with those of trains.

However, if we consider the opposite case, where block flows regularly change, this approach becomes inefficient, as train routes may not be adequate in many cases. If that's the case, and if other conditions are met, train routing should be placed, instead, at an operational level. That is, train routes should be reconsidered at a regularly basis. To increase the efficiency of this process, train routing and block-to-train assignment should be tackled as a whole.

This integrated approach will designated as train design problem, and is introduced in Section 2, where all characteristics are broadly described. In Section 3, a heuristic procedure is proposed to obtain efficient solutions for the problem. Finally, some robustness tests are presented in Section 4.

2. Problem Description

As mentioned before, a block is a set of railcars, transporting freight, which will be moved as a single group from one yard to another. Each block has an associated number of railcars, weight and length, as well as an origin and destination yard. In a graph representation, yards will correspond to nodes in the network, which is also composed by links corresponding to the existing and usable railways.

The train design problem, integrating train routing and block-to-train assignment problems, consists in determining routes for trains, which take into consideration several technical constraints, and assigning blocks to train routes so that certain criteria are met. In the context considered, no reclassification is allowed. This means that blocks are considered to be unchangeable throughout the whole trip, and moves between trains are allowed for blocks only as a whole. Those operations are called block swaps [14], and a constraint is considered to upper bound the number of swaps a block may be involved during its trip. That upper bound is denoted by MBS.

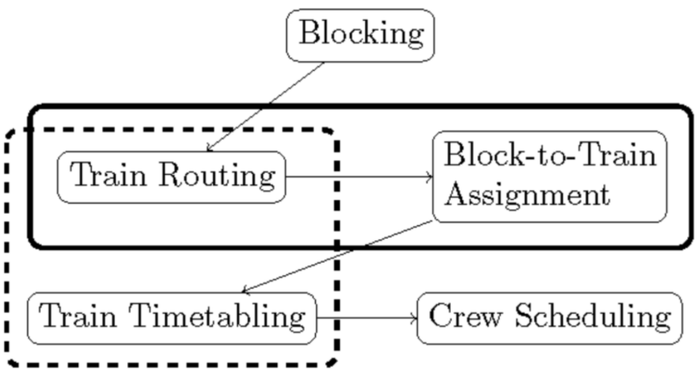

Figure 1. Schematic Illustration of Some Important Phases of Freight Train Management

Since blocks are assigned to trains, special care should be given to the formations obtained, in terms of length and tonnage, as well as the total number of blocks carried by a train. Therefore, constraints are considered which bound the length and weight of a train, and the total number of blocks carried, for each segment of its trip, to certain pre-defined values denoted, respectively, by ML(a), MW(a) and MBT.

Picking up and setting off blocks can also be seen from the perspective of the train, since these operations usually imply substantial effort, both in terms of time consumption and the financial resources involved. Any pickup or set-off block operation associated with a train, at any of its intermediate stops, is called a work event. To keep resources at sustainable levels, usually a constraint on the number of work events is included, for each train. The maximum number allowed for those operations, for each train, is denoted by MWE.

Furthermore, crews are required for the train operation and typically have associated pre-defined routes, further designated by crew segments, where they perform that operation. Therefore, train routes have to match crew segments, from the train's start node up until its end node. To guarantee this, a train may be required to travel without any blocks, however inefficient it may seem. Crew segments are regarded as bi-directional, in the sense that a certain crew can operate a train in any, or even in both, of the crew segment's directions. In the context of this work, crews are also considered to be unlimited and available on any of the segment's endpoints, even if that requires deadheading.

Each crew segment comprises a certain number of adjacent links, with an associated distance. In order to guarantee rail road infrastructure stability, each link a may have an upper bound restriction regarding the total number of trains that can use it, denoted by TTL(a).

In Figure 2, an instance is presented and two possible train designs are illustrated. The network is depicted in the upper left corner where, besides nodes representing yards, the origin and destination of each block are included. In the bottom left corner, the existing crew segments, and their respective sequence of yards, are shown. On the right, two possible solutions are presented with some relevant differences. For each solution, the routes of trains are presented, including the crews operating the trains in each segment. In solution 1, block b1 first travels with train t1 and then swaps to train t2 at node 2. The swap operation is represented in a dashed line, with the reference to the block. Notice that train t1 has to continue its route until node 4 due to the structure of the crew segment cs1, which is assigned to it. Block b1 is then carried by train t2, until it drops at yard 3. Moreover, train t2 picks up block b2 at node 3 and drops it at node 2. It could be possible for that train to take, at that point, the swap of b1 from train t1. However, since block b1 is meant to be delivered at node 3, the train itself would still have to go to node 4 and back again because of the structure of the crew segment assigned to it, which is cs2. One last observation regarding this solution is the existence of work events. Train t1 has only one work event, which corresponds to the block swap carried out at node 2. Note that picking up block b1 at node 1 is not considered to be a work event, since the train starts at that precise node. Additionally, train t2 has four work events corresponding to picking up and dropping off blocks b1 and b2, since these operations occur at intermediate yards.

In solution 2 only one train is routed. In this case, block b1 is set off at the same time as block b2 is picked up. These two operations, however, are accounted as only one work event.

As opposed to train scheduling, in which medium to long term decisions are made regarding train routes and timetables, train design dynamically establishes routes as a function of block requirements. Therefore, every train has a startup cost, denoted by TSC. Additionally, a variable cost is considered depending on the mileage traversed, designated as train travel cost, and denoted by TTC. Note that this cost does not depend upon whatever the train may be carrying at a certain moment. The distance each train travels is calculated as a sum of the mileage of each crew segment it traverses. The mileage of crew segment k is denoted by d(k). On the other hand, work events may occur along each train's route, according to what was previously described. Assume that a work event cost, denoted by WEC, is assigned to each of those operations.

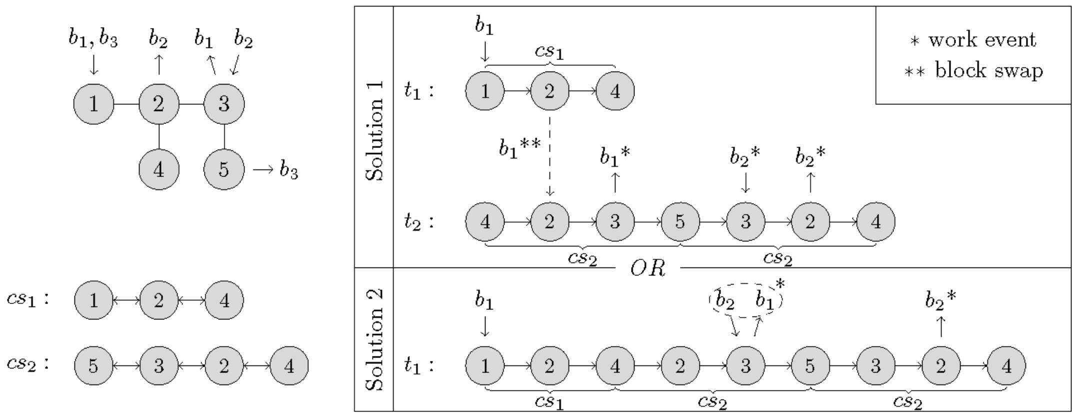

Figure 2. Example of Two Possible Train Designs in a Small Sized Instance

Also as mentioned, blocks may be swapped from one train to another. This operation incurs in what is designated as a block swap cost, denoted by BSC(i). Observe that this sort of cost depends on the yard where it takes place, due to different local configurations. When a block b is carried, a car travel cost, denoted by CTC(b), should be accounted. This cost varies for each block, according to the number of cars in the block. Note that this car travel cost is a per mile cost and should, therefore, be multiplied by the distance traveled. Additionally, there is a missed block cost, denoted by MBC(b), representing the loss given by the non-deliverance of a certain block b. Again, this cost depends on the number of cars in the block.

The objective of the train design problem is to find a solution minimizing the sum of the following costs:

i) Train startup cost (sum of this cost for all trains);

ii) Total train travel cost (product of the train travel cost by the mileage traversed, for each train);

iii) Total car travel cost (product of the car travel cost by the mileage traversed, for each block);

iv) Total work events cost (work event cost times the number of such operations);

v) Total block swap cost (sum of the block swap cost at each yard where such operation takes place);

vi) Total missed cars cost (sum of the missed car cost, for each block).

A feasible solution has to observe the following constraints:

i) The number of blocks carried in each train cannot exceed MBT, at each possible link;

ii) The number of swaps per block cannot exceed MBS;

iii) The number of work events per train cannot exceed MWE;

iv) The length of each train cannot exceed ML(a), at each possible link a;

v) The weight of each train cannot exceed MW(a), at each possible link a;

vi) The number of trains passing through link a cannot exceed TTL(a);

vii) Each train has to be operated by one crew, at each of its segments.

3. Heuristic

In this section we propose a metaheuristic to provide efficient solutions for the problem. This metaheuristic includes two levels of procedure, designated as upper and lower-level. In the lower-level, the heuristic's algorithm will iteratively consider the dispatch of blocks, between its corresponding pair origin/destination. For each block to be sent, a number of limited resources are used, as a result of train settings and crew assignments. The method finds the cheapest dispatch solution for each block, with no regard for the blocks that are yet to be considered, thus consisting in a greedy heuristic.

Since the decision upon the sequence by which the blocks are dispatched is crucial for the process of obtaining a good solution, an upper-level genetic algorithm is implemented such that it will sort through different sequences of blocks, so it can converge to one, leading to a low cost solution.

The lower-level procedure is defined over an auxiliary network, which is introduced in subsection 3.1, and which contains information about the possibilities for each block to be routed. In subsection 3.5, the problem of optimally sending the block is described, which has been identified as a slightly modified resource constrained shortest path problem. Finally, subsection 3.6 contains the description of the upper-level genetic algorithm.

3.1. Network Configuration

For each sequence of blocks provided by the upper-level genetic algorithm, the heuristic builds a feasible solution by sending a block at a time. To be able to send a block, some resources have to be available, or be made available. In particular, either a set of trains has to be already available, or new trains have to be created, in order to carry the new block to its destination. This operation has, of course, to obey the existing constraints, such as maximum block swaps or work events, as mentioned in the problem's description.

At a certain stage of the procedure, some blocks will have already been sent, which means that a set of trains has already been set up. These trains can still be used to carry further blocks. Moreover, new trains can also be set up to transport those blocks.

To model this situation, an auxiliary network is defined such that it represents all the feasible routes a new block may use, in order to be carried from its origin to its destination. This network is then used to solve the subproblem of optimally route the new block.

In this sense, nodes (i,k,t,u) are included in the network, representing the possibility of the new block being on board of train t, as it traverses yard i, operated by crew k, in its direction u (u will be 1 or 2, corresponding to both directions of the crew). Links are considered between those nodes standing for all the possible moves of a new block along the established trains, namely, being picked up or dropped off, being carried by a train from one yard to another or even being swapped between trains.

3.2. Network Initialization

The auxiliary network will also have to conceive the possibility of new trains to be set up. This means that besides the nodes associated with the existing trains, nodes also have to be included, such that they represent all the possible paths for new trains. Note that before dispatching the first block, no trains will have been set up, and therefore the network will only include these nodes.

To keep a homogeneous notation, nodes corresponding to those trains are denoted by (i,k,t,u), with $t = 0$. Since no *a priori* route exists for those trains, a set of nodes $\{(i_1,k,0,1); (i_2,k,0,1); ...; (i_{n(k)},k,0,1)\}$ is included for each crew k, in its away direction. Similarly, nodes $\{(i_{n(k)},k,0,2); ..., (i_2,k,0,2); (i_1,k,0,2)\}$ are included, corresponding to the home direction of crew k. These nodes are then linked so that each crew segment is laid out in the network. Trains that use these nodes will, often throughout the paper, be informally designated as 0-index trains. This terminology is used to characterize potentially new trains. In this section it will become evident that 0-index trains can represent, in fact, more than one train, or even an extension of an existing train.

To conclude the network initialization, crews have to be connected at shared yards. Therefore, for each yard i where at least two crews, k_1 and k_2, intersect, it is necessary to include links on the network, that describe a block swap. That purpose is achieved by inserting arcs between nodes $(i,k_1,0,u_1)$ and $(i,k_2,0,u_2)$, with u_1 and u_2 being 1 or 2, for the two possible directions of crews k_1 and k_2. Note that, if i is the first yard of crew path k_1, with respect to direction u_1, this means the block is to leave the crew path in its first node, and so, it wouldn't make sense to join it in the first place. Therefore, these arcs are not inserted for that case. The same holds, if i is the last yard in crew segment k_2, with respect to direction u_2. In that situation, since i is the last yard, a block could only be swapped to another crew k_3 or be dropped off. It's obvious that no optimal solution would use the arc between $(i,k_1,0,u_1)$ and $(i,k_2,0,u_2)$ under these conditions, since it would more efficient to swap directly from crew k_1 to k_3, or be directly dropped off by crew k_1. These exceptions will be considered throughout the network's construction.

In the general situation, $2\times(c\text{-}1)\times c$ arcs are inserted for each yard, where c is the number of crews that share that yard. This number may be reduced by the issue previously pointed out.

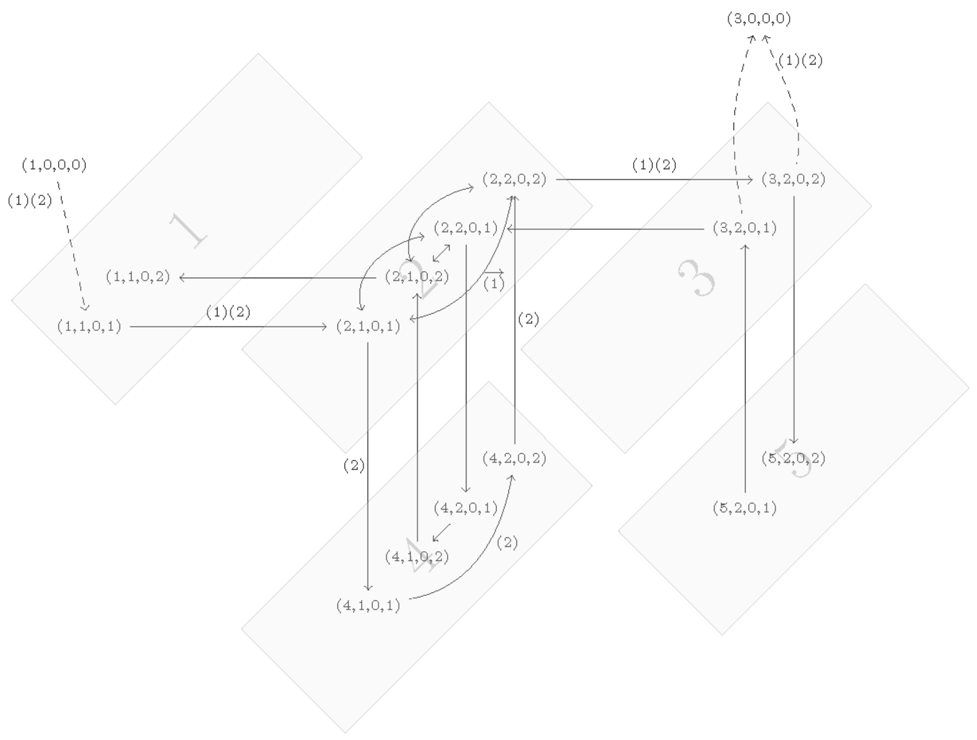

Figure 3. Heuristic Backbone Network for the Example in Figure 2

To illustrate the layout of the initial backbone network, consider again the instance in Figure 2. For this case, the backbone network is depicted in Figure 3, where nodes were included according to the argument above described. The arcs included at this stage are represented in solid lines. For each yard, a shaded rectangle is depicted to highlight the corresponding nodes. Additional information is also included, with respect to aspects focused in the following paragraphs.

When a block is considered, a dummy origin node and a dummy destination node are inserted. These nodes are represented by (o,0,0,0) and (d,0,0,0), where o and d are, respectively, the origin and destination yards of the block. These nodes are then connected to the other nodes sharing the same yards, in the correct directions.

For instance, consider that block b_1 is to be routed in the previous example. Accordingly, nodes (1,0,0,0) and (3,0,0,0) are included as well as arcs connecting those node with the backbone network. That situation is also depicted in Figure 3, where those arcs are represented in dashed lines. With the network composed as such, it is clear that routing the block consists in finding a path from the origin to the destination node. To illustrate that fact, two possible paths are indicated, which are denoted by (1) and (2).

Each path will induce not only information about the block route, but also information about the way trains have to be created. For example, path (1) includes an arc linking two different crews at yard 2. However, that yard is not an end point of any of the crews. As mentioned in the section on the problem’s description, trains have to be operated so that their routes match crew segments, from one end to the other. Then, the train used to carry the block up to yard 2, cannot be the one that continues the operation further on. As a consequence, another train has to be created.

In the case of path (2), the arc linking (4,1,0,1) to (4,2,0,2) is used. Since yard 4 is an end point for both crews, no new train is needed besides the one leaving yard 1, which continues the operation.

3.3. Network Expansion

After having sent each block, the auxiliary network has to be updated accordingly. That procedure has to occur, due to the fact that some resources may have been consumed, such as train length per link or number of blocks per train. If the maximum length, or the maximum weight, has been reached for a certain train in a certain link, as a consequence of the newly routed block, that specific link may no longer be used by subsequent blocks. Therefore, that link should be removed from the network.

Additionally, as a consequence of the block routing, some new trains may have been set up, or existing trains may have been extended. In this sense, the auxiliary network has to be expanded, in order to include nodes that represent the routes of the new trains, following the notation already introduced. Notice, however, that in some links the maximum number of trains may have been reached. In a case like that, no new train may use the link. Due to constraint vii), presented in the problem's description (see section 2), any crew segment that includes that link will have to be completely discarded for new trains.

Finally, after deciding the new block to be evaluated and after the origin and destination dummy nodes have been included, such as already indicated in the previous subsection, the characteristics (i.e. length and tonnage) of the new block may be such that some links become unusable. Since this restriction is block dependent, those links should not be removed, but locked. After routing the block, the links are unlocked again.

To illustrate this network update procedure, consider now that block b_1 was routed according to path (1) presented in figure 3. Denote by 1 and 2 the two new trains, whose creation is implied by that route. The auxiliary network must now be augmented due to that fact. In this case, the set of nodes $(i,k,1,u)$ and $(i,k,2,u)$ are inserted according to the crews that operate those trains. Furthermore, for each of those new trains, three groups of arcs have to be inserted, in order to properly connect the new nodes between themselves and the rest of the auxiliary network. The first group of arcs will connect each pair of sequential nodes, following the routes of the two new trains. The second group of arcs will connect the nodes representing each intermediate yard on the routes of those trains to other nodes that correspond to the same yard in another train. This represents the possibility for a new block to swap both between two existing trains and between an existing train and a new one. Finally, the third group of arcs will connect each of the end nodes of a train, to nodes that correspond to the same yard in a different train. This final group of arcs stands in fact for different types of situations. There are arcs both to other existing trains and to new trains. The latter, may represent the effective creation of a new train, but can also represent an extension of the existing train. Since these two cases hold, two parallel arcs are inserted in the network so these two situations are properly modeled. Extensions can occur both in the start point and the end point of a train, but can only be applied to trains that have not reached the maximum number of work events. Note that this type of links may need to be erased further on, if an extension does occur on the corresponding node, implying an update on the start or end point of the train.

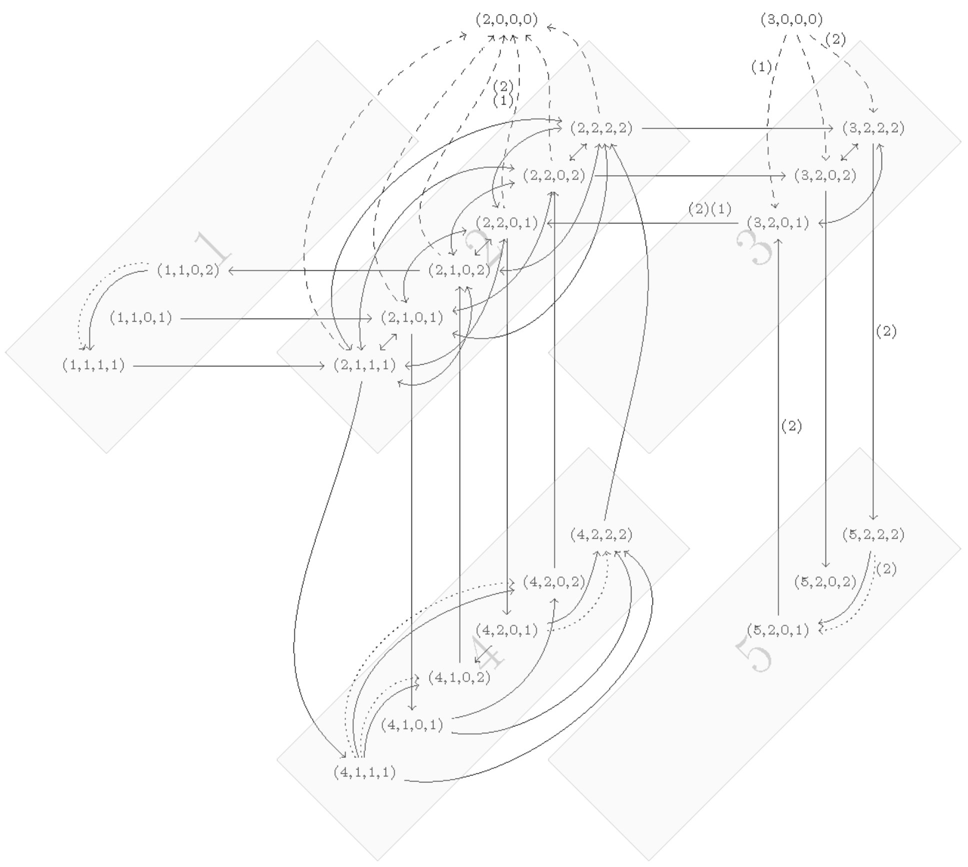

Figure 4. Heuristic Network After Routing Block b_1

The new sets of nodes and arcs are represented in Figure 4, including all the situations that were mentioned. This figure, as previously, points out two possible paths to route block b_2. If routed via path (1), block b_2 boards a new train at yard 3. Thereafter, the block is carried by that train to yard 2, where it is dropped off. The train then continues along the crew segment, up until yard 4. If, on the other hand, block b_2 follows path (2), existing train 2 picks up the block and carries it up to yard 2. Note that, an extension is taking place at yard 5, represented by the dotted arc from node (5,2,2,2) to node (5,2,0,1). One other important observation is that, if a similar path would be taken which, instead of using the mentioned dotted arc, would use the parallel solid arc, a new train would be created departing from yard 5 and operated by crew 2. In terms of the block's resources, this other solution would mean an extra block swap between train 2 and this new train.

Consider that block b_3 is the next to be evaluated. The options to route that block depend on the several conditions of the network thus far. It's important to highlight at this stage the role taken by the constraints mentioned in the problem's description. To illustrate the way some of those constraints are incorporated in the auxiliary network configuration, consider that the maximum number of trains traversing a link is equal to 3, with exception of the link between yards 2 and 3, where that bound is 2. Assume now that block b_2 was routed along path (2). That option leads to having two traverses on that link (in fact, made by the same train), which makes it impossible for that link to be used again. As a consequence, the arc from node (2,2,0,2) to node (3,2,0,2) and the arc from node (3,2,0,1) to node (2,2,0,1) are erased from the network. Additionally, since every train, when operated by a certain crew, has to travel necessarily through all its yards, from first to last, all other arcs corresponding to the same crew and the same direction should also be erased.

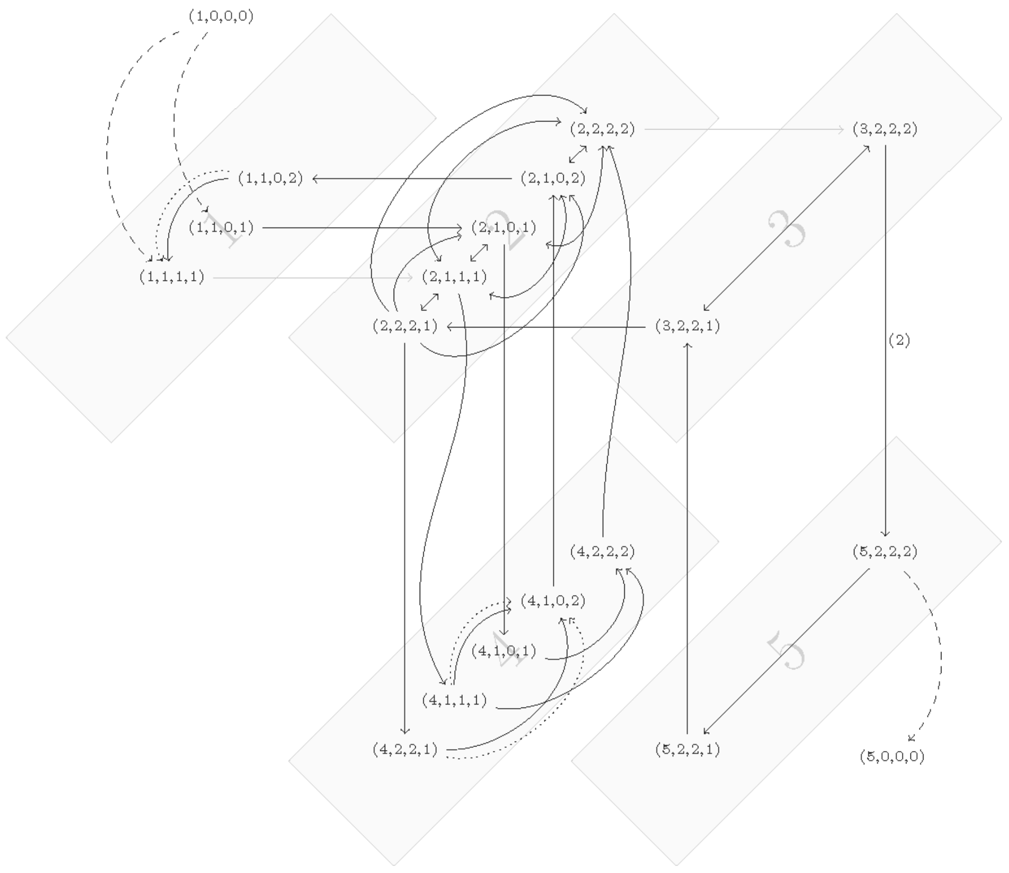

Figure 5. Heuristic Network after Routing Block b_2

Consider additionally that blocks have, respectively, a length of 20, 10 and 20 units. Admitting that trains cannot carry a length higher than 30, the third block cannot be in the same train, at the same time as the first one. This means that all links respecting trains already carrying block b_1 cannot be used to transport block b_3. Therefore, those links are locked for this particular step, meaning they cannot be used by this particular block but may become available later for a subsequent block with different characteristics, such as length or tonnage.

Figure 5 depicts the auxiliary network, incorporating all the new links, following the procedure previously explained, as well as the link removal above mentioned. Moreover, locked links are represented in light grey lines. This figure also includes the auxiliary links connecting the new block's origin and destination nodes. By observing this network, one cannot find a possible path between those two nodes. Hence, the block will have to be considered as missed.

This procedure continues until all blocks are evaluated. At the end of the procedure, a solution is obtained, both in terms of blocks as well as in terms of trains and crews.

3.4. Block Routing

As described in the previous sections, at each stage of the lower-level procedure, the route used to send a block has to be determined. That route should take into consideration the minimization of costs. As such, each link should have a cost, such that the total costs of the links used by a chosen path, represents the incremental cost of routing the block on the total operational cost. For example, if a block boards a previously created train, it doesn't make sense to charge again the train startup cost or the train travel cost.

Furthermore, when routing a block, some technical constraints should be satisfied. As seen above, some of those constraints, regarding, for instance, the maximum trains per link or the maximum length carried by a train, can be dealt with when designing the network. However, the upper bound on the number of block of swaps and on the number of work events per train can't be computed offline. It wouldn't be possible to make *a priori* decisions on the arcs since the information on any of those two bounds depend on the actual route taken by the block, which is computed during the procedure. Therefore, it becomes necessary to include this information into the routing problem. Adequate values of the consumption of these resources should be assigned to each arc, along with costs. In that sense, consider that $r^{BS}(i,j)$ and $\mathrm{r}^{\mathrm{WE}}(t_k,i,j)$ are, respectively, the amount of block swaps and work events that are "consumed" when traversing arc (i,j) in the auxiliary network.

The objective is to find a minimum cost path from node $(o,0,0,0)$ to node $(d,0,0,0)$ such that resource constraints are met. This problem, formulated as such, is a resource constrained shortest path problem (RCSPP). However, as will be explained later, the problem here studied is in fact a slight variation of the RCSPP.

With respect to work events, one relevant aspect is that the information about where work events are taking place determines how those resources are "used" in a certain link. That is, if a block is routed, such that it swaps from an intermediate yard of a certain train, the link representing the swap will only charge a work event if no other was taking place there, previously. In what concerns the block swap information, no similar issue arises, since that constraint is exclusively related with the block.

Table 2 contains costs, as well as resource values, for each group of arcs, referenced by a capital letter, presented in the first column. The second column contains information about each specific group of arcs. These groups were created according to different configurations, in terms of costs and/or resource values. For simplicity, the set of arcs belonging to a certain type X will be referred to as A_{X}. The third column in the table, designated by I(k), indicates the set of crews for which the yard concerning the arc is an intermediate one. Similarly, the fourth column, designated by I(t), indicates if some of the yards are intermediate nodes on the global path of each train. In both of these columns, if the cell is blank, it means that the yard is an endpoint, respectively, of a crew segment or of a train route. Otherwise, in case a cell is shaded, it means that the information about the position of the yard is irrelevant. The fifth column contains information about the role of each group of arcs, following the description already mentioned. The sixth column indicates the cost of each arc, which is the result of the several types of costs. The seventh column indicates whether or not a block swap occurs. Finally, the eighth column displays the set of trains for which a work event is accountable.

When assigning costs and resource consumptions to arcs, the issue about extension emerges. As previously observed, extensions may arise forward or backward. To illustrate this question, figure 6 depicts a generic situation where extensions might happen. Consider, for instance, that the block is routed such that its path contains a segment from node (i,k_1,t_1,u_1) to node (j,k_4,t_2,u_4). Note that there are two situations containing parallel arcs, which correspond to the extensions previously pointed out. Therefore, the block's route may take, in each case, one of the two options. If the block uses arc a, in principle this would mean that a new train would be created. However, if arc *d* is used by the path, at the end of the segment, then a backward extension is considered. Hence, the creation was not exactly so, but instead a swap operation from train t_1 to train t_2. When assigning costs, a train startup cost has to be associated to arc *a*. However, if the situation as explained occurs, and the effective creation does not hold, that cost has to be canceled. As a mean to achieve that goal, a negative value of TSC is assigned to arc *d*. Still, this is not a definite case, since it may also happen that arc *d* is used with the segment beginning with arc *b*. In that case, no train startup cost should be canceled for there was no such cost assigned to arc *b*, in the first place. Consequently, this specific cost has to be considered as conditional. This condition can only be verified during the execution of the shortest path algorithm.

One other possible combination in the mentioned segment includes arcs *b* and *d*. In this case, two extensions are occurring, in a certain sense. This means that the two trains, t1 and t2, end up being merged via crew segment k_2. Note that this solution is only feasible if the upper bound on work events per train is not surpassed by the sum of work events in t_1 and t_2.

$$\begin{tikzcd}
& & & (h,k_3,0,u_3) & \\
& & & (h,k_2,0,u_2) \arrow[u, "e"] \arrow[dr] & \\
{} \arrow[r, dashed] & (i,k_1,t_1,u_1) \arrow[r, shift left, "a"] \arrow[r, shift right, dashed, "b"'] & (i,k_2,0,u_2) \arrow[ur] & & (j,k_2,0,u_2) \arrow[d, bend right, "c"'] \arrow[d, bend left, dashed, "d"] \\
& & & & (j,k_4,t_2,u_4)
\end{tikzcd}$$

Figure 6. Representation of a Problematic Situation in the Auxiliary Network

The other two possible combinations represent simple forward or backward extension of, respectively, t_1 and t_2. Note that, if a previous block was either being picked up or dropped off at an end point of an existing train, a work event should be considered in the link corresponding to that extension. This situation is handled by including a work event "consumption", respecting the corresponding train, in the link associated with it.

In generic terms, arcs *a*, *b*, *c* and *d* belong to groups of arcs I, H, K and L included in Table 2. One other important issue concerns the way work events are properly assigned during the execution of the shortest path algorithm. Take as an example, one other possible route for the block in figure 6 that includes arc *e*. Since yard *h* is intermediate in crew segment k_2, this situation induces a work event on the train dropping off the block. The question is how to know which train is dropping off the block, due to the ambiguity of 0-index trains already highlighted by the previous situation. Specifically, if the path also includes arc *a*, it means that the work event happens in a new train. However, the work event happens on train t_1, if the path includes arc *b*. A first approach to model this issue would be considering all possible combinations for this kind of arcs, thus enabling the proper assignment of work events to trains. However, this would mean enlarging the network, since the ambiguity of the 0-index trains would have to be deconstructed, therefore significantly increasing the dimension of the auxiliary network. To overcome the additional computational effort which would result from that approach, we use an online approach, in the sense that information is incorporated, which will be updated during the execution of the shortest path algorithm. Note that this work event concern may also be extended to the train that picks up the block, whenever the yard is also an intermediate one in the corresponding crew.

The issues above reported are responsible for the fact that the problem of routing a block is in fact a slight variation of the RCSPP. Also, as a consequence, the ambiguity concerning the 0-index train implies some additional parameters to effectively control resources usage. To start with, each arc contains information about the consumption of work events with respect to new trains. That means that $r^{WE}(t_k,i,j)$ includes information for all trains t_k, with $k \in T \cup 0$. However, since the 0-index trains may, in fact, correspond to train extensions, which means that work events should be accounted for the respective extended train, an additional attribute will have to be appended with information about eventual train extensions. This situation will be further explained in the next subsection, in the context of the proposed algorithm. One additional aspect, which has to be carefully observed, is that in certain cases links connect nodes associated with somewhat different 0-index trains. This happens, for instance, in link e on Figure 6. Because yard h is intermediate in crew segment k_2, if the block proceeds through link *e*, a different new train has to be created. So that link involves necessarily two trains, and work events may be differently accounted for each of those trains. Since no predefined information exists about the trains in stake, besides information about how much of work events is consumed for each of the existing trains, two additional consumption values have to be added, concerning the two new trains. Therefore, one has to consider parameters $r^{WE}(t_{01},i,j)$ and $r^{WE}(t_{02},i,j)$ to serve that purpose, which enables a proper application of the algorithm proposed in the following subsection.

3.5. Path Algorithm

As indicated in the previous section, at each stage of the lower-level procedure, a shortest path problem has to be solved. This problem is, in fact, a resource constrained problem where resources are block swaps and work events. Many algorithms conceived for path problems have been based on labelling techniques, since the seminal paper by Dijkstra [9] introduced that sort of technique for the shortest path problem (see Ahuja *et al.*, [3] for some thorough review on labelling techniques for path problems). As usual among those algorithms, resource labels are attached to nodes and updated throughout the algorithm with information about the total consumption of each resource, so that constraints are met. Consider that l represents a label associated with node i, which is denoted by $(i;c_l;r_l;p_l)$, where c_l represents the cost of the path from the origin node up to node i, r_l represents the total amount of consumed resources by that path, and p_l represents the predecessor label of i on the path. The resource element of the label is, in turn, composed by $r^{BS}(l)$, regarding the number of times the block has swapped between trains in the route up to that node, and by $r^{WE}(t_k,l)$, with respect to the number of work events occurred on train t_k, with k belonging to the set of existing trains, which will be denoted by T.

As suggested by Irnich and Desaulniers [12], the RCSPP requires that more than one label is attached to each node, since a cheaper path might end up being unfeasible. In that sense, a set L_i of labels has to be considered in each node. The algorithm that finds a minimum cost constrained path starts by setting the label of the current block's origin, which is first considered to be unchecked. The values assigned to the first label reflect the non-existence of block swaps for the current block, and additionally the total number of events registered for each of the existing $|T|$. That vector with the number of work events is denoted by w^T. The algorithm keeps iterating over the unchecked labels, and updating the label's successors. The procedure stops when no further unchecked label exists. The set of unchecked labels is denoted by UL. The structure of this algorithm is outlined in the pseudo-code designated by RCSP(b^*), where b^* is the block to be sent.

Algorithm 1: Resource constrained shortest path (RCSP(b^*))

```
$l \leftarrow (o(b^*); (0; w^T(t_1), \ldots, w^T(t_{|\mathcal{T}|})); 0)$        /* Initialize labels */
Associate $l$ with node $(o(b^*), 0, 0, 0)$
$UL \leftarrow l$
while $UL \neq \emptyset$ do
    $l^* \leftarrow CL(UL)$        /* Choose an unchecked label */
    $UL \leftarrow UL - \{l^*\}$        /* Remove the label from the set of unchecked labels */
    SLU($l^*$)        /* Update the successors of the node associated with the chosen label */
```

In the RCSP algorithm, when updating labels, the concept of path dominance arises differently than in a simple unconstrained path setting. A label l_1 is said to dominate another label l_2 if the cost and resource values in l_1 are not higher than the corresponding value in l_2, and at least in one of the cases, a strict relationship holds.

In the SLU algorithm, which updates the successor labels of a certain label l^*, the resulting new label is tested against the ones already assigned to the successor. If one of those labels is dominated by the new label, it should be disregarded. On the other hand, if the new label is dominated by some existing label, then the update procedure stops and the new label is disconsidered.

Algorithm 2: Successors' labels update (SLU(l^*))

```
$i^* \leftarrow n(l^*)$
foreach $j \in \Gamma^+(i^*)$ do
    if $r_{l^*}^{BS} + r_{(i^*,j)}^{BS} \leq MBS$ and $r_{l^*}^{WE_{t_k}} + r_{(i^*,j)}^{WE_{t_k}} \leq MWE, \forall k \in \mathcal{T}$ then
        $l \leftarrow (y(j); c_{l^*} + c_{(i^*,j)}; r_{l^*} + r_{(i^*,j)}; l^*)$;
        label_enters_list $\leftarrow$ $true$;
        foreach $l_j \in L_j$ do
            if $l_j$ dominates $l$ then
                label_enters_list $\leftarrow$ $false$
            else if $l$ dominates $l_j$ then $L_j \leftarrow L_j - l_j$ ;
        if label_enters_list $= true$ then $L_j \leftarrow L_j \cup l$
        $UL \leftarrow UL \cup l$ ;
```

Due to reasons already pointed out in the previous subsection, updating work events information may not be as plain as the previous label update algorithm could induce. For those pair of nodes, where two parallel links exist corresponding, respectively, to the creation of a new train or to the extension of an existing one, the resulting dominated label has to be saved for an eventual need to recover it afterwards. To achieve that purpose, Δ_{l^*} is used to represent the eventual dominated label, dropped at some point in the condition before mentioned. If no dominated label is associated to l^*, then $\Delta_{l^*} = \emptyset$. Additionally, information about an eventual situation of train extension has to be aggregated to each label. Thus, let ET_{l^*} be a set which contains the reference to the train being extended, or $\emptyset$ otherwise.

The pseudo-code in algorithm 3, designated by WEU, illustrates the procedure used to correct, at each label's update, the information regarding work events. Before describing in detail that code, it's important to understand that sequences of train extensions may happen in a certain path. This means that additional information has also to be associated with a label, concerning the possible train merge situation. Let MT_l represent the set of merges taken place in the path induced by label l. The set MT_l is composed by subsets of trains.

The algorithm WEU first tries to identify if arc (i^*,j) corresponds to a situation which, as previously mentioned, demands special attention. If the train associated with node j has an index 0, there are two situation that should be considered: i) a train is created in the arc (i^*,j), and ii) a train is extended in the arc (i^*,j). In the first situation, if the train associated with i^* is an extended one, it is necessary to update the information regarding the number of work events in such extended train. Additionally, the algorithm has to update the information on trains that may have been previously merged with the extended train. Finally, as a new train is created, such that it may or may not be involved in a backwards extension in a further moment, it is necessary to keep the information about an eventual work event ($r^{WE}(0,l)$). In the second situation, where arc ($i*$,j) stands for an extension, the information about what train is being extended is saved on ET_{l*}. If neither of these situations occur and if the train associated with the start node is an extended one, the algorithm updates variables ET_{l*} and $r^{WE}(0,l)$, assigning them with the values held by the same variables in the predecessor label.

If, on the other hand, the train associated with node j is an existing one, we have to take special care with arcs representing a backwards extension. This backwards extension, as explained in the previous subsection may, however, be itself one of two situations: i) an extension of a 0-index train, that was itself, an extension of another train (*e.g.*, $b + d$ in figure 6), or ii) an extension of a train that was supposedly created in a previous iteration of the RCSP($b*$) (*e.g.*, $a + d$ in figure 6). In the first case, the two trains are merged, and MT_l has to be updated. Also, it is necessary to update the number of work events on all the trains that are being merged. As for the second case, as a TSC had been previously charged when the train was thought to have been created, it is now necessary to cancel such cost. Finally, the number of work events of the backwards-extended train has to been updated, with the eventual work event done on the 0-index train, assigned to $r^{WE}(0,l^*)$.

Algorithm 3: Work events update (WEU(l^*, i^*, j))

```
if $t(j) = 0$ then
    if new train is created on arc $(i^*, j)$ then
        if $ET_{l^*} \neq \emptyset$ then
            $r_l^{WE_{t_k}} \leftarrow r_l^{WE_{ET_{l^*}}} + r_{(i^*,j)}^{WE_{0_1}}, \forall t_k \in mt : mt \in MT_l$ and $mt \ni ET_{l^*}$
            $\mathbf{max\_no\_WE} \leftarrow \max\{\mathbf{max\_no\_WE}, r_l^{WE_{ET_{l^*}}}\}$
        $r_l^{WE_0} \leftarrow r_{(i^*,j)}^{WE_{0_2}}$
    else if train is extended on arc $(i^*, j)$ then
        $ET_l \leftarrow t(i^*)$
    else if $ET_{l^*} \neq \emptyset$ then
        $ET_l \leftarrow ET_{l^*}$
        $r_l^{WE_0} \leftarrow r_{i^*}^{WE_0}$
        $r_l^{WE_{t_k}} \leftarrow r_l^{WE_{ET_l}} + r_{(i^*,j)}^{WE_{0_1}}, \forall t_k \in mt : mt \in MT_l$ and $mt \ni ET_{l^*}$
        $\mathbf{max\_no\_WE} \leftarrow \max\{\mathbf{max\_no\_WE}, r_l^{WE_{ET_l}}\}$
else
    if train is extended on arc $(i^*, j)$ then
        if $ET_{l^*} \neq \emptyset$ then
            if exists $mt \in MT_l : ET_{l^*} \in mt$ then
                $mt \leftarrow mt \cup t(j)$
            else if exists $mt \in MT_l : t(j) \in mt$ then
                $mt \leftarrow mt \cup ET_{l^*}$
            else
                $MT_l \leftarrow MT_l \cup \{\{ET_{l^*}, t(j)\}\}$
            $r_{l^*}^{WE_{t_k}} \leftarrow r_{l^*}^{WE_{t(j)}} + r_{l^*}^{WE_{ET_{l^*}}}, \forall t_k \in mt : mt \in MT_l$ and $mt \ni t(j)$
        else
            $c_l \leftarrow c_l - TSC$
            $r_l^{WE_{t_k}} \leftarrow r_l^{WE_{t(j)}} + r_{i^*}^{WE_0}, \forall t_k \in mt : mt \in MT_l$ and $mt \ni t(j)$
        $\mathbf{max\_no\_WE} \leftarrow \max\{\mathbf{max\_no\_WE}, r_l^{WE_{ET_{t(j)}}}\}$
    else if $ET_{l^*} \neq \emptyset$ then
        $r_l^{WE_{ET_{t_k}}} \leftarrow r_l^{WE_{ET_{l^*}}} + r_{i^*}^{WE_0}, \forall t_k \in mt : mt \in MT_l$ and $mt \ni ET_{l^*}$
        $\mathbf{max\_no\_WE} \leftarrow \max\{\mathbf{max\_no\_WE}, r_l^{WE_{ET_{l^*}}}\}$
```

The global variable *max_no_WE* indicates the maximum number of work events in which any train occur, in each iteration. Evidently, this value has to be not higher than the MWE upper-bound. Therefore, every time the number of work events of any train is updated, so is *max_no_WE*.

Due to these special cases, SLU algorithm needs to be updated in order to accommodate for the WEU algorithm, and in order to assign the Δ_{l^*}, when needed. The revised version is presented in algorithm 4.

Algorithm 4: Successors' labels update (SLU'(l^*)) - Revised Version

```
1  $i^* \leftarrow n(l^*)$
2  $\Delta_l \leftarrow \emptyset$
3  foreach $j \in \Gamma^+(i^*)$ do
4      max_no_of_work_events $\leftarrow \max_{k \in \mathcal{T}}(r_{l^*}^{WE_{t_k}} + r_{(i^*,j)}^{WE_{t_k}})$
5      $l \leftarrow (j; c_{l^*} + c_{(i^*,j)}; r_{l^*} + r_{(i^*,j)}, l^*)$;
6      if $(t(j) \neq 0$ $and$ $not$ $Is\_Train\_Extended((i,j^*)))$ $or$ $Is\_New\_Train\_Created((i,j^*))$ then
7          $\Delta_l \leftarrow \emptyset$
8      else
9          $\Delta_l \leftarrow \Delta_{l^*}$
10     max_no_of_work_events $\leftarrow$ WEU($i^*, j$)
11     if $r_{l^*}^{BS} + r_{(i^*,j)}^{BS} \leq MBS$ $and$ max_no_of_work_events $\leq MWE$ then
12         label_enters_list $\leftarrow true$;
13         foreach $l_j \in L_j$ do
14             if $l_j$ dominates $l$ then
15                 label_enters_list $\leftarrow false$
16                 if $((i*,j)_l \in A_I$ $and$ $(i*,j)_{l_j} \in A_H)$ $or$ $((i*,j)_l \in A_L$ $and$ $(i*,j)_{l_j} \in A_K)$ then
17                     $\Delta_{l_j} \leftarrow l$
18             else if $l$ dominates $l_j$ then
19                 $L_j \leftarrow L_j - l_j$
20                 if $((i*,j)_l \in A_H$ $and$ $(i*,j)_{l_j} \in A_I)$ $or$ $((i*,j)_l \in A_K$ $and$ $(i*,j)_{l_j} \in A_L)$ then
21                     $\Delta_l \leftarrow l_j$
22         if label_enters_list $= true$ then $L_j \leftarrow L_j \cup l$
23         $UL \leftarrow UL \cup l$ ;
24     else if $\Delta_l \neq \emptyset$ then
25         $UL \leftarrow UL \cup \Delta_l$
```

It is now possible to describe the lower-level procedure in more general terms, designated by LLP, such as depicted in algorithm 5. The first step in the algorithm is to create the initial backbone network and initializing the total cost to 0. Afterwards, following the order *B* received as input, a path is sought for each block, from its origin to its destination, respecting all constraints. For this to be possible, each iteration starts by preparing the network for the block's characteristics by: i) updating the block travel cost in arcs belonging to group E (see Table 2), ii) replacing the origin and destination node from the previous block by the ones from the new block, and connecting them to the network, iii) locking links that the block can't use due to length and weight constraints, and iv) removing all the previous labels from the network's nodes. Finally the network is ready, and the RCSP algorithm is called. This procedure will generate every feasible path between the block's origin node and destination node. That set is denoted by *P*(*b*), where *b* denotes the block. Each of the paths in *P*(*b*) is associated with a label in the block's destination node. Each path's route can be recursively obtained, following each label's predecessor.

```
Algorithm 5: Lower-level procedure (LLP($B$))
1  $N \leftarrow Create\_Backbone\_Network$
2  total_cost $\leftarrow 0$
3  foreach $b \in B$ do
4      $Prepare\_Network(N, b)$
5      $RCSP(b)$
6      $P(b) \leftarrow Recover\_Paths(N, b)$
7      foreach $\rho \in P(b)$ do
8          $\rho \leftarrow Correct\_Path(\rho)$
9          $\rho \leftarrow Improve\_Path(\rho)$
10     if $P(b) \neq \emptyset$ then
11         $\rho^* \leftarrow \rho : \min_{\rho \in P(b)} c_\rho$
12         total_cost $\leftarrow$ total_cost $+ c_{\rho^*}$
13         $Update\_Network(N, \rho, \mathcal{T}, b)$
14     else
15         $missed_b = true$
16         total_cost $\leftarrow$ total_cost $+ MBC(b)$
```

If more than one path exists between the block's origin and destination nodes, the choice of which to use will reside in the one offering a cheaper cost. However, the network isn't able to model some improvement gaps. To illustrate those situations, consider again figure 4 where route (2) is one possibility for block b_2 to follow. This route forces the extension of train 2, operated by crew 2, in yard 5. However, this extension can happen independently of carrying at that moment this new block, which could be picked up on the extended segment. This other solution clearly saves the mileage from yard 3, in node (3,2,2,2), to yard 5, and back again to yard 3, on node (3,2,0,1). This type of situation, depicted in figure 7, could not be efficiently modeled by the network, for it would mean that more than one arc was required for the pickup operation, associated with different possibilities of previous extensions. Therefore, using a post-processing improvement procedure, designated by *Correct_Path*, was considered to be more adequate. That procedure seeks for possible corrections similar to the ones used as an example.

$$\begin{tikzcd}
(3,0,0,0) \arrow[r, dashed] & (3,2,2,2) \arrow[r] & (5,2,2,2) \arrow[r, dotted] & (5,2,0,1) \arrow[r] & (3,2,0,1) \arrow[r] & (2,2,0,1) \arrow[r, dashed] & (2,0,0,0) \\
(3,0,0,0) \arrow[rrrr, "\text{Train } t_2 \text{ is extended}"] & & & & (3,2,0,1) \arrow[r] & (2,2,0,1) \arrow[r, dashed] & (2,0,0,0)
\end{tikzcd}$$

Figure 7. Example of a Corrected Path

In a similar situation, *Improve_Path* tries to avoid the creation of new trains, by extending a train that end its route on the yard where the new train's route would supposedly begin. This situation arises only when a block boards the new train in an intermediate yard of the crew, which can't be defined directly on the network as extension links only connect end yards of a certain crew.

Finally, assuming that there is at least one feasible path, the one with the lowest cost is chosen. The total cost is updated, and information is saved regarding the block's chosen route. It is, then, necessary to update the information with respect to the trains carrying the block. If new trains are created, or old ones are extended, the network must be updated in order to represent the new possibilities. Lastly, it is necessary to evaluate the block's impact on resource's consumption. If any links in the network reaches its limit of any resource, it must be removed, for it can't be used by further blocks. Consequently, some crew segments may be erased (see subsection 3.1). If, on the other hand, there is no feasible path, the block is considered to be missed, and the according missed block cost is added to the total cost.

The procedure continues until all the blocks in the permutation are evaluated.

3.6. Upper level Genetic Algorithm

As stated at the beginning of this section, the order by which the blocks are considered will have a great impact on the quality of the final solution. For this reason, it is important to analyze different permutations of blocks, in order to try to converge to one with the lowest possible cost. From now on, a permutation of blocks will be referred as an upper-level solution, and will be represented by $B = \{b_1, b_2, ..., b_{|B|}\}$, where each b_i belongs to the set of blocks B. Based on the impact caused by varying the permutations, a genetic algorithm is recommended, and as such, one was devised according to the usual structure of those methods [21]. Therefore, the upper level procedure, based on a genetic algorithm, is presented in algorithm 6.

```
Algorithm 6: Upper-level procedure
1  $\mathcal{P} \leftarrow Create\_Initial\_Population(\mathcal{B}, population\_size)$
2  foreach $B \in \mathcal{P}$ do
3      $\mathbf{total\_cost}_B \leftarrow LLP(B)$
4  while $Stop\_Criterion = true$ do
5      $\mathcal{P}' \leftarrow$
6      while $| \mathcal{P}' | \leq population\_size \times (1 - crossover\_rate)$ do
7          $\mathcal{P}' \leftarrow Selection(P, \mathbf{total\_cost})$
8      while $| \mathcal{P}' | \leq population\_size$ do
9          $parent_1 \leftarrow Selection(P, \mathbf{total\_cost})$
10         $parent_2 \leftarrow Selection(P, \mathbf{total\_cost})$
11         $\mathcal{P}' \leftarrow Crossover(parent_1, parent_2)$
12     $number\_of\_mutations \leftarrow 0$
13     foreach $B \in \mathcal{P}$ do
14         if $Generate\_Random\_Number([0, 1]) \leq mutation\_rate$ then
15             Mutate(B)
16     foreach $B \in \mathcal{P}$ do
17         $\mathbf{total\_cost}_B \leftarrow LLP(B)$
```

In each iteration (or generation) a new population is created, consisting of a set of upper-level solutions with size *population_size*. The initial population is filled with randomly generated upper-level solutions. It is also recommended that this initial population is filled with upper-level solutions that follow some criterion, for instance one which the blocks that are dispatched first are the ones with a higher number of cars or with a larger distance between the origin and destination yard. Each of the following generations of populations will be constituted by solutions created by two different processes. The first process is a simple copy of solutions from a previous generation. In the second process, each new upper-level solution, commonly known as the offspring, is generated via the crossover (or mating) of two solutions of the previous generation, known as parents. As each upper-level solution is a permutation, it is necessary to use an according crossover operator. For this upper-level algorithm the chosen crossover operation was the order crossover (OX), by Davis [8]. Each

OX operation, between two parent solutions, generates two offspring, which preserve the relative ordering of the parents. The ratio of solutions created by each of these two processes is determined by the *crossover_rate*.

Note that in both these processes it is necessary to select which solutions from the previous generation are to be copied or mated. The selection method used in this upper-level algorithm is the tournament selection ([6],[11]), so that solutions with lower costs are more probable to be selected. Also, to ensure elitism, the best solution in each generation is always copied to the next one.

Finally, in order to ensure some diversity along different generations, and, as such, avoiding that the genetic algorithm gets stuck in some local optima, each solution in each population is mutated with a probability of *mutation_rate*. The mutation operator chooses two random positions in the permutation and exchanges the blocks occupying those positions.

New generations are created until a certain criterion is met. For example, this criterion can be a maximum number of generations, or some sort of convergence check. For this algorithm we propose the use of a convergence check, which stops the procedure if there is no change in the value of the best solution in the pool for a certain number of iterations. In the next section, the various parameters of the genetic algorithm will be the subject of the presented tests.

4. Results

In order to improve the performance of the algorithm, some tests were done with different parameters of the Genetic Algorithm. These parameters include the crossover rate, mutation rate, pool size and the number of iterations in which the best value doesn't change that take for the genetic algorithm to stop (*convergence*(*iter*)). Some tests were made with adaptive crossover and mutation rate, which increase if the genetic algorithm is apparently stuck in some local optimum (*).

Table 1. Test Results

pool size	Convergence(iter)	Crossover rate	Mutation rate	Best cost	Avg cost	iter	Time(sec)
10	50	0.6	0.05	422437.31	562557.1	90.9	35
10	100	0.6	0.05	409596.75	496816.9	181	98.8
10	150	0.6	0.05	399791.80	474835.8	218.5	172.4
10	200	0.6	0.05	426611.98	472761.0	224.9	456.1
10	200	0.7	0.1	463945.03	489391.1	182.5	36.1
10	200	0.6~0.7(*)	0.05~0.1(*)	411632.83	491760.4	244.2	17.9
10	200	0.6~0.8(*)	0.05~0.1(*)	409596.75	488196.3	217.6	15.6
10	200	0.6~0.9(*)	0.05~0.1(*)	403283.66	490419.9	236.2	16.4
20	200	0.6~0.9(*)	0.05~0.1(*)	404283.05	432080	232.6	50
30	200	0.6~0.9(*)	0.05~0.1(*)	494239.93	474132.84	290.2	80.6

The first instance tested had 10 yards, a network density of 10% and 4 crew segments. Network density represents the usual proportion between the actual number of arcs and the maximum possible number of arcs in the network (see [22]). The number of blocks to be sent was 20 and crew segments were randomly placed in the network such that each corresponds to a certain pair of nodes. Even with this small instance, the number of possible combinations of train and block routes is huge. Some combinations of parameters were used to test the heuristic algorithm with this instance. For each combination, the algorithm was executed 20 times. Table 1 indicates the average cost (*avg cost*) obtained for each combination of parameters. Also, the average number of iterations (*iter*) and time it took for the algorithm to end are presented.

To have a basis for comparison, we randomly generated 20 different upper-level solutions (i.e., orders by which the blocks are dispatched), which led to 20 different feasible solutions. The average cost among those solutions was 1512339.5. Taking this into consideration, it is possible to observe that the algorithm apparently manages to converge to a good solution. Due to the great complexity of the problem, it was impossible to obtain a lower bound based on the linear relaxation of the problem's formulation for such an instance dimension. However it is possible to conclude that a heuristic solution can be obtained in a fair amount of time. Testing some larger instances, it is possible to observe, for example, that the algorithm takes less than 3 minutes to reach a heuristic solution to an instance with 20 yards, network density of 10%, 30 crew segments and 30 blocks.

The same test was tried with an instance with 50 yards, a network density of 10%, with 60 crew segments and with 60 blocks. However, the algorithm was only able to generate a low number of solutions after a large amount of time. This can be explained by the huge number of possible paths in the network, which have to be evaluated for each permutation of blocks, given by the genetic algorithm. Moreover, the number of possible permutations of 60 blocks is evidently enormous, which makes for the large amount of running time.

Nevertheless, the algorithm can be still useful for obtaining a small number of feasible of solutions, where at least the first blocks on the corresponding permutations have high probability of being sent from their origin to their destination. In these high complexity instances, where only a few permutations can be evaluated in a reasonable amount of time, it may be convenient to generate the upper-level solutions with the help of some strategic criterion. For example, it may be interesting to evaluate solutions, where blocks with higher missing costs are located in the first positions of the network, for these have the highest probability of being sent.

The algorithm was implemented on java language and all tests were run on a Intel(R) Core(TM) i5 @ 3.30Ghz with 8GB RAM.

5. Conclusions

The article introduced a new integrated approach to the problems of Train Routing and Block-to-Train Assignment, called Train Design. In this problem, trains need to be set up and routed such that they carry a set of blocks from their respective origin to destination.

A metaheuristic was proposed in order to produce efficient and feasible solutions to this problem. In this metaheuristic, a genetic algorithm is used to generate orders for block dispatching. For each combination generated by the genetic algorithm, a sequence of resource constrained path problems have to be solved, in a network that encompasses all the possible routes for each block. It was pointed out that, in fact, that subproblem is a slight variation of the traditional RCSPP, in the sense that some particular dominated labels should be stored, as the label it dominates may become unfeasible in a further iteration of the algorithm.

The generation of efficient feasible solutions showed to be reasonably fast, and the rate of improvement was visible. However, a lower bound generating method should be an important tool to better evaluate the relative quality of the solutions obtained.

Acknowledgements

This work of João Telhada is supported by National Funding from FCT - Fundação para a Ciência e a Tecnologia, under the project: PEst-OE/MAT/UI0152. The work of Martim Moniz is supported by the Interuniversity Attraction Poles Program of the Belgian Science Policy Office.

References

[1] R. K. Ahuja, K. C. Jha and J. Liu, "Solving real-life railroad blocking problems", Interfaces, vol. 37, no. 5, **(2007)**, pp. 404-419.
[2] R. K. Ahuja, J. Liu, J. B. Orlin and D. Sharma, "Solving real-life locomotive scheduling problems", Transportation Science, vol. 39, no. 4, **(2005)**, pp. 503–517.
[3] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, "Network Flows: Theory, Algorithms and Applications", Prentice-Hall, Upper Saddle River, NJ, **(1993)**.
[4] C. Barnhart, H. Jin and P. H. Vance, Railroad blocking: A network design application, Operations Research, vol. 48, no. 4, **(2000)**, pp. 603–614.
[5] L. D. Bodin, B. L. Golden, A. D. Schuster and W. Romig, "A model for the blocking of trains", Transportation Research Part B: Methodological, vol. 14, no. 1-2, **(1980)**, pp. 115–120.
[6] A. Brindle, "Genetic algorithms for function optimization", Doctoral dissertation and Technical Report TR81-2, Department of Computer Science, University of Alberta, **(1981)**.
[7] J.-F. Cordeau, P. Toth and D. Vigo, "A survey of optimization models for train routing and scheduling", Transportation Science, vol. 32, no. 4, **(1998)**, pp. 380-404.
[8] L. Davis, "Applying Adaptive Algorithms to Epistactic Domains", Proceedings of the International Joint Conference on Artificial Intelligence – IJCAI85, vol. 1, **(1985)**, ppp. 162–164.
[9] E. W. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, vol. 1, **(1959)**, pp. 269-271.
[10] M. J. Dorfman and J. Medanic, "Scheduling trains on a railway network using a discrete event model of railway traffic", Transportation Research Part B: Methodological, vol. 38, no. 1, **(2004)**, pp. 81-98.
[11] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms", Urbana, vol. 51, **(1991)**, pp. 61801-2996.
[12] S. Irnich and G. Desaulniers, "Shortest Path Problems with Resource Constraints", Column Generation, Desaulniers, G.; Desrosiers, J.; Solomon, M.M. (Eds.), Springer, **(2005)**, pp. 33-65.
[13] K. C. Jha, R. K. Ahuja and G. Sahin, "New approaches for solving the block-to-train assignment problem", Networks, vol. 51, no. 1, **(2008)**, pp. 48–62.
[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 Part E: Logistics and Transportation Review, vol. 34, no. 2, **(1998)**, pp. 101–115.
[15] A. M. Newman and C. A. Yano, "Scheduling direct and indirect trains and containers in an intermodal setting", Transportation Science, vol. 34, no. 3, **(2000)**, pp. 256–270.
[16] H. N. Newton, C. Barnhart and P. H. Vance, "Constructing railroad blocking plans to minimize handling costs", Transporation Science, vol. 32, no. 4, **(1998)**, pp. 330-345.
[17] L. K. Nozick and E. K. Morlok, "A model for medium-term operations planning in an intermodal rail-truck service", Transportation Research Part A: Policy and Practice, vol. 31, no. 2, **(1997)**, pp. 91-107.
[18] RAS Problem Solving Competition, Railway Applications Section, Informs (http://www.informs.org/Community/RAS/Problem-Solving-Competition/2011-RAS-Problem-Solving-Competition).
[19] M. Yaghini, A. Foroughi and B. Nadjari, "Solving railroad blocking problem using ant colony optimization algorithm", Applied Mathematical Modelling, vol. 35, no. 12, **(2011)**, pp. 5579–5591.
[20] C. A. Yano and A. M. Newman, "Scheduling trains and containers with due dates and dynamic arrivals", Transportation Science, vol. 35, no. 2, **(2001)**, pp. 181-191.
[21] R. L. Haupt and S. E. Haupt, "Practical Genetic Algorithms", Wiley, New York, **(1998)**.
[22] M. Gondran, M. Minoux and S. Vajda, "Graphs and Algorithms", Wiley, New York, **(1984)**.

Authors

Martim Moniz, is a PhD student in the group of Graphes et Optimisation Mathématique of the Université Libre de Bruxelles. His research regards the application of exact methods to the resolution of a combinatorial optimization problem in the area of Traffic Engineering. Prior to this, Martim Moniz graduated from the Faculty of Sciences of the University of Lisbon, with a Master on Operational Research.

Diogo Silva, is a research assistant at the Faculty of Sciences of the University of Lisbon, currently working on the SAFEPORT project of the Portuguese Navy. His research regards the resolution of a optimization problem involving the positioning and coverage of sensors. Prior to this, Diogo Silva graduated from the Faculty of Sciences of the University of Lisbon, with a Master on Operational Research

João Telhada, is professor at the Faculty of Sciences, University of Lisbon, and full member of the Operations Research Center (CIO). His research has been mainly on reformulation techniques in mixed and integer linear programming, with focus on network design and scheduling problems. João Telhada has a PhD in Operations Research, by the University of Lisbon.

Ref.	Arc	I (k)	I (t)	Description	Cost	BS	WE
A	$(o,0,0,0) \rightarrow (o,k,0,u)$		▨	Block boards a new train, operated by crew k, on its origin node	$\text{TSC} + \text{TTC} \times d(k)$	-	-
		k	▨		$\text{TSC} + \text{TTC} \times d(k) + \text{WEC}$	-	0
B	$(o,0,0,0) \rightarrow (o,k,t,u)$ $t \neq 0$	▨		Block boards a existing train, operated by crew k, on its origin node	-	-	-
		▨	t		WEC	-	t
C	$(d,k,0,u) \rightarrow (d,0,0,0)$		▨	Block leaves a train, operated by crew k, on its destination node	-	-	-
		k	▨		WEC	-	0
D	$(d,k,t,u) \rightarrow (d,0,0,0)$ $t \neq 0$	▨		Block leaves a train, operated by crew k, on its destination node	-	-	-
		▨	t		WEC	-	t
E	$(i,k,t,u) \rightarrow (j,k,t,u)$ $i \neq j$	▨	▨	Block travels from i to j via train t	$\text{CTC}(b)^{(1)} \times d(i,j)$	-	-
F	$(i,k_1,0,u_1) \rightarrow (i,k_2,0,u_2)$ $k_1 \neq k_2$		▨	Block switches from crew k_1 to crew k_2, in a new train (this may lead to the creation of another train, depending on the position of yard i in crews k_1 and k_2)	$\text{TTC} \times d(k_2)$	-	-
		k_1	▨		$\text{TSC} + \text{TTC} \times d(k_2) + \text{WEC} + \text{BSC}(i)$	✓	$0^{(2)}$
		k_2	▨			✓	$0^{(3)}$
		$k_1;k_2$	▨		$\text{TSC} + \text{TTC} \times d(k_2) + 2 \times \text{WEC} + \text{BSC}(i)$	✓	$0^{(2)} + 0^{(3)}$
G	$(i,k_1,t_1,u_1) \rightarrow (i,k_2,t_2,u_2)$	▨		Block switches from train t_1, operated by crew k_1, to train t_2, operated by crew k_2	$\text{BSC}(i)$	✓	-
		▨	t_1		$\text{WEC}^{(4)} + \text{BSC}(i)$	✓	$t_1^{(4)}$
	$t_1 \neq t_2$	▨	t_2			✓	$t_2^{(4)}$
	$t_1, t_2 \neq 0$	▨	$t_1;t_2$		$2 \times \text{WEC}^{(4)} + \text{BSC}(i)$	✓	$t_1^{(4)} + t_2^{(4)}$
H	$(i,k_1,t,u_1) \rightarrow (i,k_2,0,u_2)$ $t \neq 0$ $k_1 \neq k_2$ or $u_1 \neq u_2$			Train t is extended to crew k_2	$\text{TTC} \times d(k_2) + \text{WEC}^{(5)}$	-	$t^{(5)}$
I	$(i,k_1,t,u_1) \rightarrow (i,k_2,0,u_2)$ $t \neq 0$ $k_1 \neq k_2$ or $u_1 \neq u_2$			Block switches from train t, operated by crew k_1, to a new train, operated by crew k_2	$\text{TSC} + \text{TTC} \times d(k_2) + \text{BSC}(i)$	✓	-
J	$(i,k_1,t,u_1) \rightarrow (i,k_2,0,u_2)$		t	Block switches from an existing train t, operated by crew k_1, to a new train, operated by crew k_2	$\text{TSC} + \text{TTC} \times d(k_2) + \text{WEC}^{(4)} + \text{BSC}(i)$	✓	$t_1^{(4)}$
		k_2			$\text{TSC} + \text{TTC} \times d(k_2) + \text{WEC} + \text{BSC}(i)$	✓	0
	$t \neq 0$	k_2	t		$\text{TSC} + \text{TTC} \times d(k_2) + \text{WEC}^{(4)} + \text{WEC} + \text{BSC}(i)$	✓	$t_1^{(4)} + 0$
K	$(i,k_1,0,u_1) \rightarrow (i,k_2,t,u_2)$ $t \neq 0$ $k_1 \neq k_2$ or $u_1 \neq u_2$			Train t_2 is extended backwards to crew k_1	$\text{WEC}^{(5)} - \text{TSC}^{(6)}$	-	$t^{(5)}$
L	$(i,k_1,0,u_1) \rightarrow (i,k_2,t,u_2)$ $t \neq 0$ $k_1 \neq k_2$ or $u_1 \neq u_2$			Block switches from a new train, operated by crew k_1, to a train t, operated by crew k_2	$\text{BSC}(i)$	✓	-
M	$(i,k_1,0,u_1) \rightarrow (i,k_2,t,u_2)$		t	Block switches from a new train, operated by crew k_1, to a train t, operated by crew k_2	$\text{WEC}^{(4)} + \text{BSC}(i)$	✓	$t_1^{(4)}$
		k_2			$\text{WEC} + \text{BSC}(i)$	✓	0
	$t \neq 0$	k_2	t		$\text{WEC}^{(4)} + \text{WEC} + \text{BSC}(i)$	✓	$t_1^{(4)} + 0$

(1) The car travel cost depends on the block.
(2) The work event takes place on the train dropping off the block.
(3) The work event takes place on the train picking up the block.
(4) The work event will only be charged if a work event isn't already taking place, in that yard and in that train.
(5) If a previous block was being picked up or dropped off, in that yard and in that train, a work event will take place.
(6) If train 0 corresponds to a new train, a negative value of TSC must be charged.