A Job-Shop Scheduling Model for the Single-Track Railway Scheduling Problem

Elias Oliveira* and Barbara M Smith
School of Computing
University of Leeds
Leeds
LS2 9JT
UK
{elias,bms}@comp.leeds.ac.uk

Abstract

The Single-Track Railway Scheduling Problem can be modelled as a special case of the Job-Shop Scheduling Problem. This can be achieved by considering the train trips as jobs, which will be scheduled on tracks regarded as resources. A train trip may have many tasks that consist of traversing from one point to another on a track. Each of these distinct points can be a station or a signal placed along the track.

Conflicts may occur when the desired timetable would result in two trains occupying the same section of the track at the same time, and these are resolved by re-timing trains. The objective is to minimize the total delay. Here we consider the case where delays can only be introduced at the start of each trip. We show how the problem can be modelled using constraint programming and successfully solved.

In addition, it discusses a group of practical constraints, incorporated into the software, that arise in real-life problems to which little attention has been paid hitherto. Results of solving 19 real-life problems gathered from literature are also presented.

1 Introduction

The railway services have increasingly become more competitive and so they demand more new features in the planning process. Hence, a tool to help the planner to meet changes in passenger demand is desirable.

This work is concerned with the single-track railway scheduling problem where trains are only allowed to pass one another at stations, sidings and double-track sections. In this paper both stations, sidings and double-track sections will be named *passing points* and, therefore, will be considered as similar entities for the model proposed in Section 3.

Unlike a track segment, a passing point has a specified limit (≥ 1) on the number of trains it can hold at any one time. A conflict can occur when two or

*The author is a PhD student partially funded by CAPES under Grant BEX-1162/96-9
more train services are planned to use the same track segment at the same time. Likewise, at a passing point a conflict occurs when its capacity is exceeded.

In order to resolve a conflict, one of the conflicting tasks must be delayed at that track section or passing point.

The single-line track railway scheduling problem has been tackled by modelling it as an MIP [3, 9, 12]. Considering the NP-completeness of this problem [1], some Meta-heuristic approaches have also been proposed [1, 3, 7, 8].

The aim of previous work was basically to resolve the conflicts between trains arising from a given desirable timetable [10]. However, there are other constraints which train-operating companies would like to see also incorporated into a tool. This would provide them with the possibility of offering better planning services. Therefore, this tool must be capable of both resolving conflicts and satisfying the additional constraints.

The aim of this work is to present a Constraint Programming (CP) approach to this problem and also four practical situations which demand more sophisticated constraints than just the avoiding of conflicts. These sort of constraints have been ignored so far in the literature.

The first of them, which is very useful in practical situations, allows the planner to specify a meeting station for a pair of trains and a minimum dwell time during which they must be together at the station, for changing either crew or goods between these two trains. The second provides the dispatcher with the possibility of specifying that a particular vehicle must be used to form another trip. In this case, the second trip must necessarily be scheduled after the first trip ends, plus the minimum specified time. The third is the blocking of a track segment for a certain period of time for maintenance. Finally, the planner can also specify a time headway limit. This limit specifies a minimum time which trains must stay apart from each other during the trip either for safety or any other operational reasons. These constraints are taken into account in the current implementation.

The numerical results show that the CP approach proposed here is a promising alternative to MIP for scheduling real-world instances of problem.

This paper is organized as follows. In Section 2 the Single-Track Railway Scheduling problem is described and the applicability of the four practical constraints mentioned above are discussed. In Section 3 a model is presented which maps that problem into a special case of Job-Shop Scheduling problem.

An outline of the algorithm used to perform the experiments is given in Section 4 while in Section 5 the results of solving 19 problems gathered from Higgins’ work [3] is presented. The conclusions are given in Section 7.

2 The Single-Track Scheduling Problem

A single-track railway network, as considered in this work, consists of a set of stations, sidings and double-track sections $S$, referred to as passing points; and a set of track segments $R$.

The different types of passing points do, of course, differ in important ways. Whereas stations are places where trains can also stop to be loaded, manoeuvre and change crew, at a siding trains can only stop or slow down in order to let another cross it. However, for the purposes of resolving conflicts between trains,
these passing places will be considered in this paper as a special case of stations, as already mentioned previously.

It will also be considered in this work that a track segment \( r_i \in R \) is necessarily delimited by two signals: one at the beginning and another at the end of the segment, which will control when a train either can or cannot proceed on that segment. This control is to avoid two trains running on the same segment at a time.

A desired timetable is given as data input by the network train-operating companies. For each desired train trip the trip starting and ending points and the departure and arrival times at each station and track segment are specified. In Figure 1 is shown a graphical example of this sort of network in which 11 track segments: \( t_1, t_2, \ldots, t_{10}, t_{11} \); and 5 stations: \( s_1, s_4, s_7, s_8 \) and \( s_f \) are presented.

A train trip is then described by a set of passing points \( S_i = \{ s_i, s_{i+1}, \ldots, s_f \} \subseteq S \) which a train must pass through. In this way a train is performing a task both when it is at a station waiting for either loading or changing crew; and also when it is traversing track segments between two stations. Therefore, a trip can be viewed as a set of tasks to be performed.

The given desired timetable may have conflicts, so that the trains need to be rescheduled in order to eliminate the conflicts. The new timetable must be as close as possible to the first one. In order to eliminate a conflict a new later departure time is chosen for one of the conflicting trains.

This paper describes a program which as well as resolving conflicts between train trips, is capable of dealing with a group of some special constraints such as:

1. \( \text{meet}(A, B, s_k, d) \) - provides that train \( A \) meets train \( B \) at station \( s_k \) and they stay there together for at least \( d \) time units;
2. form \((A, B, d)\) – provides that the same vehicle used for train A will be used to form another trip planned for train B and \(d\) time units is the necessary time to set up the vehicle to perform the second trip;

3. blocking \((t_i, t_{\text{initial}}, t_{\text{final}})\) – provides that track segment \(r_i\) will be unavailable between time \(t_{\text{initial}}\) and \(t_{\text{final}}\).

4. headway \((A, B, d_{AB}, d_{BA})\) – provides that the separation between trains A and B must be at least \(d_{AB}\) time units when A is sequenced first, and \(d_{BA}\) time units otherwise.

It is necessary to stress the difference between the headway defined here and the headway already provided by the model which does not allow two trains on the same track segment at once. The former is to enforce a time distance between two trains based on the type of the train, and the latter is to prevent trains from crashing. For instance, in the first case, a dangerous freight must have a larger time headway if scheduled before a passenger train, but smaller when otherwise.

This combinatorial problem is known to be NP-complete [1] in the literature and some work has been done in order to allow a computer tool to find, at least, a good feasible solution according to a criterion [1, 7, 8]. Other work has been directed towards finding an optimal solution [2, 3, 9, 12]. In this work the objective is also to find the solution with the minimum total delay according to the constraints posed.

3 The Model

The single-track railway can be modelled as a special case of the Job-Shop Scheduling Problem. This can be achieved by considering the train trips as jobs, which will be scheduled on tracks regarded as resources. A train trip may consist of many tasks that require traversing from one point to another on a track. Each of these distinct points can be a station or a signal placed along the track separating track segments. In order to eliminate a conflict a new departure time is chosen for one of the conflicting tasks.

The network considered here is such that only one train can occupy a track segment at a time, whereas more than one train can be at a passing point at a time as long as its capacity limit is observed.

A train trip \(J_i\) is described as \(J_i = \{t_{i1}, t_{i2}, \ldots, t_{ik}\}\), where \(t_{ik}\) is the \(k^{th}\) task of \(J_i\). Each \(t_{ij}\) has its planned departure time \(d(t_{ij})\), as well as its fixed time \(p(t_{ij})\) to complete, and it is such that \(d(t_{i(j+1)}) = d(t_{ij}) + p(t_{ij}), \forall j\); unless otherwise stated.

A delay \(\delta_i = \overline{d}(t_i) - d(t_i)\) is caused when any task of a trip is delayed from its planned departure in order to resolve a conflict on a track segment. Figure 2 shows a typical situation where a conflict is resolved by delaying task \(t_{Ai}\) in order to cause the minimum delay for the whole trip.

In Figure 2, task \(t_{Ai}\) is performed when train A departs from signal \(s_i\) at \(d(t_{Ai})\) (or \(\overline{d}(t_{Ai})\), if it is delayed) and takes \(p(t_{Ai})\) time units to reach the signal \(s_{i+1}\) at the end of the track segment.

To bring all those elements described above into a job-shop scheduling context, consider \(J\) is a set of jobs \(J_1, J_2, \ldots, J_n\) (trips in train context), where
each job $J_i$ has a set of tasks $t_{ij} \in J_i (j = 1, 2, \ldots , |J_i|)$ to be performed. In this paper, every job is considered to have equal priority.

The $\delta(t_{ij})$ variable value is the release time for the task $t_{ij}$, which is a point in time from where a task can start its processing.

Let $\mathcal{R}$ be the set of resources (or machines). In this model $\mathcal{R}$ is assumed to be ordered and a task is to be performed on a specific resource. So if $t_{ij}$ is assigned to $r_k \in \mathcal{R}$, $t_{ij+1}$ is assigned to $r_{k+1}$ when the task belongs to a job corresponding to an outbound train; and to $r_{k-1}$ when it belongs to an inbound train.

Track segments are resources such that $\text{capacity}(r_i) = 1$, whereas passing point has $\text{capacity}(r_i) > 1$. Moreover, the number of conflicts at a time $t$ on a track section is given by $\text{conflicts}(r_i, t)$ and the overall delay is given by the expression:

$$D = \sum_{J_i \in \mathcal{J}} \delta_i$$

(1)

The aim is then to find a feasible departure time for all tasks, so that the conflicts are resolved, the additional constraints are satisfied, and the overall delay, given by the expression in (1), is minimized.

4 The Algorithm

The program developed to solve the problem described above has been implemented in Ilog Scheduler [5, 6], a C++ library that enables the user to represent scheduling constraints in terms of resources and tasks. Constraints can specify the sequencing order of any two arbitrary tasks, conditions on the utilization of a resource, setup time transitions, and so on. Ilog Scheduler is built on top of Ilog Solver, a constraint programming tool, which provides the mechanism of a systematic search that instantiates each variable, either according to the order they are presented or a particular given order. When a variable is assigned a
value, the propagation deletes values from the other variable domains which are inconsistent with the assignment and the set of constraints posted. When either a variable domain becomes empty or no value can be assigned to the variable in order to satisfy all the constraints, a backtrack is needed in order to return to the previous assignment and choose another value for it to continue the search. A problem is infeasible when it is not possible to find values to instantiate all the variables which satisfy the constraints posted.

In order to solve a conflict on a track segment, one of the conflicting tasks needs to be delayed on that track segment. This decision is propagated to the entire trip. According to the model proposed in Section 3, each task must start as soon as its predecessor ends, therefore, the entire trip is also delayed when one of its tasks is delayed.

Given the structure of the problem, we decided to use a chronological strategy to resolve the conflicts. The idea is to resolve conflicts in chronological order, starting from the earliest conflict, on the left of the Time-space Diagram. This is because when resolving a conflict, by delaying one of the conflicting tasks a new conflict may arise. This new conflict will certainly be later than the current conflict being resolved.

The algorithm devised, following this outline, terminates when there is no remaining conflict to resolve. In addition, it works as if pushing the activity which causes a bigger delay to the right side of the diagram. However, each decision to delay a task is made locally, ignoring the consequences it may cause for the rest of the trip or on other trips.

Algorithm 4.1 outlines the strategy used to solve the problems presented in Section 5.

Algorithm 4.1

1:   o-SPT: Schedule all track segments which a conflict is found on it
2:   begin
3:     while \( \exists r_i \mid conflicts(r_i, t) \neq 0, t \in [0, horizon] \) do
5:       pick a \( r_i \) in \( \{ r_i \in R \mid conflicts(r_i, t) \text{ is the earliest} \} \)
6:       begin
8:       - selects a task by applying the Shortest Processing Time
9:         over \( r_i \)'s tasks; and
10:      if the selected task is not possible to be in this order
11:     then backtrack choosing another task
12:    end
14:  od
16: where
18: proc Shortest Processing Time\( (t_A, t_B) \) \( \equiv \)
20: \( t_A, t_B \) are two tasks on a track section
22: if \( d(t_A) + p(t_A) - d(t_B) < d(t_B) + p(t_B) - d(t_A) \)
24: then \( t_A \) is chosen to be scheduled first
26: else \( t_B \) is chosen instead
28: fi.
30: end
This algorithm is such that while there still are conflicts on any section (i.e. resource) (line 3), it chooses the track section which has the earliest conflict to schedule first (line 4).

On a resource, tasks which are not conflicting with any other task are simply ordered according to their earliest possible departure time. For those which conflict, the Shortest Processing Time heuristic [11] (line 8) is used to schedule conflicting tasks and construct a partial solution along with the already unconflicted ones. When a chosen task cannot be scheduled after those which have already been scheduled on the resource, because it would increase the already known total delay (see below), the program backtracks (line 14) in order to choose another task on this resource. When no other task can be placed without increasing the total delay, it backtracks to an upper level, choosing a different resource if necessary.

A trip, as described in Section 2, may be scheduled at any point in time between its specified departure and the horizon schedule limit, which is calculated as \( \text{horizon} = \sum_{j \in J} \sum_{t_{ij} \in J} p(t_{ij}) \).

Initially, each \( \tilde{d}(t_{ij}) \) variable can range from \( d(t_{ij}) \) to \( \text{horizon} \). This huge range of possible values for each variable is because it is not possible to guess what a potential solution configuration is going to look like beforehand, given that the system may have constraints which need to be satisfied altogether.

Once a solution is found, an important value is gathered, which is the total delay \( D \), as defined by the expression in (1), so far obtained. Thereby a new constraint can be included, so that the total delay must be less than the current \( D \). In other words,

\[
\sum_{j \in J} \sum_{t_{ij} \in J_i} \tilde{d}(t_{ij}) - d(t_{ij}) < D; \quad \forall J_i \in J.
\]

This constraint reduces the possible movement in time of a task, and consequently a trip; in other words, it discards branches of the search tree which would lead to worse solutions than what has already been found.

5 Numerical Results

The datasets used to perform the following experiments were provided by Higgins and were used in [3]. However, because the data provided was in some cases incomplete, it was necessary to arbitrate some values of the desirable departure and arrival time for some trains. Although these times may be slightly different from the original, the problems are still the same as far as the number of conflicts to be resolved are concerned.

This section has two purposes. Firstly, to show the capability of dealing with the special constraints presented in Section 2, which have been paid little attention in the literature. Secondly, to present the performance of solving some problems from [3] using constraint programming, following the model presented in Section 3 and the algorithm in Section 4. The objective in any problem presented here is to minimize the overall delay.
5.1 Dealing with some real-world constraints

To exemplify the utilization of the constraints described in Section 2, a small though illustrative example is given. Consider the desired timetable with four trains presented in Figure 4, where trains #11 and #13 are inbound and #14 and #16 are outbound trains.

The single-line track considered (Figure 3) has the following characteristics. It has 8 track sections, 5 stations and a double track section between station $s_2$ and $s_4$, this double track siding is considered in this paper as another station $s_3$. Between $s_1$ and $s_2$ there are 2 track segments, another one just after $s_2$ at the beginning of the double track section, and three others just after it before $s_4$.

Each station is considered, for the sake of this illustration, as having infinite capacity. An exception, of course, is station $s_3$, which has capacity 2: only two trains can use that station at one time.

When the desired group of services was planned to be offered (see Figure 4), some constraints were disregarded. For instance, there are three conflicts within the timetable: one between station $s_2$ and before siding $s_3$, between train #11 and #14 at about time 400. The second one is at the same track segment, between train #13 and #16 at nearly 700 time units. The last one is situated between station $s_5$ and $s_6$ at between 500 and 600 time units, and the trains involved are #14 and #13.

This problem is such that, in addition to looking for a timetable free of conflicts, some other constraints must also be satisfied. For instance, trains #11 and #14 need to meet each other at station $s_3$ in order to have a common dwell for at least 10 time units. Trip #13 must be performed by the vehicle which also performs trip #14. Thereby, trip #13 ought to start only after trip #14 ends, plus the necessary time to prepare the vehicle for the next trip.

Another restriction is in regard to the necessary time headway when a vehicle approaches the stations. Here train #13 needs more time to manoeuvre than train #16. Thus, in case train #16 gets first at the station a time headway of $d_{16,13}$ is set, or $d_{13,16}$ time units otherwise. Finally, blocking a track section for a period of time is analogue of fixing an activity to be performed strictly over that period on that resource. The result of blocking a track segment is then that no other activity can use that resource during that period of time.
Figure 4: A given timetable with 3 conflict points at \( s_2 \sim s_3 \), \( s_5 \sim s_6 \) and \( s_2 \sim s_3 \).

5.2 Solving 19 Higgins’ problems

<table>
<thead>
<tr>
<th>Number</th>
<th>Total</th>
<th>Inbound</th>
<th>( N^2 ) of Passing points</th>
<th>( N^2 ) Conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-29</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>6-11</td>
</tr>
<tr>
<td>40-46</td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>8-14</td>
</tr>
<tr>
<td>50-51</td>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25-35</td>
</tr>
</tbody>
</table>

Table 1: Problems’ classification according to their number of conflicts.

The program described in Section 4 was also used to solve a set of problems from [3], whose characteristics are shown in Table 1. Within these problems there are none of those special constraints discussed in Section 5.1. Therefore, the aim here is both to evaluate the performance of the algorithm used and compare the performance with that obtained by Higgins in [3].

In Table 1 the Number column gives the problem number. Total is the total number of trains to schedule in the problem. Inbound is the number of inbound trains. \( N^2 \) of Passing points states the number of passing points which trains can use to pass each other; and \( N^2 \) of Conflicts is the range of conflicts which the problems of the group have.
Figure 5: The optimal solution resolving the conflicts and satisfying the additional constraints.

The problems are classified according to the number of conflicts they have in their given original timetable. The more conflicts a problem has, the more difficult it is assumed to be [3].

These experiments were performed on a networked Silicon Graphics O2 workstation. The results are presented in Table 2.

The \textit{mDelay} column, in Table 2, is the total of each minimum necessary delay when resolving each conflict individually in a given planned timetable. It is found by considering each conflict in isolation and finding the minimum delay necessary to resolve it, ignoring the effect on other trains. Hence, this value represents in practice a good estimation to the lower bound in the total delay for the problem.

The \textit{First Solution} columns show the time to find the first feasible solution to a problem and its respective delay. Likewise, the \textit{Best Solution} columns show the result to reach the best solution for that particular problem and the overall minimum delay.

In the \textit{CPU} column is shown the total time in seconds to find the best solution and prove its optimality. In the \% column is the time spent on finding the optimal solution as a percentage of the total processing time.

Finally, the \textit{Backtracks} column shows the number of backtracks during the process of finding solutions and proving optimality.
Table 2: Results of solving 19 problems, the minimum possible delay, the time to find the first solution, the time to find the best solution and the comparison between this time and the necessary time to prove a solution is optimal, all in seconds, and the total number of backtracks.

<table>
<thead>
<tr>
<th>Problem</th>
<th>mDelay</th>
<th>First Solution</th>
<th>Best Solution</th>
<th>Backtracks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Delay</td>
<td>Time</td>
<td>Delay</td>
</tr>
<tr>
<td>20</td>
<td>1080</td>
<td>36.79</td>
<td>74.91</td>
<td>88.44</td>
</tr>
<tr>
<td>21</td>
<td>196</td>
<td>14.36</td>
<td>16.52</td>
<td>22.30</td>
</tr>
<tr>
<td>22</td>
<td>204</td>
<td>62.75</td>
<td>62.75</td>
<td>69.56</td>
</tr>
<tr>
<td>23</td>
<td>186</td>
<td>53.76</td>
<td>55.41</td>
<td>62.81</td>
</tr>
<tr>
<td>24</td>
<td>205</td>
<td>12.84</td>
<td>22.66</td>
<td>35.25</td>
</tr>
<tr>
<td>25</td>
<td>271</td>
<td>0.97</td>
<td>6.27</td>
<td>9.66</td>
</tr>
<tr>
<td>26</td>
<td>120</td>
<td>14.42</td>
<td>18.40</td>
<td>22.25</td>
</tr>
<tr>
<td>27</td>
<td>168</td>
<td>0.91</td>
<td>1.49</td>
<td>5.66</td>
</tr>
<tr>
<td>28</td>
<td>150</td>
<td>2.47</td>
<td>4.65</td>
<td>9.35</td>
</tr>
<tr>
<td>29</td>
<td>200</td>
<td>33.19</td>
<td>16.11</td>
<td>21.41</td>
</tr>
<tr>
<td>30</td>
<td>198</td>
<td>20.01</td>
<td>39.25</td>
<td>52.73</td>
</tr>
<tr>
<td>41</td>
<td>245</td>
<td>44.63</td>
<td>80.26</td>
<td>233.42</td>
</tr>
<tr>
<td>42</td>
<td>307</td>
<td>32.62</td>
<td>55.97</td>
<td>78.12</td>
</tr>
<tr>
<td>43</td>
<td>298</td>
<td>251.73</td>
<td>251.73</td>
<td>374.32</td>
</tr>
<tr>
<td>44</td>
<td>312</td>
<td>39.67</td>
<td>110.79</td>
<td>129.68</td>
</tr>
<tr>
<td>45</td>
<td>215</td>
<td>282.73</td>
<td>284.66</td>
<td>301.32</td>
</tr>
<tr>
<td>46</td>
<td>311</td>
<td>74.88</td>
<td>179.78</td>
<td>264.51</td>
</tr>
<tr>
<td>50</td>
<td>625</td>
<td>9663.83</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>51</td>
<td>695</td>
<td>6690.44</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The algorithm proposed in Section 4, along with the value D described in Section 3, which constrains the problem even more, each time a solution is found, is capable of solving to optimality most of the problems in Table 1.

Problems 50 and 51 are exceptions, because it was not possible to reach their optimal solutions within the maximum time of 15 hours stipulated for these experiments. Thereby, in order to find their optimal solutions the program had to be re-run many times with arbitrary decreasing values for D, until reaching their optimal solutions presented in Table 2.

In some cases, such as in problem 22 and 43, the optimal solutions were found at once. In problem 43, backtracking to prove optimality was responsible for about 30% of the total processing time, whereas in the former only 10%. The high number of backtracks in problem 43 shows that once a solution is found, the constraint restricting the total delay is not tight enough to prevent the algorithm from making unfruitful decisions.

The algorithm manages, most of the time, to find a first solution with less than twice the delay of the optimal solution. The exceptions are problems 50 and 51, when the ratios go up to about 2.40 and 3.60, respectively. This suggests that the possibility of wrong decisions in the early stages of the search grows with the size of problem, causing this distance between the first and the best solution.

The factor of about 2 between the first solution found and the optimal solution is still high. The reason for having this distance is because although the
strategy of eliminating delays consists of shifting in time the conflicting task which causes the minimum delay, the process is blind as it does not take into account the global influence of each individual decision. In addition, because of the model proposed for this work, where each task (piece of trip) on a job (a trip) must be performed immediately after its predecessor ends, the entire trip is shifted when a single task is delayed. Hence, it is due to backtracking that different order of trips are tried and, therefore, better solutions are found eventually, in case wrong decisions are made at the outset.

In many cases, after finding the first solution the program found the best solution easily. In problems 25 and 44, however, the first solutions were found in about 15% and 35%, respectively, of the necessary time to find the best solution. Likewise, in problems 50 and 51, where the necessary time to reach the optimal solutions is more than 4 times the time to find the first solutions for these problems. For the other problems in Table 1, the time to find the first solution is more than 50% of the time to find the final solution.

The number of trains and passing points more than doubled for problems 50 and 51 in comparison with the previous problems; so did the number of conflicts to resolve in these two problems. In other words, the size of these problems has greatly increased. The program could not find the optimal solution for these two problems after running over 15 hours.

Because of the difficulty of finding the best solutions for the problems 50 and 51, a binary search technique is used over the interval of possible value for D in order to speed up the search for solutions which approach incrementally the optimal solution.

Initially, an interval I of allowed total delay is set, where \( I = [m\text{Delay}_0, \text{horizon}] \), and the value \( \text{horizon} \) is calculated as described in Section 4. The \( \text{horizon} \) is then large enough to accommodate all trips in sequence. The \( m\text{Delay}_0 \) is a lower bound for the overall delay.

Firstly, an initial solution is found which gives a value to D, the overall delay so far. The interval I is then updated so that \( I = [m\text{Delay}_0, D] \).

Subsequently, the program tries to find a better solution where the new overall delay \( D' < (m\text{Delay}_0 + D) \div 2 \). In case another solution is found, the interval I is once more updated, \( I = [m\text{Delay}_0, D'] \); otherwise, the strategy is changed in order to find solutions sequentially from the last feasible solution found. This is so that time is not wasted searching for solutions in an interval that is unlikely to have a solution.

When using the binary search described above, and letting the program run, for up to 15 hours, it is possible to reach much better solutions than before, within this limit of time. For instance, the program could reach solutions with delays of 1515 and 1285 for problems 50 and 51, respectively.

The binary search did improve the performance on finding better solutions. However, it is still difficult to reach the optimal solution and prove optimality for problems 50 and 51 within a maximum of 15 hours. The optimal solution for problem 50 was proved only after 29 hours using this method.

Higgins [3] reported that problems of the size of the two problems above were solved, using a tailored MIP approach, in about 46 seconds. The result presented here is still far from being comparable with those in [3] in terms of performance. However, we believe there is still room for improvement in the algorithm used here in terms of the data structures used within it. The algorithm proposed has the objective of systematically searching for better solutions and proving
optimality. A faster algorithm can be devised and used specifically to find a good initial feasible solution, and the algorithm presented here can then search for a better solution.

6 Related Work

The model we present in this paper does not allow delaying part of the trip to resolve a conflict, instead the whole trip is delayed. The change to the model, to accept the possibility of delaying part of the trip at a station to resolve conflicts, is planned for future work. It is expected that this additional flexibility will make it easier to find good solutions, but harder to prove optimality.

In this section we discuss two related approaches. These both resolve conflicts by delaying part of the trip at a station. This procedure is analogous to that used by Higgins [3] and Jovanovic [9].

The first work is by Issai in [7, 8]. These papers present a description of a Hybrid Constraint-Based Heuristic and Tabu Search to solve a single-line network scheduling problem which focused basically on finding quick solutions for the problems tackled rather than searching for the optimal one.

Three well known criteria were used to analyze the quality of the schedule during the experiments [11]. The first is the Average of Unit Waiting Time (AUWT). In order to avoid the drawback of having very good schedules for a group of trains while bad schedules to others, another function is also applied, the Maximum Ratio Waiting Time (MRWT). The third cost function applied is the Maximum Ratio of Waiting time to Journey time (MRWJ).

The third cost function takes into account the importance of each train by a weight measure. Regardless of which function is used, it is claimed the system out performed the expert planners by about 5% in the worst problem result and about 27% in the best problem result. In any case this improvement means time-saving, which could allow the operators to introduce more services on the network.

The model used in [7, 8] does not allow two trains running in the same direction to use the same track segment(s) between two consecutive stations. The first train must complete its arrival at the next station before the other train departs from the previous station.

In the model presented in [7] the conflicts are resolved at stations, as in Higgins’ model [4] and in the present work. Issai reports that the largest problem solved is one with 22 trains, 51 stations, 10 double-track segments along the track, and 62 conflicts on single-track sections. If we consider each double-track segment as an abstraction of a station $s_i$ with $capacity(s_i) = 2$, we will notice that in [7, 8] the ratio of conflicts to stations is nearly 1 for the largest problem, and 1.3 conflicts per station on average.

Higgins [4] discusses the effect of reducing the number of stations on the trains’ delays. The result is that the time to resolve a problem increases exponentially as the number of stations decreases, when the number of trains is fixed. The time to solve a problem has similar behaviour when the number of conflicts increases in relation to the number of stations. It is reported in [4] that a problem with 31 trains, 52 conflicts, 12 stations was solved in about 50 seconds, whereas introducing an other 18 trains, giving in total 49 trains, 100 conflicts, and the same number of stations, increases the time to solve the
problem to optimality to 10 minutes. The problems we solve in this paper have a larger ratio of conflicts per station than those in [7]. Here, the ratio is larger than 2 conflicts per station for the most difficult problem and not less than 1 on average; and so the problems we are solving here can be expected to be more difficult than those solved by Issai.

Another related work by Kreuger, et al, is in [10]. This work is based on using Constraint Logic Programming to solve the Single-Track Railway Scheduling problem by mapping it as a Job-Shop Scheduling problem, as in the present paper. However, their work aimed only at finding a good feasible schedule by eliminating the inter-train conflicts, rather than going beyond that to introduce new constraints which might be of interest in a practical context. Moreover, the number of conflicts resolved for the set of trips exclusively on the single-line section of the network is not revealed. Therefore it is difficult to compare their results. Furthermore, to avoid dealing with a great number of tasks to be scheduled, Kreuger [10] adopted a simplification which consists of modelling the problem as having only one track segment between any two stations. However, a constraint was imposed to allow any two trains running in the same direction to use the same track, as long as there was a minimum physical distance between them.

This procedure was also adopted by Higgins [3], but it does not actually consider the fact that, in real life, an unforeseen event can occur to a train, and another one would be running just behind it. Therefore, a more realistic approach might be considering the signals placed along the track as a manner of delimiting each track segment, as is the case in the present work. In this way, a train is alone on a track segment no matter what may happen to the others near by on another track segment.

In [10] the unknown is how much waiting time is to be given for each trip at each station in order to eliminate any possible conflict. The system can be set to constrain the overall delay, the total time for each trip and, also, the makespan. All these can be used as features to prune the search tree.

No attempt was made in [10] towards controlling the maximum number of trains allowed to stay at a station simultaneously. The non-existence of this kind of constraint can result occasionally in an unrealistic schedule if a number of trains are scheduled to a station which cannot hold them all at anyone time.

7 Conclusions

This paper presents a model which maps the Single-Track Railway scheduling problem to a Job-Shop Scheduling problem. Some practical constraints that arise in train operating context are described and some illustrative examples of their use are given. These constraints are, for instance: to able to specify a meeting station for a pair of trains and a minimum dwell time in which they must be together at a station, to specify that a particular vehicle must be used to form another trip, and to specify a time headway distance between two trains.

The model presented in this paper restricts the way conflicts can be resolved. This is because the whole trip is delayed so that the conflict is resolved, whereas the alternative could be as proposed in [3, 9] by delaying only the trip section where the delay occurs. This improvement in the model is under development.

In Section 5, we reported the results of experiments to evaluate the proposed
algorithm on 19 real-life problems used by Higgins [3], although, as we have said, Higgins' model resolved conflicts in a different way to ours. Our results are worse than Higgins' in terms of CPU time. However, the flexibility and expressiveness of CP allows one to describe easily more complex constraints than those explored so far in the literature without needing change in the procedure to solve the problem.

The algorithm needs still some improvements in terms of the data structure it uses. This is because each time the algorithm needs to choose a resource to start scheduling activities (line 4 of the algorithm), it searches for the earliest conflict through a long list of activities, provided by Ilog Scheduler, containing all the tasks in the system. The improvement to be carried out soon is to have, on a resource, the view of only their activities. We believe that this can speed up the process of searching for a solution.

In addition, a tighter value to constrain the delay that each trip can have is needed, because the effect of the constraint in (2), Section 4, comes much later during the searching process.

Moreover, a faster algorithm can be devised and used to find a good and quick initial solution, and the algorithm presented in Section 4 be used basically to search for others and prove optimality.

The results show that CP is promising alternative to MIP for solving real-life instances of problems. In addition, some practical constraints which are hard to specify in MIP, are manageable with CP.

8 Acknowledgements

The authors are very thankful to Andrew Higgins who kindly provided us with the dataset used to carry out the experiments showed here.

The first author would also like to thank CAPES – the Brazilian Government – for the PhD scholarship under Grant N- BEX - 1162/96-9.

References


