The TUFF train scheduler

Two duration trip scheduling on multi directional track networks

Per Kreuger, Mats Carlsson, Jan Olsson,
Thomas Sjöland, Emil Åström
Swedish Institute of Computer Science (SICS)
Box 1263, SE-164 29 Kista, Sweden

Abstract

This work describes a constraint model for scheduling train trips on a network of tracks used in both directions. We argue that a generalization of a job-shop scheduling formulation can drastically decrease the problem size. Preliminary complexity and performance results and a set of examples of schedules generated by our implementation are presented. The system was implemented in the Oz programming system and some experiences and opinions on this choice are noted in the paper.

1 Introduction

This paper describes a scheduling application implemented in the Oz programming language [Sm 95] using DFKI Oz [ST 95] as a development environment. The first version of this application was implemented in about three months using the Oz programming system as a prototyping environment. The system uses the finite domain libraries in Oz and the modelling of the domain was implemented using the object layer of Oz. We extract a constraint model from the object representation, solve it and feed the result to the gui sub system which was implemented with the Oz embedded Tk package.

Oz rich higher order functional syntax provides convenient tools for manipulating the data structures of the constraint model. The tread package and its CCP style implicit synchronization mechanisms can be used to implement constraint propagators (solvers) and very intricate search behaviours, although these aspects are not described in the present state of this work. The system is currently being extended in a multitude of directions, still using Oz as a development environment.

To schedule train trips on a network of multi directed tracks is a problem that resembles but also differs in important ways from other typical scheduling tasks. This paper describes a constraint model and solver for scheduling trains on a network of single tracks used in both directions. This kind of problem can be modelled as a job-shop scheduling (JSS) problem [CP 89, AC 91] considering train trips as jobs to be scheduled on tracks regarded as resources.

Each train trip traversing a track represents a task. Since in general tracks can be used in both directions the JSS formulation requires each trip to demand exclusive use of the track as a resource for the duration of the track traversal. In order to achieve reasonable schedules with such a model it is necessary to keep traversal times and hence individual tracks short. This results in very large scheduling problems (in terms of the number of tasks).

By generalizing this model to take into account two distinct durations for each task we are able to use a much coarser net without losing precision. Since train trips travelling in the same direction no longer require exclusive use of the track resource for the duration of the traversal time the network can be used more efficiently without increasing the granularity of the network representation.

The problem can be concisely stated as:

- Schedule a set of train trips over a fixed network of predetermined paths where trains travel in both directions on single tracks connecting nodes where trains can meet and overtake
- Maintain reasonable bounds on waiting and total times

The rest of the paper is organized as follows: First comes an introductory note illustrating a geometric interpretation of our model. This is followed by a short description of the network representation used and how the problem specifications are built and maintained in the form of a data structure (object representation) we call *plans*. Then we describe the constraint model used and formulate constraints stating soundness criteria of the generated schedules. This section is concluded with a discussion of the enumeration strategy used and some possibilities for improvement in this area.

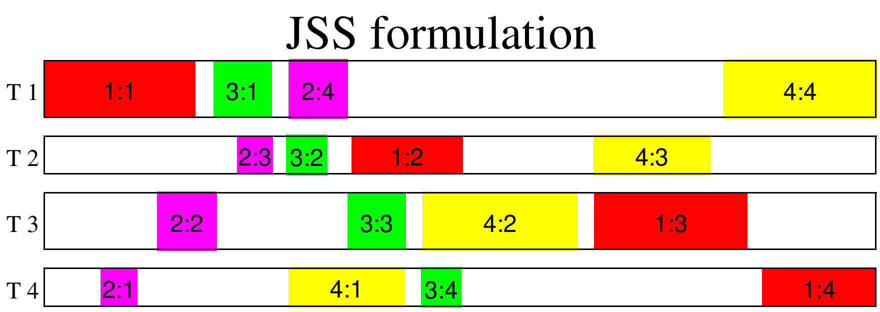
We then briefly consider some complexity and performance issues and conclude the paper with some notes on limitations in the model, some possible lines of future work and with a set of examples.

2 A geometric intuition

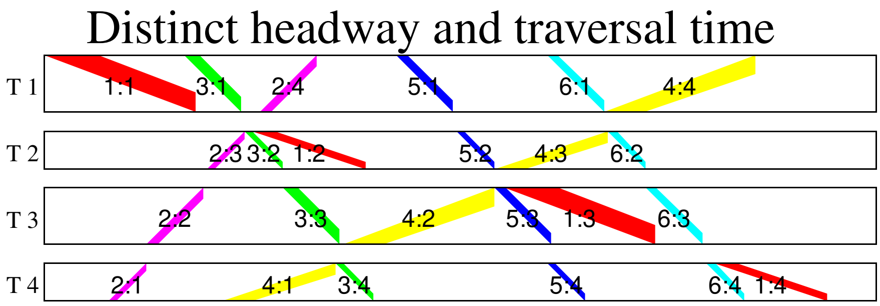
The following two diagrams illustrate the difference between a job-shop scheduling formulation and our model. We use an adaptation of diagrams traditionally used to represent train schedules. Time is on the x axis and the height of each track represents the spatial distance between the end points of each track.

In the JSS formulation (fig. 1) there is only one duration associated with each start time. This duration is illustrated as the width of each task rectangle. Rectangles on a single track must not overlap and track traversals belonging to a single trip follows in sequence.

In the second diagram (fig. 2) the width of each rhombus represents the headway while its extension in the x axis represents the traversal time. The angle of the rhombus represents the speed of each individual traversal. In the example below we have chosen to give slow trains a long headway and quick trains a short headway.



(Fig. 1)



(Fig. 2)

It is easy to see that for comparable total times the number of trains that can be scheduled with JSS on an identical network is significantly lower. It is possible to achieve a schedule similar to the second one using a network with a larger number of (shorter) tracks but then the size of the scheduling problem increases accordingly.
If job-shop scheduling can be seen as the problem of packing the rectangles of the regular Gantt diagram under the precedence constraints imposed by the jobs, the single track train scheduling problem can be seen as the corresponding truncated triangle packing problem.

3 Network representation

A railroad network is represented as three interrelated sets:

- Locations (nodes)
- Tracks (arcs)
- Predetermined paths

Each location is characterized by a unique name, geographical coordinates, a set of tracks adjoined to it and a set of paths passing through it while a track encodes the two locations at its end points, a unique name, a length and a maximum speed at which trains may traverse it. Each path, on the other hand, is determined by a unique name and an ordered sequence of track/direction pairs.

4 Problem representation - plans

We use the notion *plan* to denote a data structure representing a scheduling problem in terms of a given network and a set of required train trips, each with constraints on departure and arrival times at a subset of the locations passed by the trip.

Trips are determined by a unique name, the path the train is required to traverse, a specifications of constraints on arrival and departure times, maximum speeds on involved tracks and average speed for involved vehicles.

The scheduling problem is extracted from the plan as a set of trips each consisting of a set of time slots representing the track traversals of the trip in the required schedule. Furthermore the set of slots associated with each track and path is extracted.

Each slot encodes a unique name, the name of a train trip traversing a track, the name of the traversed track, the direction of the track traversal, headway and traversal times.

The scheduling problem can now be solved by imposing precedence constraints for all departure and arrival times associated with each trip, serializing the slots associated with each track and searching for schedules consistent with these constraints.

In this process we can choose to constrain also the accumulated waiting and total time for each trip and the total and accumulated waiting time required to execute the entire schedule.

There is a trade-off between these two quality measures of the generated schedule such that plans with high demands on (short) waiting times will, in general, have a longer total time and vice versa. See the examples of section 10.1 for an illustration of this phenomenon.

5 Constraint model

Slots represent traversals of tracks by trips at specific time intervals. We use a vector of finite domain variables to represent departure times for such traversals and associate with each slot a variable and *not one but two* durations:

- headway: (time) distance between any two trains departing from some location in the *same* direction on *single* track
- traversal time: time during which no two trains in *opposite* direction may use a *single* track

This model is similar to those used for job-shop scheduling (see e.g. [CP 89, AC 91]) but differs essentially in the use of two durations for each task. The model reduces to JSS if headway is identical to traversal time i.e. if no two trains can ever simultaneously use the same track.

5.1 Constraints

There are two distinct sets of constraints used in the model. The first set is for maintaining the precedence of the track traversals for each single trip. The only uncertainty here is how much slack (waiting time) to allow at each node, for each trip and for the complete schedule.

The second set of constraints is used to enforce a satisfiable serialization of all the slots associated with each track. To summarize:

1. For each trip all track traversals must follow in the sequence defined by its path and not overlap
2. For each track
   - Any two trains travelling in the same direction must respect headway (even if speeds are different)
   - Any two trains in opposite direction must not simultaneously use the same (*single*) track

We will now examine the formulation of these constraints in some detail.

5.1.1 Precedence (trip) constraints

For all slots associated with a trip select uniquely:

- One finite domain variable to represent departure time
- One finite domain variable to represent waiting time (slack) at the location at the end of the traversal

and state:

- One sum constraint per slot to represent precedence between departure and arrival times and upper bounds on waiting times

More formally, assume that we have for each trip unique finite domain variables representing requirements on departure and arrival times, $DepTime_i$ and $ArrTime_i$, a vector $Trip_i$ of slots representing the track traversals of the trip. Assume furthermore that this vector is ordered by the path the trip traverses. Let the slots be represented as records using "." as a field access operation and use the fields $depTime$, $waitTime$, $travTime$ and $headway$ to represent the departure time, the waiting time at the end of the traversal, the traversal time and headway of the slot respectively. The first two contain unique finite domain variables while the second two are fixed durations associated with the slot. The following equality between the departure time of the first slot and that of the whole trip must then hold:

$$Trip_{i1}.depTime = DepTime_i$$

furthermore for each slot $Trip_{ij}$ in the vector but the last the following equality must hold:

$$Trip_{ij}.depTime + Trip_{ij}.travTime + Trip_{ij}.waitTime = Trip_{i(j+1)}.depTime$$

For the last slot $Trip_{in}$ this is replaced by:

$$Trip_{in}.depTime + Trip_{in}.travTime = ArrTime_i$$

The waiting times at each location can furthermore be limited either individually or by sums over the waiting times associated with each and every trip. In addition we may state limits on the latest arrival time allowed in a particular schedule. These two parameters can, as we shall later see, be used to select schedules with different properties.

5.1.2 Resource (track) constraints

Define a disjointness relation as:

$$Disjoint(T_1, D_1, T_2, D_2) \Leftrightarrow (T_1 + D_1 \le T_2) \vee (T_2 + D_2 \le T_1) .$$

Then, for each track traversed by more than one trip:

1. For each pair of trips traversing a single track in same direction let one disjointness constraint ensure that headway is respected at both departure and arrival.
2. For each pair of trains traversing the track in opposite directions let one disjointness constraint ensure that trains will not collide.

This second case corresponds to exclusive use of the track as resource and is equivalent to the situation in job-shop scheduling. The first case however uses essentially the fact that we consider two distinct durations for each slot.

Assume that we have for each track $Track_i$ a set of slots representing trips traversing the track. Then for each pair $(Track_{ik}, Track_{il})$ of slots in the set:

1. If the two trips traverse the track in the same direction the following relation must hold

   $$Disjoint(Track_{ik}.depTime,\ Distance_{ikl},\ Track_{il}.depTime,\ Distance_{ilk})$$

   where

   $$Distance_{ikl} = max(Headway_{ik},\ Headway_{ik} + Track_{ik}.travTime - Track_{il}.travTime)$$

   Note that this formalization in itself does not guarantee that two trains running in the same direction will not run into each other. This is the case only when the headways (for each slot) exceeds the maximum traversal time variance.
2. If the two trips traverse the track in opposite directions instead the following relation must hold

   $$Disjoint(Track_{ik}.depTime,\ Track_{ik}.travTime,\ Track_{il}.depTime,\ Track_{il}.travTime)$$

Note the similarity with the standard job-shop scheduling formulation in the simple case of avoiding collisions (2). The choice between the two cases is completely determined by the relative direction of the traversals and does not in itself introduce indeterminacy into the model.

5.2 The constraint solver

Since the model uses disjunctions in the disjointness constraints for the tracks, to enumerate the departure times generally involves search. We have investigated two distinct methods to do this.

- The most straightforward approach is to enumerate the departure times themselves in some order, rely on propagation to further restrict the domains of remaining departure times and to detect failure in the case of an inconsistent instantiation.
- An alternative approach is to first serialize the slots for each track. The propagation behaviour of the disjunction propagators used then ensures that enumeration of the departure times can be done without search if the minimum or maximum value of each domain is consistently chosen (See e.g. [BP 95]). This possibility and its implications is further discussed in section 8.1.

5.2.1 Enumeration of departure times

The performance figures given in section 7 below was achieved with an approach using the number of constraints associated with each variable as a variable ordering criterion and bisecting the domain of each departure time trying first the lower half of the domain for each variable.

Clearly such a strategy does not lend itself well to optimization. The search tree for this formulation is simply too large. However given reasonable upper bounds on waiting times it is a quick way to find reasonable solutions. Unfortunately search explodes when the bounds place us close to optimal total time which effectively excludes branch and bound search for optimal solutions.

6 Some complexity results

For most realistic problems it is not absolutely clear what is a good problem size measure. The number of trips for a given net is a measure which comes naturally to mind but the number of slots for problems sharing such a size varies irregularly with the particular paths selected for each problem, at least for small problems sizes (number of trips less than 100 on a network of about 45 nodes).

Even if we use the number of slots as a size measure, the complexity varies with the actual selection of paths in the problem. This phenomenon arises because the number of constraints grows quadratically with the number of slots associated with each single track. Some problems distribute their slots evenly over the tracks. These are relatively easy problems. Difficult problems have a few very “tight” tracks with a large number of slots. For such problems the number of constraints can grow very quickly.

The number of variables used to represent the problem are first of all 2 for each slot: One for the departure time and one for the waiting time. From a complexity point of view the second is redundant but in practice it is essential.

There is furthermore one arithmetic equality constraint stated for each slot. So far everything is linear. The real complexity comes from the serialization of the track traversals. We state one disjointness (disjunctive) constraint for each pair of trips traversing it. This operation amounts to finding each subset of size 2 of the slots associated with a track, i.e.

$$\binom{N}{2} = \frac{N(N-1)}{2} = O(N^2)$$

where the measure N is the number of trips traversing a single track.

For most realistic problems average and maximum N will vary greatly from track to track and from plan to plan. Some typical averages (for randomly generated problems) are listed in (Figs. 3 and 4) below.

7 Performance

Performance figures are rough and since the main limitation is memory the problem size we can handle is not really large enough to do a more thorough empirical time complexity analysis.

We currently schedule 200 trips traversing on the average 6 tracks each in a network consisting of 45 tracks where the average number of trips passing a single track is 28 and the number of variables and constraints used are 5541 and 73281 respectively. A first schedule is found in about 74 seconds on a 120 Mhz Pentium PC with 40 MB of RAM memory.

This is searching for a “good” but not necessarily optimal solution (in terms of total time to execute the complete plan) where individual waiting times are bounded by a fixed (5%) fraction of the traversal time. Optimal solutions for smaller examples can be found by (manually) decreasing an upper bound for the total time but for larger examples (> 60 trips) search times tends to increase very fast close to an optimal solution.

On a 248MHz Sun Ultra Sparc using 512 Mb of RAM memory we schedule 1 000 trips in a network of 45 tracks where the average number of trips passing a single track is 138 and the number of variables and constraints used are 9 227 and 586 201 respectively in about 25 minutes. This approaches a realistic size for the highly practical problem of scheduling all major train movements in the Swedish rail network during 24 hours. The real net is of course larger[1].

The corresponding figures for some other sample problems are given in (Figs. 3 and 4) below.

120 MHz Pentium using 16 Mb of memory								
No. of trips	25	50	75	100	125	150	175	200
No. of tracks used	44	45	45	45	45	45	45	45
No. of slots	145	345	450	532	728	901	1 066	1 245
Av. no. of trips/track	3,29	7,67	10,00	11,82	16,18	20,02	23,69	27,67
Av. no. tracks/trip	5,80	6,90	6,00	5,32	5,82	6,01	6,09	6,23
No. of Vars	*222*	*497*	*677*	*834*	*1105*	*1353*	*1593*	*1847*
No. of constraints	472	2 126	3 565	4 870	8 652	12 962	17 814	24 427
Propagator invocations	8 725	73 664	128 403	195 074	491 555	866 931	1 349 553	2 085 021
Time (seconds)	*2,1*	*4,0*	*5,9*	*11,9*	*15,2*	*19,0*	*30,5*	*73,9*

(Fig. 3)

248 MHz Ultra sparc using 512 Mb of memory					
No. of trips	200	400	600	800	1000
No. of tracks used	45	45	45	45	45
No. of slots	1 245	2 490	3 735	4 980	6 225
Av. no. of trips/track	27,67	55,33	83,00	110,67	138,33
Av. no. tracks/trip	6,23	6,23	6,23	6,23	6,23
No. of Vars	*1847*	*3692*	*5537*	*7382*	*9227*
No. of constraints	24 427	95 260	212 500	376 147	586 201
Propagator invocations	2 085 021	15 014 919	48 477 686	115 295 819	223 643 484
Time (seconds)	*9*	*73*	*290*	*716*	*1500*

(Fig. 4)

8 Future work

Since there is currently a lot of time and mainly memory spent in search, search behaviour has to be analysed in more detail. Both domain dependent heuristics and general search procedures such as hill climbing techniques or limited discrepancy search ([HG 95]) should be investigated.

An alternate approach would be to attempt to define global constraints for our model using e.g. edge finding and transitive closure techniques. Some initial experiments in this direction seem to indicate that the required schedules are too sparse (under-constrained) for these techniques to provide significant improvements in propagation behaviour but this possibility should be investigated further. Since memory consumption seems to be the main

1. An early estimate seem to indicate about 250 meeting points and about 300 tracks in the rail network.

limiting factor of how large problems we can attack, global constraints may also be useful to reduce the overall memory consumption.

The model lacks limitation on vehicle and staff resources as well as limits on how many trains that can actually meet in any one location. These are serious limitations in practice and essential for this kind of technique to be useful in solving real life problems. There is as far as we can see nothing in our model that would not extend naturally to this more complex domain but we have not yet seriously addressed these issues.

In practice it will probably also be both possible and necessary to subdivide the complete problem into several smaller and relatively independent sub-problems. For larger realistic problems such as producing schematic schedules several months in advance problem subdivision is most likely the only possible approach. The standard way to do this is to subdivide the problem into that of producing several weekly and daily schedules and to do detailed scheduling for separate geographic regions given non local constraints only on e.g. interregional trains.

8.1 Enumeration of boolean variables representing serialization choices

We have been experimenting with serializing the slots for each track before actually assigning values to the departure times but have so far not been able to reproduce the performance results given with the naive strategy above. It appears that enumerating departure times gives better propagation thus eliminating most of the search while a bad serialization choice early in the enumeration process can potentially generate a huge amount of search much later in the enumeration process.

We believe however that this approach will eventually prove to be superior. First the search tree in this formulation identifies all solutions that do not result in a reordering of the slots within a given track. This means that it will be more realistic to search for optimal or close to optimal solutions, possibly using the naive approach to determine reasonable upper bounds.

Second it is possible with such an approach to formulate more intricate search strategies. It may e.g. be a good idea to first serialize the tracks with the largest number of slots or with the largest number of constraints summed over all its slots. Such tracks would represent the tight resources in the network.

It could also be a good idea to choose first to serialize long tracks or to use the fact that the upper bound on total time is to a large extent dependent on a few relatively long trips, at least for the kind of problem sizes considered so far.

To formulate a good strategy using such ideas is of course not trivial and presently represents future work. It may also turn out that generalizing existing methods to implement global constraints for job-shop scheduling problems e.g. edge finding (see e.g. [BP 95]) and/or transitive closure is in fact feasible. This would represent a complementary approach to refining the present results.

Choosing to represent potential schedules by a boolean vector also allows for more flexible rescheduling and incremental updates of exiting schedules. Investigations of these aspects of constraint programming is currently under way within our group.

9 Conclusions

We have described a novel constraint model for train scheduling and evaluated a corresponding solver on a number of more or less realistically sized problems. With a relatively straightforward solver we have been able to attack problem sizes approaching those appearing in actual practice. It is notable also that even though the simple solver we used in the performance tests does not lend itself well to optimization, the schedules found are in general quite close to manually found optima.

Our modelling approach seem therefore to represent a major improvement of more traditional alternatives.

The choice to use Oz as developmentmade it possible for us to quickly implement a first prototype of the scheduler. In the currently ongoing work to extend the system to handle resource limitations of various kinds we have chosen to continue to use Oz.

The main features used include:

- The object system was used to model the different aspects of the problem domain.
- The higher order functional syntax and concurrent guarded clause conditionals of the language gave us high level tools to manipulate the models, and customize the built in constraint solving libraries to suit our domain.
- The Tk interface provides an adequate gui development environment.

We used the finite domain library extensively and found it to very efficient. It does not yet include some of more sophisticated global constraints found in e.g. the CHIP system but it is possible to extend the constraint packages both on level of Oz it self and in C using Oz's constraint propagator interface.

The Ilog system provide a richer gui development environment but the advantage of the high level constructs present in Oz was clearly an advantage, at least in the early development face. We are currently collecting experience with using Oz in a more long term and ambitious project.

The Oz system allowed us to implement a first prototype very quickly and is well suited to the explorative development work that we pursue in our current projects.

10 Sample schedules

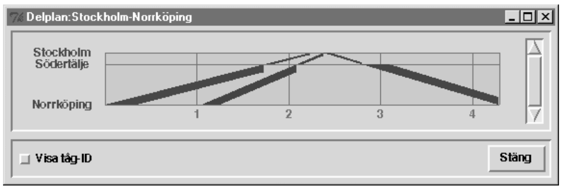
The following diagrams illustrate some sample schedules generated by our implementation. The implementation was done completely in the Oz programming language [Sm 95, ST 95] using the embedded Tcl/Tk interface for the GUI parts. The diagrams encode time on the x axis and spatial distance between locations on a given path on the y axis. The width of the rhombi representing the trip's track traversals encode the headway for each individual trip on the given track. The angle of the rhombi sides encodes the speed of the train during the traversal with angles larger than $90°$ encoding trips moving in opposite direction.

10.1 Trade-off between total time and waiting time

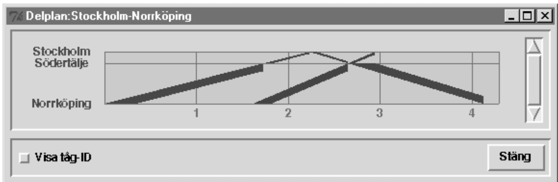
The trade-off between total time and waiting time is nicely illustrated by the following three schedules.

In the first (fig. 5) we have set a very tight limit on the waiting time for all trains at the intermediate location (Södertälje). So tight, in fact, as to exclude any meetings at this location. In the second schedule (fig. 6) we loosen this bound slightly to allow a maximum of two trains to meet at this location. In the last (fig. 7), finally, we allow an arbitrarily long waiting time thus allowing all three trains to meet at this point. Note how the total time decreases throughout this process.

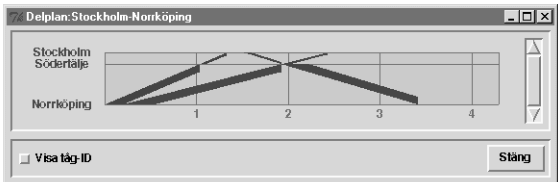
Note also that headway is respected even where trains travel at different speeds. The steeper the angle the quicker the train.



(Fig. 5)



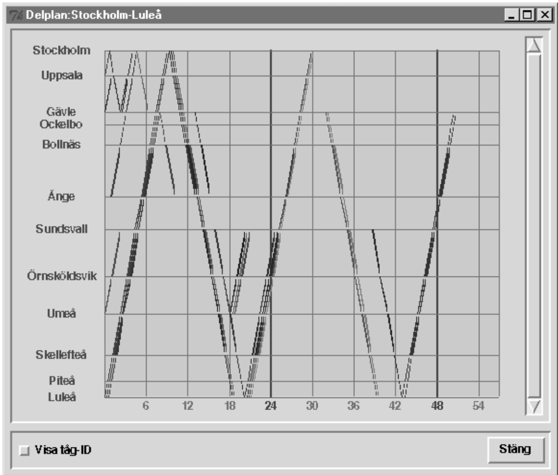
(Fig. 6)



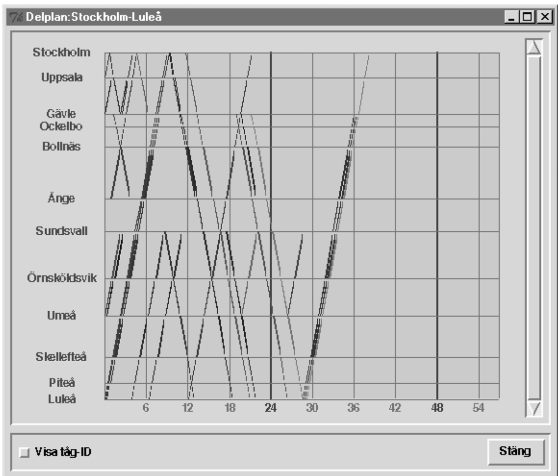
(Fig. 7)

10.2 Some larger schedules

The following examples illustrate some larger schedules and how much bounds on the waiting times influences the sparseness of the schedules. The following diagrams represent identical fragments (all traversals of the tracks of one particular path) of 2 different 60 trip schedules. The first (fig. 8) with low tolerance on waiting times, the second (fig. 9) with high tolerance on waiting times. Note also how the total time is decreased with about 25% by bounding waiting times to allow approximately three trips to meet at each location.



(Fig. 8)



(Fig. 9)

11 Acknowledgements

The work reported in this paper is one of the results of a pilot study done in cooperation with the Swedish State Railway (SJ). The motivation behind this cooperation has been to evaluate the feasibility of using modern constraint techniques for realistically sized problems in their domain.

We would like to thank Jolanta Drott and her group at the Swedish State Railway (SJ) for presenting us with the problem and for good cooperation in the TUFF project, on which the present results are built.

12 References

AC 91

Applegate, D.; Cook, W. “*A computational Study of the Job-Shop Scheduling Problem*”. In *ORSA Journal of computing*, 3(2):149-156, 1991.

BP 95

Baptiste, P.; Pape, C. L. “*A theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling*”. In the “*Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence Montreal, Quebec*” pp. 400-606, 1995.

CL 94

Caseau, Y.; Laburthe, F. “*Improved CLP Scheduling with Task Intervals*”. In *Proceedings of the Eleventh international Conference on Logic Programming*, Santa Margherita Ligure, Italy, 1994.

CP 89

Carlier, J.; Pinson, E. “*An Algorithm for solving the Job-Shop Scheduling Problem*”. In the *Management Science 35(2) 164-176, 1989.*

CP 94

Carlier, J.; Pinson, E. “*Adjustments of Heads and Tails for the Job-Shop Scheduling Problem*”. In the *European Journal of Operational Research*, 78:146-161, 1994.

HG 95

Harvey, W. D.; Ginsberg, M. L. “*Limited Discrepancy Search*”. In the *Proceedings of the International Joint Conference on artificial Intelligence*, pp 607-613,1995.

Sm 95

Smolka, G. “*The Oz programming model*” In van Leeuwen, J, ed. “*Computer Science Today*”. In *Lecture Notes in Computer Science* volume 1000: 324-343. Springer Verlag, Berlin, 1995.

ST 95

Smolka, G.; Treinen, R. Eds. “*DFKI Oz Documentation Series*”. Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany 1995.

SC 93

Smith, S. F.; Cheng, C. C. “*Slack-Based Heuristics for Constraint Satisfaction Scheduling*”. In proceedings of the *Eleventh National Conference on Artificial Intelligence*, pp. 139-144, American Association for Artificial Intelligence, Washington, 1993.

VH 89

Van Hentenryck, P. “*Constraint Satisfaction in Logic Programming*”. Programming Logic Series. The MIT Press, Cambridge, MA, 1989.

XSS 92

Xiong, Y.; Sadeh, N; Sycara, K. “*Intelligent Backtracking Techniques for Job-Shop Scheduling*” In the Proceedings of the *Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992.