RAILWAY TRAFFIC MANAGEMENT
Meet & Pass Problem[*]

Pedro A. Afonso, Carlos F. Bispo
Instituto de Sistemas e Robótica, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
pafonso@sapo.pt, cfb@isr.ist.utl.pt

Keywords: Re-scheduling, Single Track, Railway, Meet and Pass, Train, Conflict, Decision Support System.

Abstract: With the growing interest in the railway sector, mainly because of energetic reasons, there is also a need to increase the efficiency of the railway lines. One way to optimize this sector is to improve quality in the train control process itself. Nowadays, train dispatching is still mostly done by human operators that use elementary tools and thereby solving conflicts sub-optimally. Motivated by these factors, this report presents a model capable of detecting and solving conflicts, for single track railways. More specifically, this model proposes two resolutions methods: a heuristic resolution and a search for the optimal solution. To evaluate the quality of the developed model, several tests were made, obtaining encouraging results. These results showed that it is possible to solve conflicts optimally or near optimally, in a feasible amount of time. This program comes with a graphic interface so that the interaction with the dispatcher can be more user friendly. The major novelties of this work regard the improvement on the conflict detection process, the introduction of capacity conflicts, and the creation of several parameters to adjust the search for the optimal solution.

1 INTRODUCTION

The railway industry plays a vital role in many countries. All railway companies try to achieve more regular and reliable train services, in order to satisfy their customers. One way to optimize these services is to improve quality in the train control process itself. Therefore, railway operators plan train services in detail (i.e. timetables), defining the train order and timing, at junctions and platforms. A robust timetable should be able to deal with minor delays occurring in real-time. However, unexpected events such as technical failures, track incidents, etc., may cause primary delays, which affect the running times, dwelling and departing events. Due to the interaction between trains, these delays may be propagated as secondary delays to other trains, and so, disturbing the entire network. This is why train dispatching is very important. Not only do dispatching orders keep the railway safe from collisions, but also have the objective of minimizing secondary delays throughout the network. As the first objective may be simpler to ensure, the second one is a lot more complex and challenging. Even so, train dispatching is still mostly done by human operators that use elementary Decision Support Systems (DSS). These systems help them to quickly and effectively re-schedule train movements, according to simple dispatching rules. Such rules solve conflicts by means of a simple and local decision criterion, not taking a global look at the system and hence, not searching for an optimal solution. In the last decade however, with the strong competition facing rail carriers, the privatization of many national railroads, and the enormous advances in technology, like computers and telecommunications, many researchers developed new optimization models. Despite the recent advances, providing satisfactory solutions for such a complex problem is still drawing the attention of researchers.

The main goal of this work is to develop, implement, and evaluate a model capable of solving the meet and pass problem and then provide feasible solutions to the human controller. The objectives of this work are: Detect conflicts between trains in a single

[*]This work was supported by the FCT (ISR/IST pluriannual funding) through the PIDDAC Program funds.

railway line; Create a first solution for the conflicts, in a short period of time; Search for the optimal solution; and Propose several near optimal solutions to the dispatcher.

The paper is organized as follows. Section 2 presents a review on the train re-scheduling problem as well as the main technologies and systems already developed to address the problem. Firstly, some basic notions and terminology about this problem are provided. Section 3 introduces the model developed in this paper, by presenting its architecture and a functional description of each of its modules. Then the mathematical formulation of the model is provided along with two formal results that reduce the complexity associated with finding conflicts. Section 4 presents a description of all the modules that constitute the developed application, particularly the algorithms implemented, in order to allow the reader to get an understanding of the entire process. Section 5 presents a summary of the results for the tests carried out to evaluate the proposed solution. Finally, Section 6 is dedicated to the conclusions.

2 BASIC CONCEPTS

In this section we will start by introducing some terminology, after which we review some issues regarding timetables, safety technologies, dispatching rules, typical objective functions, and a short literature review to place our work in context.

2.1 Definition of terms used

Along the paper we will be using some technical terms which will now be presented:

Single railway track - One where traffic in both directions shares the same track.

Track Segment - A track segment is a part of a railway line that is bounded by two distinct end points, in this case the meetpoints.

Block - Section of the railway line delimited by signals.

Siding - A section of track which can be used for the crossing or passing of trains under single track operations. The terms “crossing loop” or “passing loop” are also used in some countries to describe such track sections. A train station on a single line track will usually contain, at least, one siding.

Meetpoint - Location where two trains may cross simultaneously. In this context, meetpoints include not just stations, but also sidings.

Train Conflict - Basically there are two cases: when two trains approach each other on a single line track travelling in opposite directions (Meet); and when a faster train catches a slower train travelling in the same direction (Pass).

Minimum Headway - The minimum time length separating two trains on a single line track. This is usually determined by signals in the case when the trains are travelling in the same direction. When travelling in opposite directions, the minimum headway is determined by the time required for one train to clear the track segment sufficiently before the opposing train can enter it.

2.2 Timetables

Train dispatchers have two main tools to supervise the network, which are the timetable and the train diagram. The timetable is a schedule of trains on a given railway infrastructure. It contains the arrival and departure times of the trains, not only from stations but also from intermediate stations and sidings. The train diagram, or graphical timetable, is a representation of the timetable in a more intuitive way. It is basically a time distance graph where all the routes for traveling trains are represented. The advantage of this diagram is the fact that conflict detection is relatively simply to perform. The horizontal axis represents time of day and the vertical axis the sequence of stations in distance scale. Lines indicate the movement of trains, with the slope indicating direction and speed, horizontal meaning stand-still. Usually, the outbound direction is defined upwards.

2.3 Safety technologies

There are two different systems to ensure the safety of the railway networks: the fixed block technology and the moving block technology. A block section is a track segment between two signals. The signals control the train traffic and impose safe distance headways. There are signals along the lines and also before every station, passing loops, junctions, etc. The most common type of signalling is the three-aspect signalling. A signal aspect may be red, yellow or green. If the subsequent block is occupied by another train, the signal is red. A yellow signal aspect means that the subsequent block section is empty, but the following block is occupied by another train. Finally, a green signal aspect indicates that the next two blocks are empty. Trains have to stop when facing a red signal and wait for that signal to change to green or yellow. When facing a yellow sign the train may proceed its’ course but has to decelerate so that it can stop in case the next signal is red. Each block may host at most one train at a time. As for the mov-

ing block technology, it does not need signals since the exact position and speed of each train is known. Safety is ensured by the regulation of their respective speeds. The safety standards define a maximum speed for each train, depending on the distance from the preceding train, necessary to grant space to stop completely in case of emergency. With this technology, a track segment can host more than one train at a time.

2.4 Dispatching rules

Dispatching rules solve conflicts by means of a local decision criterion. Two of the most common rules are the first-in-first-out (FIFO) rule and the first-out-first-in (FOFI) rule. The FIFO rule solves conflict situations by assigning the block section in discussion to the first train that required it. This means that this rule actually does not require any special dispatching order and lets the traffic proceed with its actual order. This rule is also known as first-come-first-served (FCFS). The FOFI rule assigns the block to the first train that is able to leave it. For this evaluation, the time that each train will take to enter the block and leave it available again has got to be calculated first. The precedence is then given to the train that is able to leave the block first. The FOFI is also referred to as first-leave-first-served (FLFS).

2.5 Objective functions

To evaluate the quality of the obtained solutions, dispatching models need to have an objective function. There are several objective functions each one with its advantages and disadvantages. The choice of what function to use depends on the dispatcher or corporate criteria. The following four optimization criteria, i.e., objective functions have been selected as the most common: Minimization of the total tardiness – $D = \sum_{i=1}^{n} T_i$; Minimization of the total weighted tardiness – $WD = \sum_{i=1}^{n} w_i T_i$; Minimization of the maximum tardiness – $T_{\max} = \max\{T_1, T_2, \ldots, T_n\}$ – this criterion minimizes the maximum delay but many trains may suffer small delays, and is more suited when passenger trains prevail for scheduling; and Minimization of the maximum weighted tardiness – $WT_{\max} = \max\{w_1 T_1, w_1 T_2, \ldots, w_n T_n\}$ –, this function is more suited for mixed traffic situations where the weights w_i are usually the same for trains with the same priority.

2.6 Review

The majority of the models found in the literature follow the structure of a typical Decision Support System, (DSS). These systems have three main functions: predicting train movements, detect possible conflicts, and propose solutions for the conflicts found. Several approaches have been proposed since the early seventies of the last century. However with the advances in technology the most relevant progress has been made in the last two decades. An extensive survey of relevant railway traffic scheduling and rescheduling approaches can be found in (Cordeau et al., 1988).

The work of (Sahin, 1999) uses a heuristic algorithm to reschedule trains. It compares the planned arrival times of trains with the current expected arrival times for conflict detection. Then, it uses a discrete event simulator to evaluate the alternative choices to solve the first conflict found. The system proceeds solving a conflict at a time using this approach. In (Adenso-Díaz et al., 1999) a mixed integer programming model is proposed and a branch-and-bound solution is used. To reduce the size of the search tree, a fixed lenght horizon, in terms of number of services, is assumed.

Another simulation based approach is proposed in (Medanic and Dorfman, 2002), where each train reaching a meetpoint will be compared with nearby trains. If the train can reach the next meetpoint safely, the simulation proceeds. Otherwise, the procedure choses a vicinity train to be stopped and proceeds.

In (D'Ariano et al., 2007a; D'Ariano et al., 2007b) the train rescheduling problem is formulated as a job-shop scheduling problem with blocking and no-wait constraints, and uses the alternative graph of (Mascis et al., 2002; Mascis and Pacciarelli, 2002) as the model structure. The conflicts are solved with dispatching rules.

3 MODEL DESCRIPTION

Our model can be classified as a variable-speed decision support system in real time. It is not supposed to replace the dispatcher but to help him/her in taking the best possible solutions. In other words, this model is a useful tool that allows train dispatchers to foresee the consequences of their decisions and also provides them with other feasible and probably better solutions.

The system architecture is presented in Figure 1. It shows how the DSS will be incorporated in the dispatching process and also illustrates the type of information that will be interchanged with the dispatcher.

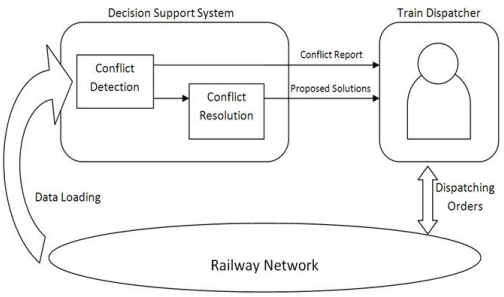

Figure 1: Train dispatching system architecture.

Inside the DSS block are represented its two main functions: the conflict detection and the conflict resolution blocks.

3.1 Problem Definition

Being railway scheduling such a rich and complex problem, it is necessary to define all the model's limitations, assumptions and inputs. This model considers a single railway line that serves trains travelling in both directions. The railway is formed by track segments, which make the connection between all the meetpoints, as shown in Figure 2. In this context, meetpoints include not just stations, but also sidings or any location where two trains may cross simultaneously. So, for this model, trains are only able to meet or pass at meetpoints. As it is represented in Figure 2, train directions will be defined as inbound for trains going from right to left and outbound otherwise. Meetpoints and track segments are numbered in the outbound direction.

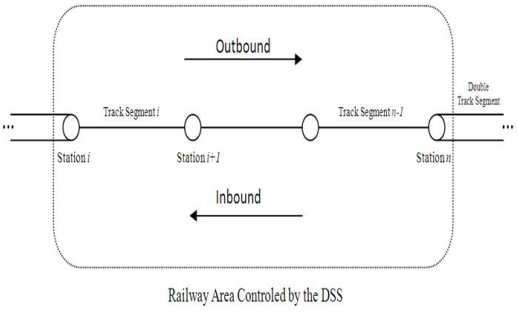

Figure 2: Model Railway Line Topology.

The safety technology considered is the fixed block signalling system. Therefore, trains can follow each other on a track segment with minimum safety headway. Considering this, the model also assumes that trains depart from stations as soon as possible, or in other words, as soon as the following block clears. This policy concerning trains following each other will be very important in the resolution of the pass conflicts that will be explained further ahead in Section 3.4.

Only three different types of trains are considered in the rest of this paper but the model is valid independently of the number of train types. The considered train types are: Fast Passenger Train; Slow Passenger Train; and Freight Train.

This order of presentation is also the usual order of priorities between them. Fast Passenger Trains have priority 1, which is the highest, and Slow Passenger Trains and Freight Trains have priorities 2 and 3, respectively. The first and last meet points, of the considered railway network, do not have to be terminal stations. They can be the interface from single line segments to double track segments or even a denser rail network, as it happens in most cases. Anyway, the safety time intervals between arrivals and departures in these stations will not be checked. In order to make this model more realistic, all meet points have limited capacity. This is a very important aspect that makes a big difference in the quality of the final solution. Very few models take capacity into consideration. For simplicity, the rest of the paper does not explicitly consider acceleration and deceleration time losses. In addition to these model definitions, the following model assumptions are as follows. It is assumed that the location and speed of all the trains in the network is known at all times. Minimum dwell times of the trains at stations and their running times for each track segment are needed in order to verify the constraints described in the following section. Train priorities have to be given for each train. Maximum finite capacities for all meet points are assumed.

In conclusion, a conflict is said to occur when two trains meet or pass at a segment track, when they arrive and depart from stations without the minimum safety intervals, and when a train arrives at a station that is full.

3.2 Mathematical formulation

The meet and pass problem, is essentially an optimization problem, subject to several constraints. Therefore it can be described mathematically. According to the definitions made in the previous section, the set of constraints is now presented in a formal manner. The selected optimization criterion was the minimization of the total weighted tardiness. Below we present the notation being used, after which we detail the mathematical formulation.

i – Train index

u – Meetpoint index
k – Segment index
I_o – Set of outbound trains
I_i – Set of inbound trains
I – Set of trains, $|I| = n$, $I = I_i \cup I_o, I_i \cap I_o = \{\oslash\}$
S^u – Number of trains at station u, $S \subset I$
C^u – Maximum capacity of station u
U – Set of meetpoints, $|U| = m$
K – Set of segments, $|K| = m-1$
r_i^k – Running time for train i at segment k
τ_i^k – Minimum allowed running time for train i at segment k
s_i^u –Dwell time
ω_i^u – Minimum dwell time
d_i^u – Departure time of train i at station u
a_i^u – Arrival time of train i at station u
α_i^m – Scheduled arrival time of train i at terminal station m
s_i^k – Start time of train i at segment k
f_i^k – Finish time of train i at segment k
w_i – Weighted priority of train i
h^k – Minimum headway between arrival and departure times of two consecutive trains at segment k
g_u – Minimum headway between arrival and departure times of two consecutive trains at station u

Objective function:

$$\min Z = \sum_{i=1}^{n} w_i \max\{0, (a_i^m - \alpha_i^m)\} \tag{1}$$

subject to

Free running time constraints:

$$r_i^k \geq \tau_i^k, \forall i \in I, k = 1, 2, \ldots, m-1 \tag{2}$$

Consecutive departure and arrival constraints:

$$f_i^k \geq s_i^k + \tau_i^k, \forall i \in I, k = 1, 2, \ldots, m-1 \tag{3}$$

Minimum dwell time constraints:

$$s_i^u \geq \omega_i^u, \forall i \in I, u = 1, 2, \ldots, m \tag{4}$$

Headway constraints on arrival times at stations:

$$\begin{gathered} a_i^u \geq a_{i'}^u + g_u \oplus a_{i'}^u \geq a_i^u + g_u, \\ \forall i, i' \in I, i \neq i', u \in U \end{gathered} \tag{5}$$

Meet condition:

$$\begin{gathered} d_i^{u+1} \geq a_{i'}^{u+1} + g_u \oplus d_{i'}^u \geq a_i^u + g_u, \\ \forall u \in U, i \in I_i, i' \in I_o \end{gathered} \tag{6}$$

Pass Condition:

$$\begin{gathered} \left(d_i^u \leq d_{i'}^u + h^k \wedge a_i^{u+1} \leq a_{i'}^{u+1} + h^k\right) \\ \oplus \\ \left(d_{i'}^u \leq d_i^u + h^k \wedge a_{i'}^{u+1} \leq a_i^{u+1} + h^k\right), \\ \forall u \in U, \{i, i'\} \in I_o \end{gathered} \tag{7}$$

Meetpoint capacity limits:

$$S^u \leq C^u, \forall u \in U \tag{8}$$

In conclusion, if a schedule does not respect all of the above constraints, it means that a conflict will occur and this schedule is considered unfeasible. However, verifying the meet and pass constraints between all of the trains is unnecessary. In order to explain this statement, it is helpful to rewrite constraints (5) and (6) in relation to the segment instead of the stations.

Meet condition:

$$f_i^k < s_{i'}^k \oplus f_{i'}^k < s_i^k, \forall k \in K, \{i, i'\} \in I \tag{9}$$

Pass Condition:

$$\begin{gathered} \left(s_i^k < s_{i'}^k \wedge f_i^k < f_{i'}^k\right) \oplus \left(s_i^k > s_{i'}^k \wedge f_i^k > f_{i'}^k\right), \\ \forall k \in K, \{i, i'\} \in I \end{gathered} \tag{10}$$

The safety variables g_u and h^k are not considered to simplify the analysis with no loss of generality.

Before the introduction of these main results, it is necessary to make the following assumptions: The trains are considered to be ordered by their entrance times (s_i^k) at a given segment k. This can be represented by the following equation.

$$s_i^k < s_{i+1}^k < s_{i+2}^k \tag{11}$$

Condition (11) must always be verified for the following results to hold. There is also one condition which is always valid since trains cannot move instantaneously, which is stated as follows.

$$s_i^k < f_i^k \tag{12}$$

Theorem 1 If train i does not collide with train $i+1$, and train $i+1$ does not collide with train $i+2$, then train i cannot collide with train $i+2$.

Proof: See Appendix.

Following the reasoning of the proof, one must conclude that the non-conflict condition has the transitivity property. Hence, it can be generalized for all trains in the segment. If there are no conflicts between consecutive train pairs then, one must conclude that there are no conflicts in the segment at all.

Theorem 2 If there is a conflict between train i and train p, with $p \geq i+2$, then there is also a conflict between trains i and $p-1$, or between trains $p-1$ and p.

Proof: See Appendix.

Theorem 2 states that any conflict between two trains, which are apart in terms of their entering order, implies a conflict between two consecutive trains, or a conflict between closer trains.

Corollary 1 In order to conclude about the existence, or non-existence of conflicts, in a given track segment, it is only necessary to check for conflicts between consecutive trains, in terms of their entering order.

Proof: See Appendix.

The importance of these results is the complexity reduction in the conflict detection problem, which was a combinatorial problem and now becomes a linear one. More specifically, for a schedule with n trains, instead of making all the (C_2^n) comparisons, it only needs $(n-1)$ comparisons. Furthermore, this algorithm is used recursively in the search for the optimal solution, which makes the reduction impact even bigger.

3.3 Solving Conflicts

Now that all the constraints for the problem are properly defined, the next step is to explain how to solve the detected conflicts. Conflicts are grouped into four different types, as follows: Meet conflict; Pass conflict; Safety intervals at the station; and Capacity conflict.

For each type of conflict, the range of possible solutions is presented next.

3.3.1 Meet Conflict

This first type of conflict involves two trains and so, there are only two valid solutions to consider. These solutions are either to stop the inbound train or the outbound train. In Figure 3 is an example of a meet conflict (top plot), between trains i and j, followed by its two solutions on the bottom plots.

In Figure 3 we also represent the introduced delay and the safety time interval g_u. The time interval introduced can be g_u or h^k, the chosen interval is the one which has the bigger value. This is also valid for the pass conflict. In meet conflicts, delays are always introduced in the dwell times of trains at the stations but the same does not happen in the type of conflict explained next.

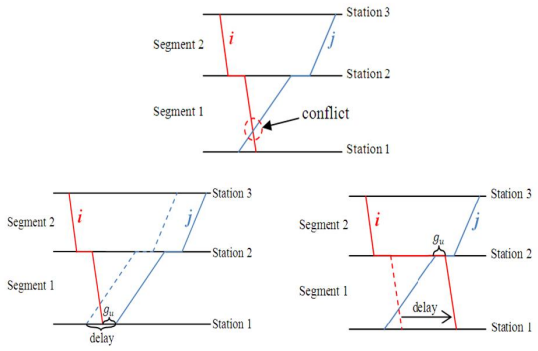

Figure 3: Resolution of a Meet conflict.

3.3.2 Pass conflict

The pass conflict is not as simple as the first one because, in this case, running times also have to be taken into account. As it was referred earlier, it is assumed that trains leave their stations as soon as possible, in order to achieve the minimum possible delay. Therefore, if a faster train succeeds a slower one, its running time will have to be decelerated in order to avoid red signals. The following Figure 4 illustrates the Pass conflict (top) between fast train j and slow train i and their alternative solutions (bottom).

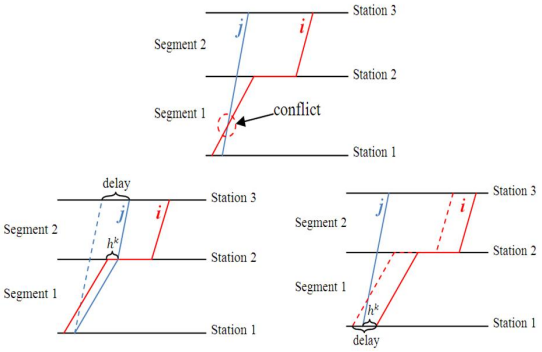

Figure 4: Resolution of a Pass conflict.

In Figure 4 the bottom left plot is a good example of an affected running time. Train j had to be slowed down so as to arrive at station 2 without having to stop in the middle of the segment track, and wait for train i to clear the following block.

3.3.3 Safety intervals at stations

As for the arrivals and departures at stations there are three different situations of conflict: two arrivals, two departures, one arrival and one departure. Because they are very similar situations, only the case with one arrival and one departure will be shown in Figure 5.

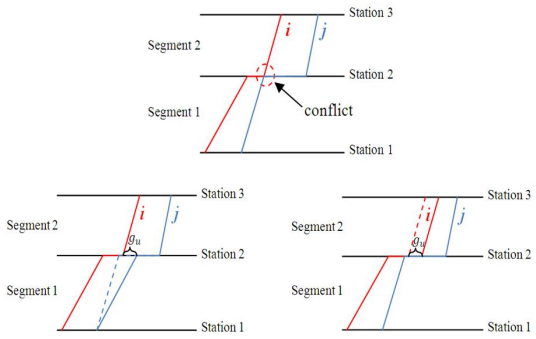

Figure 5: Resolution of a Safety conflict.

3.3.4 Capacity conflict

The Capacity conflict is by far the most complex of them all. The train chosen to wait for the full station to be available again, is not necessarily the last one in the schedule to arrive. In fact, all of the trains at the station, when the conflict is detected, are candidates to be re-scheduled. Hence, one can say that a conflict in a station with capacity N, will have N + 1 possible solutions. The policy in this situation is to hold one of the trains at the previous station until the first train at the full station departs. To help understand this resolution better, the following example in Figure **??** shows a capacity conflict in station 2 with capacity for two trains.

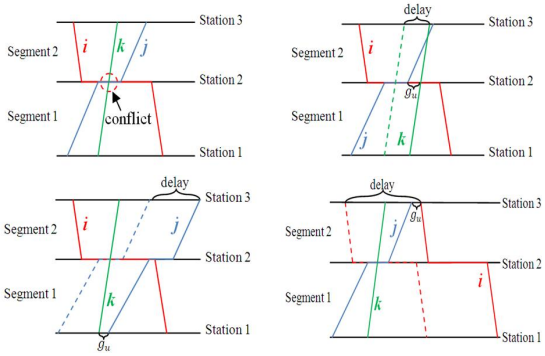

Figure 6: Resolution of a Capacity conflict.

3.4 Towards a feasible solution

Now that the resolutions for each conflict type are defined, it is necessary to select them properly in order to generate the best possible solutions. The objective of this work is to provide the dispatcher with feasible solutions, in a short amount of time, but also try to obtain the optimal solution if possible. Therefore, two approaches were developed in order to satisfy these requirements: the heuristic approach and the search based approach. An explanation of each one of the approaches will be presented next.

3.4.1 Heuristic approach

Taking into account that this program will work in real-time, there is the need to generate solutions as fast as possible to help the dispatcher. Therefore, the objective of this algorithm is to find a feasible solution for the conflict in a short amount of time, and also to reproduce the dispatchers' behaviour in the resolution process. It can be useful to imitate train dispatchers so as to compare the quality of their solutions with the optimal ones. The proposed decision criterion is based mainly on train priorities and in the FOFI rule. For every conflict that involves only two trains, which means all conflict types except for the capacity conflict, the decision criterion works as follows. If one train has higher priority than the other, that's the train that will always go first. If both trains have the same priority, then the FOFI rule is applied. This rule will check which one of the trains is able to solve the conflict in a faster way. In other words, this rule verifies which decision generates de smallest delay on the stopped train.

As for the capacity conflicts, there are $N+1$ trains to consider and the procedure is as follows. Between all the trains involved in the conflict, choose the train with lowest priority in order to be delayed. In case there is more than one train with low priority, choose the last one to arrive at the station.

3.4.2 Search based approach

This approach was developed to provide the dispatchers with optimal or near optimal solutions, better than the ones they usually take. To do so, a search tree is considered. The nodes of the search tree are the conflicts and the branches are the respective solutions. This means that the path from the first node to any leaf node represents a feasible solution of the rescheduling problem. Because the time available is a key element, the search mode adopted was the depth first search (DFS). The DFS allows reaching a first feasible solution without having to search the entire tree as it would happen in a breath first search.

This problem is NP-complete. Therefore, there is a need to adopt some strategies to ensure a solution is produced. The DFS has a branch-and-bound mechanism and each search is limited in computational time and time horizon for the schedule. That is, there is an upper bound on computational time, after which the system is supposed to return the best solution found

so far. To ensure a larger set of options searched, the objective is to produce a conflict free schedule from the present time, t, up to some upper bound in time, say $t+T$. Therefore, the best solution returned is conflict free for T units of time. This enables the system to be re-started to search for more solutions in a computational time of order T.

4 IMPLEMENTATION

In our implementation, there are two distinct parts: the Conflict Detection Loop and the Conflict Resolution part. These parts are indicated below in Figure 7 within the dashed rectangles.

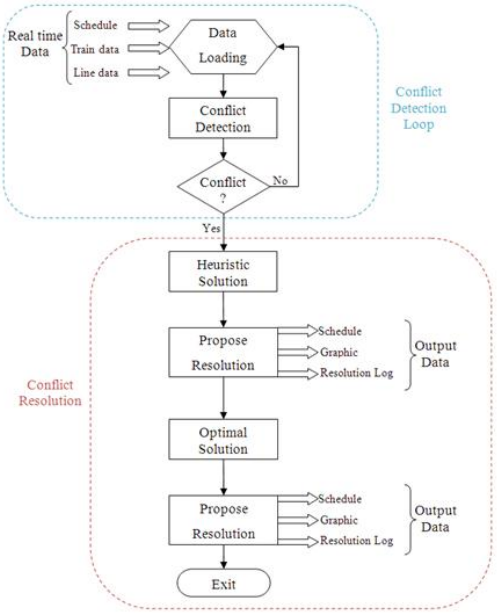

Figure 7: Structure of the software package.

The Conflict Detection Loop is responsible for the surveillance of the railway network. It is also in charge of updating all the necessary data, so that the conflict detector can work with valid information. This loop is supposed to keep running until a conflict is detected. The Conflict Resolution is evidently the part responsible for solving the detected conflicts. This part is divided into two main blocks which are the heuristic solution and the search based solution. These two algorithms that were already introduced in Section 3 will be explained in detail further on. The Conflict Resolution part is also in charge of reporting the achieved solutions to the dispatcher.

4.1 Conflict Detection

The detection of conflicts is executed by the inspection of the network timetable. Two scans at the timetable are performed, the first one tests out for meet and pass conflicts, as for the second scan, it checks for the safety intervals at stations and for capacity conflicts. Here in Table 1 is an example of a timetable with all the arrival and departure times of all the trains from every meetpoint in the railway. The double line between Train 3 and Train 4 is separating the outbound trains (Train 1 to Train 3) from the inbound trains (Train 4 to Train 6). As explained earlier, the meetpoints and segment tracks are numbered in the outbound direction. The times presented in the table are in minutes.

Table 1: Example of a Timetable

	Meetpoint1		Meetpoint2		Meetpoint3	
	ARRIVAL	DEPARTURE	ARRIVAL	DEPARTURE	ARRIVAL	DEPARTURE
Train1	0	10	45	54	62	300
Train2	10	15	30	50	65	73
Train3	5	30	36	62	68	120
Train4	146	149	140	141	104	136
Train5	85	300	72	78	0	66
Train6	100	190	110	160	100	105

Entrance times at Segment2

The first scan sorts the train order for each track segment and then for each pair of trains it verifies the meet and pass constraints. In other words, the algorithm compares the safety constraints between two consecutive trains entering a given track segment.

Considering the example of Table 1, the order of trains entering track segment 2 corresponds to the sort of the highlighted values within the red circles. In this case it would be as indicated in Table 2.

Table 2: Entrance order in track segment #2.

	Track Segment #2	
	Entering times	Order
Train 1	54	2
Train 2	50	1
Train 3	62	3
Train 4	136	6
Train 5	66	4
Train 6	105	5

The scan would begin by checking between Train

2 *vs.* Train 1, then Train 1 *vs.* Train 3 and so on, until the last pair is analized, Train 6 *vs.* Train 4. This type of approach allows saving precious time checking for meet and pass constraints between all the trains in the timetable, and so reducing the complexity of the algorithm. When a conflict is found, the scan does not stop because it might not be the earliest one to occur. It is intended in this program to solve conflicts by the order they occur. Although this may be not very accurate, the time of conflict is assumed as the instant when the first involved train leaves the meetpoint before the conflict. It is logical to think this way considering that this is the deadline to take a dispatching measure. After that instant, the options considered for conflict resolution are not feasible any more.

Table 3: Sorting Meetpoint #1.

	Meetpoint #1	
	Arrival Order	Departure Order
Train 1	1	1
Train 2	3	2
Train 3	2	3
Train 4	6	4
Train 5	4	6
Train 6	5	5

The second scan is very similar to the first one except that in this case, it is not about entering times in segment tracks, but entering and leaving times from meetpoints. The algorithm sorts out the arrival and departure times for one station and compares all the consecutive times checking if the safety time intervals are respected. Simultaneously, a train count for that same station is performed in order to detect capacity conflicts. The blue shaded columns in Tableŕeftimetable will be the example for this scan.

4.2 Heuristic Solution

This algorithm is intended to be fast and effective as explained in Section 3. Therefore, the detected conflict is evaluated concerning only train priorities and the FOFI rule which is a sub-optimal and myopic rule. A flowchart explaining the solution procedure is presented below in Figure 8. If all of the involved trains have the same priority, the program performs the FOFI rule to see which option can cause de smallest delay.

Having decided which train to delay, its schedule is re-arranged accordingly and always respecting future dwelling times. Next, it is necessary to check for more conflicts in the schedule. If another conflict is detected, the process is repeated again from the beginning. This cycle continues indeterminately until the schedule becomes free of conflicts, which means a solution was found.

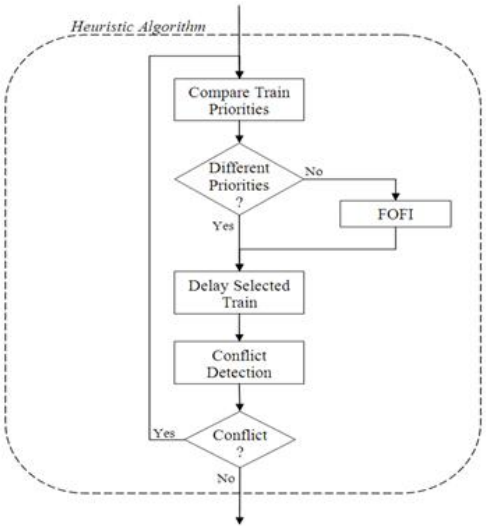

Figure 8: Flowchart of the heuristic algorithm.

4.3 Search based solution

The search algorithm was developed with the objective of reaching an optimal solution. To do so, it should supposedly verify all the possible solutions and then choose the best one among them. Due to the enormous size that the search tree may reach in dense traffic railways, it is unnecessary and also impossible to store the entire tree in the computer memory and to search it in feasible time. The Depth First Search was then chosen because it does not require a lot of memory, and because it provides solutions faster even if they are not optimal. Because time is a key factor in this search, there is the need to adopt some techniques to reduce the size of the search tree. One of them is the introduction of an Upper Bound search limit. The Upper Bound stops the depth search, whenever the considered solution is worse than the best one found until that moment. The quality of each solution is calculated by the cost function presented in Section 3. Another technique used to reduce the search tree is the time horizon. The conflict detector only reports conflicts earlier than the time horizon, ignoring all the following future conflicts. This horizon enables the algorithm to prune a big part of the search tree and consequently saving precious time. Even with these features, there are still schedules that require too much time to be optimally solved. Hence, the dispatcher has another possibility which is the Maximum search time. If this time is exceeded, the algorithm will stop

the search and report the best solution found until that moment. In Figure 9 we present the flowchart of this algorithm.

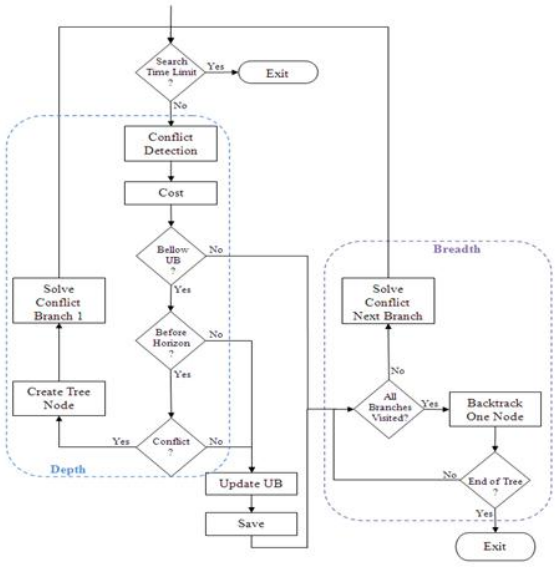

Figure 9: Flowchart of the search algorithm.

5 PERFORMANCE EVALUATION

The development of any application of this kind is not finished without a serious and meaningful evaluation of its performance. Therefore, this chapter aims to present and analyze the results of the tests carried out to check this model's efficiency. More specifically, the main objective of these tests is to understand how the program is influenced by its input parameters. The parameters in study are: Initial schedule; Time horizon; Number of Solutions; Cost function; Maximum search time; and Upper Bound.

Next, each one of these features will then be analyzed individually. The package was implemented in Matlab R2007b on the Windows XP operating system, and all the experiments were conducted on a laptop with an Intel Pentium M 1.20 GHz processor and 1.23 GB of RAM. There are five schedules, each one with different number of trains and meetpoints in order to evaluate the behavior of the algorithms in different complexity environments.

5.1 Initial Schedule

In this first test, it is intended to demonstrate how the increase in the complexity of the input schedule affects the search for a solution. As for the other features, their input was as follows: the time horizon considered for all these experiments was one day (24h); the maximum search time was established at thirty minutes (1800 s); Also the number of requested solutions was one (1); finally, the cost function used was the total weighted tardiness whose weights are the following: Priority 1 = 0.75; Priority 2 = 0.20; and Priority 3 = 0.05.

In Table 4 the simulation results are presented. In the first column is the identification number of the input schedules which are ordered by their complexity. The number of initial conflicts indicates how disturbed the initial schedule is at the beginning of the simulation. In the last two schedules, 4 and 5, the optimal solution is not found before the maximum search time meaning the solutions presented are not optimal but the best ones found in 1800 seconds. The limits columns present values of the number of times that each limit was used to stop the search.

Table 4: Results for different initial schedules (* Optimal solution not found).

Input Schedules	Trains	Meetpoints	Initial Conflicts	CPU Time (s)		Weighted Tardiness		Limits	
				Heuristic Solution	Optimal Solution	Heuristic Solution	Optimal Solution	Time Horizon	Upper Bound
1	6	3	8	2,63	2,58	37,35	23,80	79	429
2	12	6	11	2,52	3,97	101,55	44,30	10	353
3	12	24	22	3,01	944,89	46,54	43,72	62	141694
4	20	24	38	2,73	1800*	52,91	52,14	330	250591
5	40	24	240	6,09	1800*	321,9	382,27	669	205350

Through the analysis of Table 4, the first and most important conclusion is that the more complex and disturbed the schedule is, the longer it takes for the program to find the optimal solution. Quite surprisingly, the algorithm that finds the heuristic solution is barely affected by this complexity increase. Another good result is the quality of the proposed heuristic solution that revealed a near optimal solution in most cases. In Schedule 5, the quality of the heuristic solution could even overcome the thirty minute search of the optimal algorithm. This aspect will be analyzed later on this section. As for Schedule 2, this is a case where the optimal solution, with half the cost of the heuristic solution, is found in just four seconds. This variety from schedule to schedule reinforces the fact that independently of their complexity, schedules may change a lot from one another making greedy algorithms less reliable. As for the limits used in the search for the optimal solution, the upper bound limit revealed more effective than the time horizon limit. Nonetheless, the time horizon was at its maximum value and for shorter time horizons it is

obviously more useful.

5.2 Time Horizon

With this second test, it is expected to show if the variation in the time horizon can accelerate the search for the optimal solution. Once again, it is important to clarify that the indicated solutions are optimal but only within the time horizon limit. They are not the optimal solutions for the whole schedule. For this second set of simulations, the maximum search time, number of solutions and cost function are the same of the previous tests.

Table 5: Variations in the time horizon (* Optimal solution not found).

Input Schedules	Time Horizon (h)	CPU Time (s)		Weighted Tardiness	
		Heuristic Solution	Optimal Solution	Heuristic Solution	Optimal Solution
3	2	2,62	2,44	1,754	1,754
	5	2,81	2,74	9,822	9,822
	10	3,72	27,88	25,30	24,61
	24	3,01	944,89	46,54	43,72
4	2	2,90	2,42	1,762	1,762
	5	2,63	2,73	11,734	11,734
	10	3,09	29,67	29,53	26,72
	24	2,73	1800*	52,91	52,14
5	2	3,17	2,9	0,438	0,438
	5	2,93	1800*	70,42	64,31
	10	3,53	1800*	152,64	186,80
	24	6,09	1800*	321,90	382,27

By the inspection of Table 5, one can conclude that the decrease in the time horizon reduces drastically the search time need to obtain an optimal solution. This result is of course expected since that a short time horizon will ignore the majority of the future conflicts and so reducing the search space. For Schedules 3 and 4, with a time horizon of ten hours, which is more than acceptable, the algorithm is able to solve the conflicts optimally in less than thirty seconds. As for Schedule 5, the densest schedule, the heuristic algorithm completely outperforms the optimal one for long time horizons. Once more, this heuristic values lower than the optimal ones will be explained later in this section.

5.3 Number of Solutions

This test aims to check if the increase in the number of requested solutions has a big influence on the program performance.

This hypothesis comes from the fact that, the higher the number of requested solutions is, the higher the upper bound will be, and thus, less restrictive. Once again the input parameters remain the same except for the time horizon that will be reduced to ten hours. The tests reveal that the number of solutions

Table 6: Variations in the number of solutions.

Input Schedules	Number of Solutions	CPU Time (s) Optimal Solution
3	1	27,88
	5	30,78
	10	32,79
4	1	29,67
	5	33,39
	10	35,16

is not relevant in the behavior of the program. Differences like six seconds are not significant when talking about train dispatching. Besides, there is probably more interest in having a set of feasible solutions available so that the dispatcher can have more choice options.

5.4 Cost Function

The purpose of this test is to show how the heuristic solution may have different quality performances with different cost functions. To do so, four different sets of weights were considered

Table 7: Sets of weights for the weighted tardiness function.

	Priority 1	Priority 2	Priority 3
Set 1	0.7	0.2	0.1
Set 2	0.6	0.3	0.1
Set 3	0.5	0.4	0.1
Set 4	0.5	0.3	0.2

In order to be more realistic, there were two conditions that the above sets had to verify: 1) $\text{Priority}_1 > \text{Priority}_2 > \text{Priority}_3$; 2) $\text{Priority}_1 + \text{Priority}_2 + \text{Priority}_3 = 1$;

The case where all the weights are the same is also considered with the total tardiness cost function. The time horizon is set to 10 hours and the maximum search time was set to 300 seconds. Bellow in Tableẽfresults4 are the results of this experiment. Analyzing the results, it may be concluded, as expected, that the gap between the heuristic solution and the proposed optimal solution increases when the weights are more evenly distributed.

The heuristic algorithms performs as if all the weight is in the train with higher priority and hence these results. It is now up to the dispatcher to analyze the different produced solutions and work with different cost functions in order to decide which set of weights fits better to reality.

Table 8: Variations in the cost function.

Input Schedules	Cost Function	Sets of Weights	Solution Cost	
			Optimal	Heuristic
4	Weighted Tardiness	Set 1	23,97	24,66
		Set 2	29,97	31,27
		Set 3	31,76	37,88
		Set 4	27,23	30,00
	Total tardiness	_	69,47	94,52
5	Weighted Tardiness	Set 1	182,07	169,92
		Set 2	170,94	174,50
		Set 3	158,93	179,14
		Set 4	161,52	209,23
	Total tardiness	_	433,75	771,02

5.5 Maximum Search Time

The interest in studying the maximum search time is mainly to observe how fast does the algorithm converge into an optimal solution. The input parameters are the same as in the first test. To better illustrate how this algorithm behaves in time, the graphic in Figure 10 was made. The solutions quality, or cost, is not presented in its absolute value but instead it is normalized to make their comparison easier. Considering R_{opt} the quality of the optimal solution and R_i the quality of the current solution, then the normalized quality R is defined as being R_i/R_{opt}. In Schedules 4 and 5, the optimal solution considered was the one obtained at 1800 s. In Table 9 are indicated the absolute cost values obtained in this experiment.

Table 9: Variations in the maximum search time (*Optimal Solution found in 944s).

Input Schedules	Maximum Search Time (s)					
	15	30	60	300	600	1800
3	49,73	49,22	48,12	46,29	46,29	43,72*
4	59,79	56,22	55,72	54,12	54,12	52,14
5	386,22	386,12	385,84	385,62	382,27	382,27

As it was concluded in the first test, the more complex the schedule is, the longer it takes for the program to obtain the optimal solution. This conclusion is once more evidenced in this graphic where the lines are logarithmic tendency lines. These tendency lines are slightly overestimated because in the normalization, the best solutions found at 1800 seconds, were considered as being the optimal ones.

5.6 Upper Bound

With this set of tests, it is intended to study the effect of the initial upper bound value, in the performance of the optimal algorithm. By default, the optimal algorithm does not take into consideration the cost value of the heuristic solution. The initial value of the upper bound is set to infinite. This decision was made so that the program could also consider solutions worse than the heuristic solution and let the dispatcher analyze them in the end. The set back of this option is that, by giving a high initial upper bound, the algorithm's performance is affected and so, it might not be as effective as it could. Again, the input parameters are the same as in the first test.

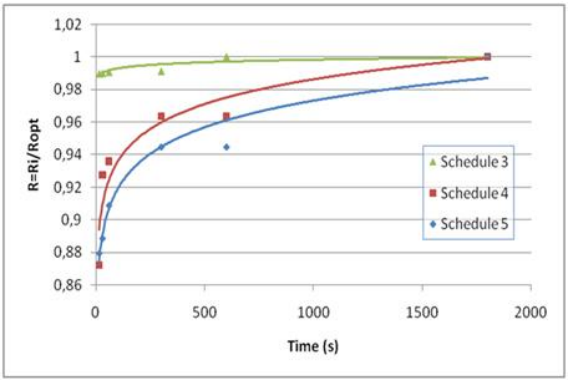

Figure 10: Search algorithm performance.

Table 10: Variations in the value of the initial upper bound.

Input Schedules	Initial Upper Bound	Maximum Search Time (s)					
		15	30	60	300	600	1800
3	Infinite	49,73	49,22	48,12	46,29	46,29	43,72 (944s)
	Heuristic Solution = 46,54	46,54	46,54	46,54	46,54	46,54	43,72 (903s)
4	Infinite	59,79	56,22	55,72	54,12	54,12	52,14
	Heuristic Solution = 52,91	52,91	52,91	52,91	52,91	52,91	51,95

To better illustrate the difference between the two situations, the graphics below were created. Like in the previous section, these values are normalized to the optimal solution or in the case of Schedule 4 to the best solution found in 1800s.

6 CONCLUSIONS

In Section 3, we established a procedure to reduce the complexity of the conflict detection. Also, the capacity conflicts are the most complex to solve, but with the advantage of giving more realism to the model. Finally, two resolution methods for re-scheduling were

proposed. One is the heuristic solution based on a simple dispatch mechanism, which is a greedy algorithm that provides a solution for the problem in a short amount of time by means of a local decision criterion. The second method is a search based mechanism, an algorithm that checks for all the possibilities, by using a search tree and a branch-and-bound technique. In order to make the optimal search less exhaustive and more useful in real-time, the time-horizon and the maximum search time were adopted. With a proper management of these search parameters, the dispatcher may solve a conflict in a short amount of time, pushing the conflicts further on, and gaining precious time for another search, looking for better results. The heuristic algorithm performed fast in all situations, independently of the complexity of the initial schedule. The quality of its solutions was mainly near optimal but, as expected for a greedy algorithm, there were cases were the proposed solution was far from optimal. The search based algorithm depends very much on the complexity of the initial schedule. Nonetheless, the time horizon parameter revealed very effective, with the tradeoff of shortening the validity of the solution. The number of requested solutions has almost no effect on the performance and provides the dispatcher with a set of useful different solutions.

A PROOFS

In both proofs of the theorems, train i is always considered an outbound train. The case when i is inbound, is analogous with the outbound situation, and so it will not be discussed, given the fact that the proof is not conditioned by the direction.

Proof of Theorem 1:

Case 1) $i+1$ is an outbound train. If train i does not collide with train $i+1$, then:

$$f_i^k < f_{i+1}^k \tag{13}$$

If train $i+1$ does not collide with train $i+2$, then there can be two cases:

Case 1.1) $i+2$ is an outbound train. If train $i+1$ does not collide with train $i+2$, then:

$$f_{i+1}^k < f_{i+2}^k \tag{14}$$

And so, from (13) and (14) we get

$$f_i^k < f_{i+2}^k \tag{15}$$

From conditions (11) and (15), it can be concluded that, in this case, train i does not collide with train $i+2$.

Case 1.2) $i+2$ is an inbound train. If train $i+1$ does not collide with train $i+2$, then:

$$f_{i+1}^k < s_{i+2}^k \tag{16}$$

And so, from (13) and (16) we get

$$f_i^k < s_{i+2}^k \tag{17}$$

From conditions (11) and (17), it can be concluded that, in this case, train i does not collide with train $i+2$.

Case 2) $i+1$ is an inbound train. So, if train i does not collide with train $i+1$, then

$$f_i^k < s_{i+1}^k \tag{18}$$

If train $i+1$ does not collide with train $i+2$, then there can be two cases

Case 2.1) $i+2$ is an outbound train. If train $i+1$ does not collide with train $i+2$, then

$$f_{i+1}^k < s_{i+2}^k \tag{19}$$

And so, from conditions (18) and (19) we get

$$f_i^k < s_{i+1}^k < f_{i+1}^k < s_{i+2}^k \Longleftrightarrow f_i^k < s_{i+2}^k \tag{20}$$

With conditions (11) and (20), it can be concluded that, in this case, train i does not collide with train $i+2$.

Case 2.2) $i+2$ is an inbound train. If train $i+1$ does not collide with train $i+2$, then

$$f_{i+1}^k < f_{i+2}^k \tag{21}$$

And so, from equations (18) and (21) we get

$$f_i^k < f_{i+2}^k \tag{22}$$

From conditions (11) and (22), it can be concluded that, in this case, train i does not collide with train $i+2$.

□

Proof of Theorem 2:

Case 1) p is an outbound train. If there is a conflict, then

$$f_i^k > f_p^k \tag{23}$$

Case 1.1) $p-1$ is an outbound train. For train $p-1$ not to collide with train i, it must be the case that

$$f_i^k < f_{p-1}^k \tag{24}$$

For train $p-1$ not to collide with train p, it must be the case that

$$f_{p-1}^k < f_p^k \tag{25}$$

Which means that

$$\begin{cases} f_i^k < f_{p-1}^k < f_p^k \\ f_i^k > f_p^k \end{cases} \Longleftrightarrow \text{Impossible} \tag{26}$$

Thus in this case, train $p-1$ will either collide with train i or train p.

Case 1.2) $p-1$ is an inbound train. For train $p-1$ not to collide with train i, it must be the case that

$$\begin{aligned} s_{p-1}^k > f_i^k \quad &\Longleftrightarrow \quad f_{p-1}^k > s_{p-1}^k > f_i^k \\ &\Longleftrightarrow \quad f_{p-1}^k > f_i^k \end{aligned} \tag{27}$$

For train $p-1$ not to collide with train p, it must be the case that

$$f_{p-1}^k < s_p^k \iff f_{p-1}^k < s_p^k < f_p^k \iff f_{p-1}^k < f_p^k \quad (28)$$

Combining conditions (23), (27) and (28), we get

$$\begin{cases} f_i^k > f_p^k \\ f_{p-1}^k > f_i^k \\ f_{p-1}^k < f_p^k \end{cases} \iff \text{Impossible} \quad (29)$$

Thus in this case, train $p-1$ will either collide with train i or train p.

Case 2) p is an inbound train. If there is a conflict, then

$$f_i^k > s_p^k \quad (30)$$

Case 2.1) $p-1$ is an outbound train. For train $p-1$ not to collide with train i, it must be the case that

$$f_{p-1}^k > f_i^k \quad (31)$$

For train $p-1$ not to collide with train p, the following has to hold

$$f_{p-1}^k < s_p^k \quad (32)$$

These two conditions together with condition (30) yield

$$\begin{cases} f_i^k > s_p^k \\ f_i^k < f_{p-1}^k \\ f_{p-1}^k < s_p^k \end{cases} \iff \text{Impossible} \quad (33)$$

Thus in this case, train $p-1$ will either collide with train i or train p.

Case 2.2) $p-1$ is an inbound train. For train $p-1$ not to collide with train i, it must be the case that

$$s_{p-1}^k > f_i^k \quad (34)$$

This with condition (30) yields

$$s_{p-1}^k > f_i^k > s_p^k \iff s_{p-1}^k > s_p^k \quad (35)$$

This equation is impossible because it violates the initial assumption referred in (10). Hence, in this case, train $p-1$ will either collide with train i or train p.

The result follows.

□

Proof of Corollary 1:

From Theorem 1 we say that if there are no conflicts between consecutive trains on a given track segment, then necessarily there are no conflicts between any of those trains. Therefore this condition assures the non-existence of conflicts. From Theorem 2, we say that any conflict between non-consecutive trains, means that there is also a conflict between consecutive trains. Hence, this conclusion guarantees that all the existing conflicts will be detected.

□

REFERENCES

B. Adenso-Díaz and M. O. González and P. González-Torre, "On-line timetable rescheduling in regional train services," Transp. Res. – Part B, Vol. 33, No. 6, pp. 387-398, 1999.

J. F. Cordeau and P. Toth and D. Vigo, "A survey of optimization models for train routing and scheduling," Transp. Sci., Vol 32, No. 4, pp. 380-404, 1988.

A. D'Ariano and M. Pranzo and I. A. Hansen, "Conflict resolution and train speed coordination for solving real-time timetable perturbations," IEEE Trans. Intelligent Transportation Systems, Vol. 8, No. 2, 2007.

A. D'Ariano and D. Pacciarelli and M. Pranzo, "A branch and count algorithm for scheduling trains in a railway network," European Journal of Operational Research, Vol. 183, pp. 643-657, 2007.

A. Mascis and D. Pacciarelli and M. Pranzo, "Models and algorithms for traffic management of rail networks," RT-DIA-74-2002.

A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," European J. of Operational Research, Vol. 143, pp. 498-517, 2002.

J. Medanic and M. J. Dorfman, "Efficient scheduling of traffic on a railway line," Journal of Optimization Theory and Applications, Vol. 115, No. 3, pp. 587-602, 2002.

I. Sahin, "Railway traffic control and train scheduling baed on inter-rail conflict management," Transp. Res. – Part B, Vol. 33, No. 7, pp. 511-534, 1999.