An Efficient Method to Schedule New Trains on a Heavily Loaded Railway Network

Laura Ingolotti¹, Federico Barber¹, Pilar Tormos²,
Antonio Lova², M.A. Salido³, and M. Abril¹

¹ DSIC, Universidad Politecnica de Valencia, Spain
{`lingolotti`, `fbarber`, `mabril`}@`dsic.upv.es`

² DEIOAC, Universidad Politecnica de Valencia, Spain
{`ptormos`, `alova`}@`eio.upv.es`

³ DCCIA, Universidad de Alicante, Spain
`masalido@dccia.ua.es`

Abstract. With the aim of supporting the process of adapting railway infrastructure to present and future traffic needs, we have developed a method to build train timetables efficiently. In this work, we describe the problem in terms of constraints derived from railway infrastructure, user requirements and traffic constraints, and we propose a method to solve it efficiently. This method carries out the search by assigning values to variables in a given order and verifying the satisfaction of constraints where these are involved. When a constraint is not satisfied, a guided backtracking is done. The technique reduces the search space allowing us to solve real and complex problems efficiently.

1 Introduction

The train scheduling problem is basically an optimization problem which is computationally difficult to solve. Several models and methods have been analyzed to solve it [1], [2]. In our method, we consider a heterogeneous railway network and we focus on adding new trains to a railway network that is already occupied by trains in circulation. The models proposed above are not efficient for the type of problem that we are considering. The majority of the papers published in the area of periodic timetabling in the last decade are based on the Periodic Event Scheduling Problem (PESP) introduced by Serafini and Ukovich [8]. Specifically, an efficient model that uses the PESP and the concept of symmetry is proposed in [6]. However, we cannot use the concept of symmetry because: (i) we allow different types of trains and this does not guarantees the necessary symmetry to be able to use these models; and (ii) the use of the infrastructure may not be symmetrical. There are related works to railway problems such as the following: allocation of n trains in a station minimizing the number of used tracks and allowing the departure of trains in the correct order [4], allocation of new stations through the railway network to increase the number of users [7], etc. There are tools to solve certain kinds of problems such as the Rescheduling tool [3] or the

TUFF scheduler [5]. The Rescheduling tool allows the user to modify a timetable when trains in a section of track cannot run according to the infrastructure, ensuring that scheduling rules are not violated. The TUFF scheduler describes a constraint model and solver for scheduling trains on a network of single tracks used in both directions. However, our problem differs from the type of problems dealt with by the methods mentioned above. In the following section, we explain in more detail the type of problem that we have dealt with.

2 Problem Specification

We propose to add new trains on a heterogeneous heavily loaded railway network, minimizing the traversal time of each new train. The timetables for the new trains are obtained in a search space that is limited by traffic constraints, user requirements, railway infrastructure and network occupation. The problem specification does not require that all considered trains visit the same sequence of locations. There may be many types of trains, which implies different: velocities, safety margins, commercial stops and journeys. Our method takes into account the following scenario to generate the timetables corresponding to the new trains:

1. two sets of ordered locations $L_D = \{l_k, l_{k+1}, \dots, l_{k+m}\}$ and $L_U = \{l_h, l_{h+1}, \dots, l_{h+n}\}$, such that $\{\exists i, j \setminus l_i \in L_D \wedge l_{i+1} \in L_D \wedge l_j \in L_U \wedge l_{j+1} \in L_U \wedge l_i = l_{j+1} \wedge l_{i+1} = l_j\}$. A pair of adjacent locations can be joined by a single or double track section. L_U and L_D correspond to the journey of trains going in up and down directions, respectively.
2. a set of trains for each direction. $T_D = \{t_0, t_2, \dots, t_d\}$ is the set of trains that visit the locations in L_D in the same order given by this sequence and going in the *down direction*. $T_U = \{t_1, t_3, \dots, t_u\}$ is the set of trains that visit the locations in L_U in the same order given by this sequence and going in the *up direction*. The subscript i in the variable t_i indicates the departure order among the new trains going in the same direction.
3. a journey for each set of trains (T_U and T_D) specifies the traversal time for each section of track in L_D and in L_U ($R_{i \rightarrow i+1}$), and the minimum stop time (S_i) for commercial purposes in each l_i .

Considering t_{y,dep,l_x} and $t_{y,arriv,l_x}$ as the departure and arrival times of train t_y from/at location l_x , the problem consists of finding the running map that minimizes the average traversal time, satisfying all the following constraints:

- *Initial Departure Time*. The first train must leave from the first station of its journey within a given time interval ($[min_D, max_D]$ for trains going in the down direction and $[min_U, max_U]$ for trains going in the up direction).

$$min_D \leq t_{0,dep,l_k} \leq max_D \wedge min_U \leq t_{1,dep,l_h} \leq max_U . \quad (1)$$

Fig. 1. Reception and Expedition time constraint

- *Frequency of Departure.* It specifies the period (F_U/F_D) between departures of two consecutive trains in each direction from the same location,

$$\{\forall t_i, l_j \setminus t_i \in \{T_D - \{t_d\}\} \wedge l_j \in \{L - \{l_{k+m}\}\}\}, t_{i+2} \text{dep}_l j = t_i \text{dep}_l j + F_D . \quad (2)$$

$$\{\forall t_i, l_j \setminus t_i \in \{T_U - \{t_u\}\} \wedge l_j \in \{L - \{l_{h+n}\}\}\}, t_{i+2} \text{dep}_l j = t_i \text{dep}_l j + F_U . \quad (3)$$

- *Minimum Stops.* A train must stay in a location l_j at least S_j time units,

$$\{\forall t_i, l_j \setminus t_i \in \{T_D \cup T_U\} \wedge l_j \in \{L_D \cup L_U - \{l_k, l_h, l_{k+m}, l_{h+n}\}\}\},$$

$$t_i \text{dep}_l j \geq t_i \text{arriv}_l j + S_j . \quad (4)$$

- *Exclusiveness.* A single track section must be occupied by only one train at the same time.

$$\{\forall t_j, t_i, l_x, l_y / t_j \in T_D \wedge t_i \in T_U \wedge l_x \in L_D - \{l_{k+m}\} \wedge$$

$$l_y \in L_U - \{l_h\} \wedge l_x = l_{y+1} \wedge l_{x+1} = l_y\},$$

$$t_i \text{dep}_l j \geq t_j \text{arriv}_l j \vee t_j \text{dep}_l j \geq t_i \text{arriv}_l j . \quad (5)$$

- *Reception Time.* At least R_x time units are required at location l_x between the arrival times of two trains going in opposite directions (Figure 1.a).

$$\{\forall t_j, t_i, l_x / t_j \in T_D \wedge t_i \in T_U \wedge l_x \in \{L_D - \{l_k\} \cap L_U - \{l_h\}\}\},$$

$$t_j \text{arriv}_l x \geq t_i \text{arriv}_l x + R_x \vee t_i \text{arriv}_l x \geq t_j \text{arriv}_l x + R_x . \quad (6)$$

- *Expedition Time.* At least E_x time units are required at location l_x between the arrival and departure times of two trains going in opposite directions (Figure 1.b).

$$\{\forall t_j, t_i, l_x / t_j \in T_D \wedge t_i \in T_U \wedge l_x \in \{L_D - \{l_{k+m}\} \cap L_U - \{l_{h+n}\}\}\},$$

$$t_j \text{dep}_l x \geq t_i \text{arriv}_l x + E_x \vee t_i \text{dep}_l x \geq t_j \text{arriv}_l x + E_x . \quad (7)$$

- *Precedence Constraint.* Each train employs a given time interval ($R_{x \rightarrow x+1}$) to traverse each section of track ($l_x \rightarrow l_{x+1}$) in each direction.

$$\{\forall t_i, t_j, l_x, l_y / t_i \in T_D \wedge t_j \in T_U \wedge l_x \in \{L_D - \{l_{k+m}\}\} \wedge l_y \in \{L_U - \{l_{h+n}\}\}\},$$

$$t_i \text{arriv}_l x+1 = t_i \text{dep}_l x + R_{x \rightarrow x+1} \wedge t_j \text{arriv}_l y+1 = t_j \text{dep}_l y + R_{y \rightarrow y+1} . \quad (8)$$

- *Capacity of Each Station.* The number of trains that may stay simultaneously in a station depends on the number of available tracks in it.
- *Closure Times.* Traffic operations and/or passing of trains are not allowed at the closing times of a station.

3 Sequential Algorithm

In this section, we explain the algorithm that is used to solve the specified problem in Section 2. We have named it "*Sequential*" because of the way that it generates the timetable for each new train (Figure 2). For each iteration, the Sequential algorithm constitutes a subset of the whole search space where it searches the values for the problem variables that satisfy all the problem constraints (Section 2). The assignment of valid values to the problem variables generates a timetable (if there is a feasible solution in the subset) for each new train (line 08-14 in Figure 2). The elements of the reduced search space depend on the following values:

1. *Initial departure time* for the first train going in the *down* direction (init_dep_down in Figure 2). A value belonging to the time interval $[min_D, max_D]$ is chosen randomly at each iteration (Constraint 1 in Section 2). This time interval is part of the input parameters I (one of the input parameters of the Sequential Algorithm in Figure 2) given by the final user.
2. *Initial departure time* for the first train going in the *up* direction (init_dep_up in Figure 2). A value belonging to the time interval $[min_U, max_U]$ is chosen randomly at each iteration (Constraint 1 in Section 2). This time interval is part of the input parameters I (one of the input parameters of the Sequential Algorithm in Figure 2) given by the final user.

```

01  procedure Sequential_Algorithm(I,C)
02  begin
03      while (Enough_Time())
04          S=Generate_Set_Ref_Station()
05          ref_st=Get_Ref_Station(S)
06          init_dep_down=Get_Init_Dep_Time(minD,maxD)
07          init_dep_up=Get_Init_Dep_Time(minU,maxU)
08          sched1=Get_Partial_Sched(init_dep_down,lk,ref_st,TD)
09          sched2=Get_Partial_Sched(init_dep_up,lh,ref_st,TU)
10          init_dep_down=t0arrivlref_st+Sref_st
11          init_dep_up=t1arrivlref_st+Sref_st
12          sched3=Get_Partial_Sched(init_dep_down,ref_st,lk+m,TD)
13          sched4=Get_Partial_Sched(init_dep_up,ref_st,lh+n,TU)
14          new_sched=sched1+sched2+sched3+sched4
15          if (Is_Better(new_sched,best_sched))
16              best_sched=new_sched
17      end while
18      Show(best_sched)
19  end

```

Fig. 2. Sequential Algorithm

3. *Reference Station* (ref_st in Figure 2). When the assigned value to a problem variable that represents the departure time of a train violates Constraint 5 defined in Section 2 (to avoid crossings between trains going in opposite directions), the process must decide which of the two trains will have to wait for the section track release. The decision taken by the process will state a priority order between the trains competing for the same resource, the single track section.

When the two trains competing for a single track section are a train in circulation and a new train that should be added to the railway network, the priority order will always be the same. The new train will have to wait until the train in circulation releases the single track section.

When the two trains competing for a single track section are both new trains, the priority order is decided according to the position of the single track section with respect to one station, which we name the *reference station*. The *reference station* divides the journey of each new train into two parts: the first part goes from the initial station of the journey to the *reference station*; the second part goes from the *reference station* to the last station of the journey. Each train will have priority on the single track sections that belong to the first part of its journey. In Figure 3, S_2 is the reference station and it divides the journey of each new train into two parts. For the trains going in the *down* direction (t_0 , t_2 and t_4), the first part of their journey is composed of the track sections (S_0-S_1) and (S_1-S_2); the second part of their journey is composed of (S_2-S_3) and (S_3-S_4). For the trains going in the *up* direction (t_1 , t_3 and t_5), the first part of their journey is composed of (S_4-S_3) and (S_3-S_2); the second part of their journey is composed of (S_2-S_1) and (S_1-S_0). In Figure 3, the possible crossing between the trains t_0 and t_1 (in case that t_0 left from S_2 as soon as possible) is highlighted by a circle. The single track section (S_2-S_3) belongs to the first part of the journey of t_1 , and therefore this train has greater priority than t_0 on this track section. Then, t_0 will have to wait until the single track section (S_2-S_3) is released by the train t_1 . The dotted lines represent the position of the train t_0 if it had left from the station S_2 as soon as possible. The continuous lines represent

Fig. 3. Priority Order between new trains defined by a Reference Station

the real departure time of the train t_0 after the single track section had been released by t_1 . The same reasoning is applied to the other cases in Figure 3 and are designated by circles.

The sequential algorithm iterates obtaining new solutions until the time given by the user has been completely used up or until the user interrupts the execution (line 03 in Figure 2). For each iteration, the sequential algorithm compares the obtained scheduling with the best scheduling obtained to that point. The scheduling that produces the least average traversal time for each new train is the best scheduling. The function *Is_Better*(*new_timetable*, *best_timetable*) returns TRUE if the new scheduling (*new_sched* in Figure 2, obtained in the current iteration) is better than the best scheduling (*best_sched* in Figure 2, obtained until that time). Finally, the sequential algorithm returns the best timetable that has been obtained during the running time.

We have decided not use general solvers such as LINGO or CPLEX because we needed integer binary variables to model given constraints in a mathematical model and the resulting complexity was too high for real world problems. For this reason, we implement a custom solver, which uses the knowledge about this type of problem to solve it in a more efficient way.

3.1 Description of an Iteration of the Sequential Algorithm

At each iteration, the sequential algorithm generates a complete scheduling for each group of trains (T_D and T_U) in two steps. In Figure 2, *sched1* and *sched2* correspond to the scheduling generated for the first part of the journey of each new train going in the *down* and the *up* direction, respectively. In Figure 2, *sched3* and *sched4* correspond to the scheduling generated for the second part of the journey of each new train going in the *down* and the *up* direction, respectively. The complete scheduling is generated in this way in order to establish greater priority for each new train on the first part of its journey. In the case of there being the possibility of a crossing in a track section, the greatest priority will be given to the new train whose timetable had been assigned first on that track section. The first and second part of a journey are established by the chosen reference station at the current iteration. Figure 4 shows how the scheduling for one part of the whole journey is generated. *Verify_Constraints*(*st*, *next_st*, *dep_time*, t_i) verifies that all problem constraints are satisfied by the train t_i in the track section limited by the stations *st* and *next_st*. This function returns the time that must be added to the departure time of t_i in order to satisfy the violated constraint. If the function returns 0, then no constraint has been violated. Consider the set $L' = \{l_x, l_{x+1}, \dots, l_{x+p}\}$ as the ordered set of locations visited by the train t_i , from $l_x = st$ to $l_{x+p} = next_st$. This function assigns values to the variables $ti_dep_l_j$ and $ti_arriv_l_h$ such that $x \leq j < x+p$ and $x < h \leq x+p$ (Figure 5). Figure 5 shows an example of how the constraint that avoids a crossing between trains going in opposite directions is verified. Consider that train t_2 is the train t_i (the train whose timetable is being created), and t' is the train whose timetable has been created first and cannot be modified. The initial value assigned as departure time from $st=l_x$ to t_2 causes a crossing of t' with t_2 (see

```

01  procedure Get_Partial_Sched(init_dep,first_st,last_st,T)
02  begin
03      st=first_st
04      dep_time=init_dep
05      while (st != last_st)
06          next_st=Get_Next_St(st)
07          i=Get_First_Train(T)
08          while (ti!=NULL)
09              error=Verify_Constraints(st,next_st,dep_time,ti)
10              if (error>0)
11                  if (Is_Required_Frequency())
12                      i=Get_First_Train(T)
13                      ti.dep_lst=ti.dep_lst+error
14                  end if
15              else
16                  if (Is_Required_Frequency())
17                      dep_time=ti.dep_lst+FT
18                  else
19                      dep_time=ti+2.arriv_lst+ST
20                  end if
21                  i=i+2
22              end if
23          end while
24          st=next_st
25      end while
26  end

```

Fig. 4. Partial Scheduling

```

01  int function Verify_Constraints(st,next_st,dep_time,ti)
02  begin
03      k=st
04      m=next_st
05      trav_time=Get_Traversal_Time(st,next_st)
06      ti.dep_lk=dep_time
07      ti.arriv_lm=dep_time+trav_time
08      error=Verify_Crossing(ti.dep_lk,ti.arriv_lh)
09      if (error=0)
10          error=Verify_Overtaking(ti.dep_lk,ti.arriv_lh)
11          if (error=0)
12              error=Verify_Availability_Tracks(ti.arriv_lh)
13              if (error=0)
14                  error=Verify_Closure_Time(ti.arriv_lh)
15                  if (error=0)
16                      Set_Timetable(st,next_st)
17                  end if
18              end if
19          end if
20      return error
21  end

```

Fig. 5. Constraints Verification and Timetable Assignment

the picture on the left in Figure 6). The function *Verify_Constraints* computes the time that must be added to *dep_time* in order to avoid this crossing, and

Fig. 7. Average traversal time in function of the running time

quential algorithm takes into account all the problem constraints. It is important to note that LINGO and CPLEX receive the whole search space generated by each problem without any pre-processing to perform some reduction on it. Table 1 has two columns for each solver, the column labeled "Journey Time", and the column labeled "Running Time". The "Journey Time" column shows the objective function value depending on the elapsed time, and the "Running Time" column shows the elapsed time with respect to the start time of each solver. How the objective function value is improved as the elapsed time increases is shown for each solver. The results shown in Table 1 correspond to the following instance: number of new trains for each direction: 10, number of locations that are visited by each train: 42 (Vigo-A Coruna), frequency of departure: 2:00:00, initial departure time interval: [7:00:00,8:30:00].

5 Conclusions

The problem variables are ordered implicitly in the sequential algorithm. The timetable for t_i is assigned before the timetable for t_{i+1} is assigned. The timetable at the location l_x is established before the timetable at l_{x+1} is established. With this ordering, the required backtracking undoes the least number of instantiated variables. In each iteration, the variable values of the problem are computed from an initial departure time ($t_{0dep}l_k = init_dep_down$ and $t_{1dep}l_h = init_dep_up$). When the initial departure time is modified, a different subset of the whole search space is obtained, and therefore a different scheduling can be generated. The reference station establishes the priority order between two trains competing for the same resource, a single track section. When the value for this parameter is modified the priority order between trains is modified as well. Therefore, a

Table 1. Running time to obtain each timetable with different solvers

LINGO		CPLEX		SEQUENTIAL	
running time	journey time	running time	journey time	running time	journey time
120"	2:38:13	15"	2:27:56	<1"	2:28:08
160"	2:26:27	60"	2:23:22	1"	2:25:50
865"	2:21:46	120"	2:21:46	3"	2:21:46

train that in a previous iteration had to wait for the release of a track section, could be the train with greater priority in another iteration. Then, its timetable could be different. Each iteration with a different combination of values for these parameters reduces the search space to a different subset. Each iteration is solved quickly because it only has to explore a reduced part of the whole search space.

Our heuristic consists of obtaining several solutions, where each one is obtained from a different subset of the same search space. This is done instead of searching one single solution in a whole search space, which in real cases may be untractable. The sequential algorithm indicates an order among the problem variables. This order implicitly points out the priority of one train for a resource (platform in a station or single track section). A new order (and a new solution) is produced for each iteration. This is the method that we propose for solving a CSP for this type of problem.

References

1. Bussieck, M.R., Winter, T., and Zimmermann, U.T., 'Discrete optimization in public rail transport', *Math. Programming*, **79(1-3)**, 415–444, (1997).
2. Cordeau, J.F., Toth, P., and Vigo, D., 'A survey of optimization models for train routing and scheduling', *Transportation Science*, **32(4)**, 380–404, (1998).
3. C. K. Chiu, C.M. Chou, J.H.M. Lee, H.F. Leung, and Y.W. Leung, 'A constraint based interactive train rescheduling tool', *Constraints*, **7**, 167–198, (2002).
4. L. Koci and Di Stefano G., 'A graph theoretical approach to shunting problems', *Proceedings of ATMOS 2003 Algorithmic Methods and Models for Optimization of Railways, Electronic Notes in Theoretical Computer Science, Elsevier 92*, **1**, (2003).
5. P. Kreuger, J.O. Carlsson, T. Sjolund, and E. Astrom, 'The tuff train scheduler', *German Puebla, editor, Proceedings of the workshop on Tools and Environments for (Constraints) Logic Programming at the International Logic Programming Symposium ILPS'97*, (1997).
6. C. Liebchen, 'Symmetry for periodic railway timetables', *Proceedings of ATMOS 2003 Algorithmic Methods and Models for Optimization of Railways, Electronic Notes in Theoretical Computer Science 92*, **1**, (2003).
7. M.F. Mammana, S. Mecke, and D. Wagner, 'The station location problem on two intersecting lines', *Proceedings of ATMOS 2003 Algorithmic Methods and Models for Optimization of Railways, Electronic Notes in Theoretical Computer Science, Elsevier 92*, **1**, (2003).
8. Serafini, P., and Ukovich, W., 'A mathematical model for periodic scheduling problems', *SIAM Journal on Discrete Mathematics*, **2(4)**, 550–581, (1989).