The TUFF Train Schec

Two duration trip scheduling c directional track network

by Per Kreuger et al.

piak@sics.se, http://www.sics.se/~piak
Swedish Institute of Computer Science (SI
Box 1263, S-164 29 Kista, Sweden
September 1997

Introduction & Background

This talk describes a scheduling application implemented in the Oz programming language using DFKI Oz as a development environment.

- A first version of the 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 subsystem which was implemented with Oz's 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 search behaviours

The Problem

Schedule a set train trips over a fixed network of predetermined paths where:

- Trips may traverse tracks in both directions
- Paths consists of tracks connecting nodes where trains can meet and overtake.
- Schedules should respect reasonable bounds on time required to execute plan and on waiting times

Two Duration Trip Scheduling

Traditionally train scheduling problems have been modelled using a job shop scheduling (JSS) approach i.e:

- Tracks are regarded as resources used by individual train trips. C.f. “jobs”.
- Each traversal of a track by some train (a slot) is regarded as a task to be scheduled on that track.

- In this model each trip requires exclusive use of the track resource for the duration of the track traversal.
- If we use a coarse network representation (to reduce the number of resources) traversal times tend to be long.
- In this case exclusive use of individual tracks is wasteful of network bandwidth when trips run in the same direction

An alternative model

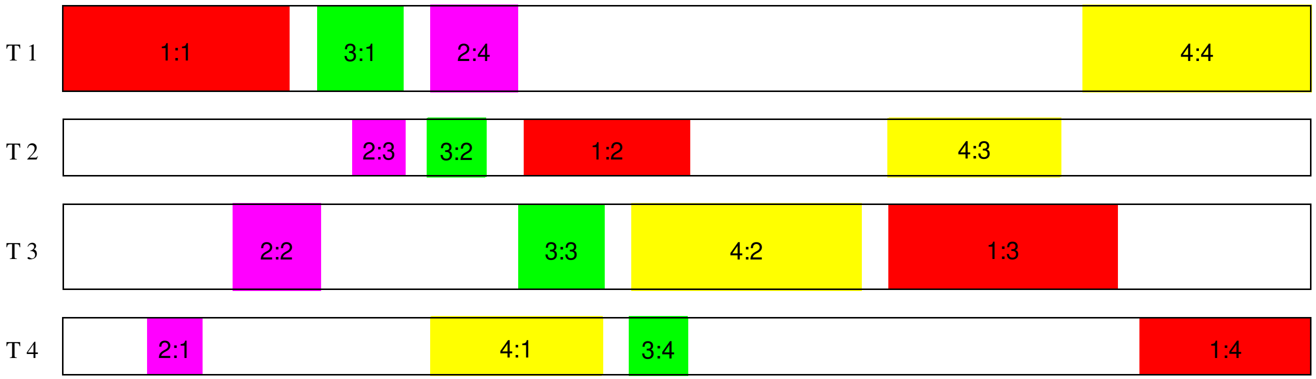
We propose a model where in addition to the traversal time we consider also a second duration associated with each task: the *headway*

- The traversal time determines the minimal time distance between two trips running in opposite direction on a track

- The headway determines the minimal (time) distance between two trips traversing a track in the same direction

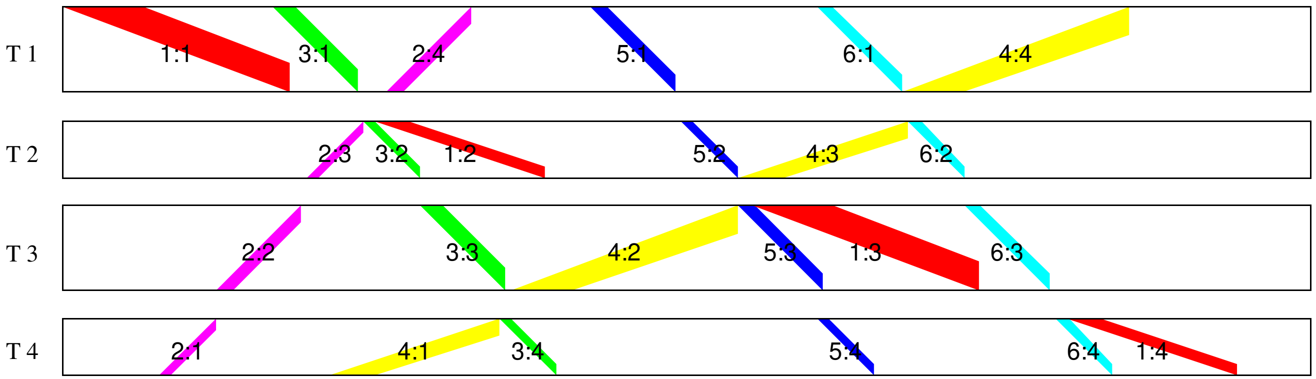
To see why this makes a significant difference consider the following two graphs:

JSS formulation



(Fig. 1)

Distinct headway and traversal time



(Fig. 2)

- Time is on the x axis and spatial distance on the y axis.

Constraint model

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, i.e:
  - For each trip all track traversals must follow in the sequence defined by its path and not overlap
  - The waiting times at each location is constrained individually, for each train and for the whole schedule

- The second set is used to enforce a serialization of all the slots associated with each track so that 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 track

The first set is straight forward to formulate. The second is somewhat more involved.

Define

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

and 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:

- 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

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

- If the two trips traverse the track in opposite directions the following relation must hold

$$\begin{aligned} &Disjoint(Track_{ik}.depTime, Track_{ik}.travTime, \\ &\qquad Track_{il}.depTime, Track_{il}.travTime) \end{aligned}$$

Notes 1

- “If the two trips traverse the track in the same direction the following relation must hold” on page 12
  Note that this formalization (first bullet) 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.
- “If the two trips traverse the track in opposite directions the following relation must hold” on page 13
  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.

Implementation

We implemented a scheduler in Oz using the techniques described above.

- We did not use the state-of-the-art scheduling primitives (edge finding etc.) included in Oz since our model uses the two duration approach
- It may be possible to generalize the edge finding techniques to the two duration case but it is not clear that it will benefit the application

Search

We use a very simple enumeration mechanism where the most constrained departure time is chosen and tentatively constrained to be smaller than the mid value of its domain

- Using such a simple search mechanism we were able to solve quite large problems
- As a comparison our industrial partner claims that the run about 1500-2000 trips/day

Performance

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*

I.e. 1000 trains in about 25 minutes.

- Memory is more limiting than time

Limitations

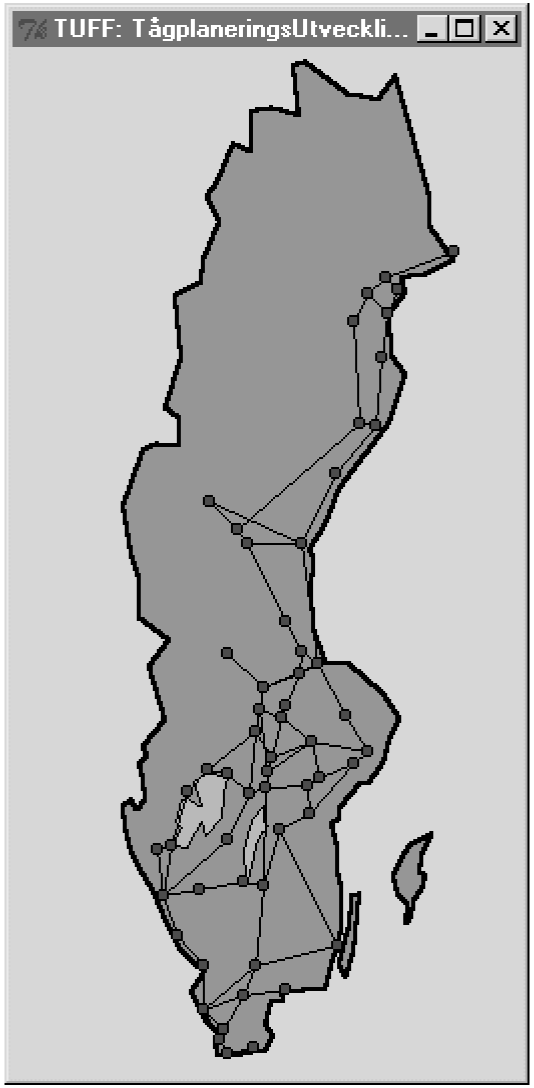
This version of the application does not take into account several features of the real world problem i.e:

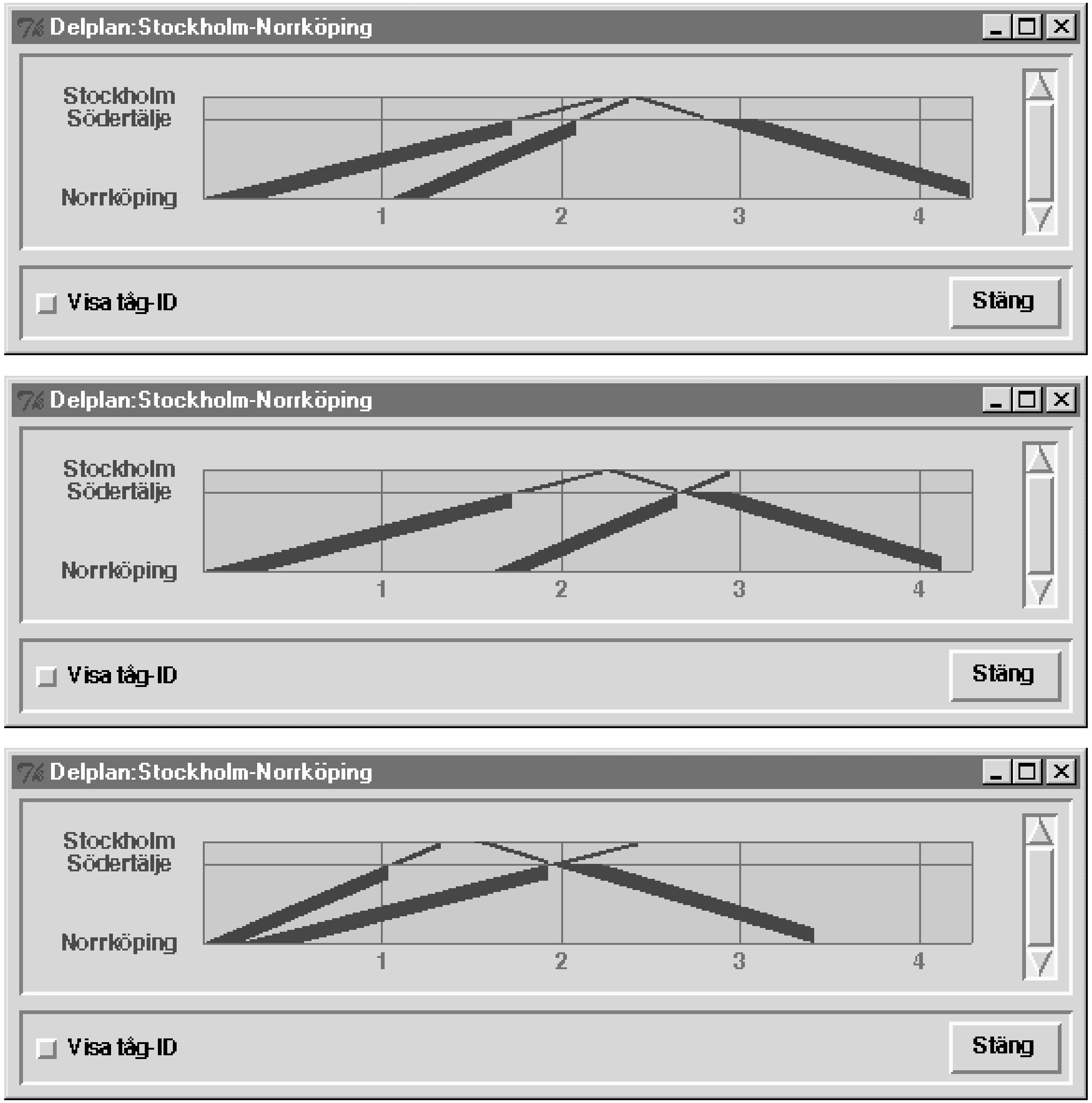
- Limitation of track resources at meeting places
- Limitation on vehicle and staff resources
- Our examples were rather underspecified compared to a more realistically posed problem

Conclusions

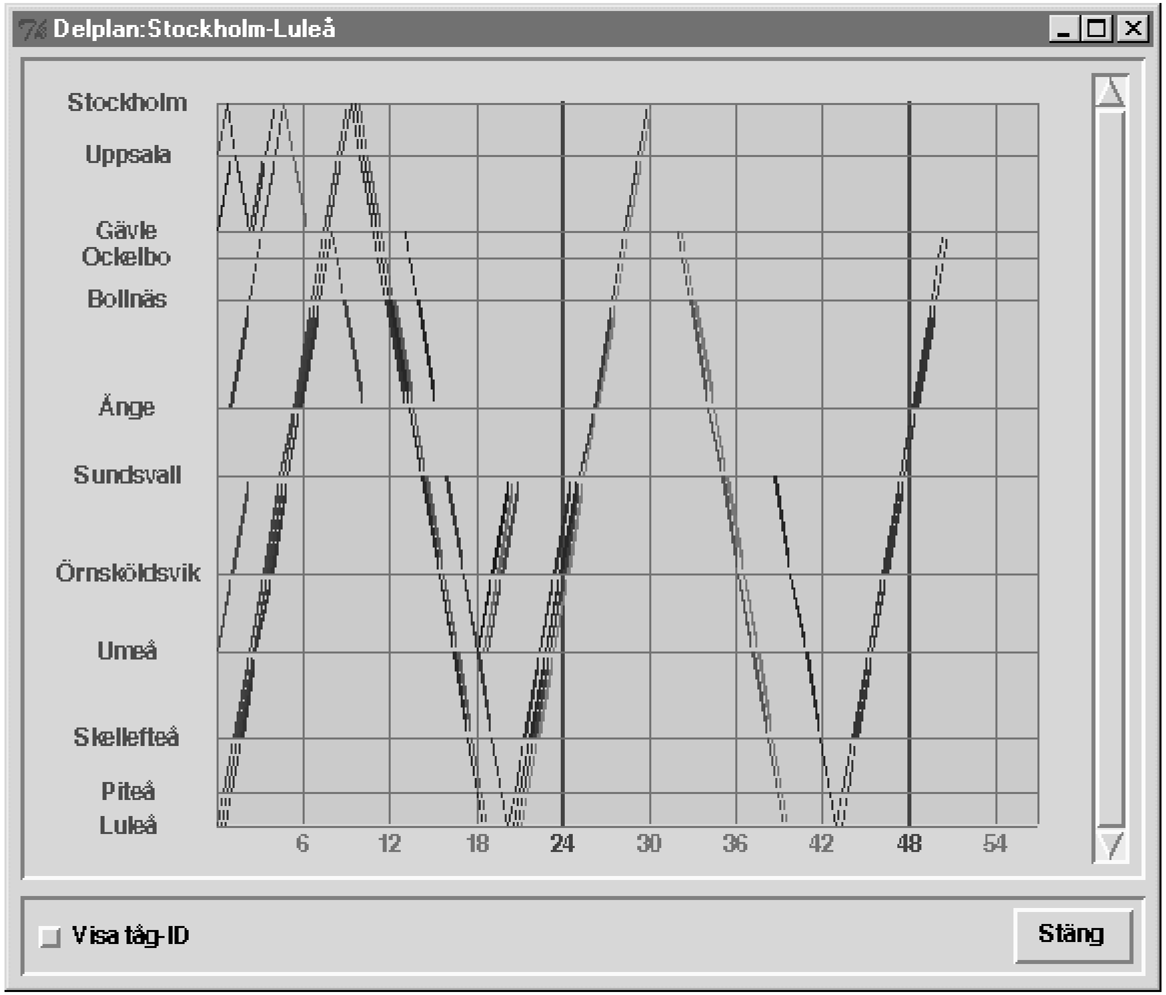
- We have describe a novel constraint model for scheduling train trips on multi directed track networks
- We were able to approach realistic problems sizes with a very straight forward implementation even if we solved only a small part of the real problem
- We found Oz to a flexible and stable tool

Sample Schedules

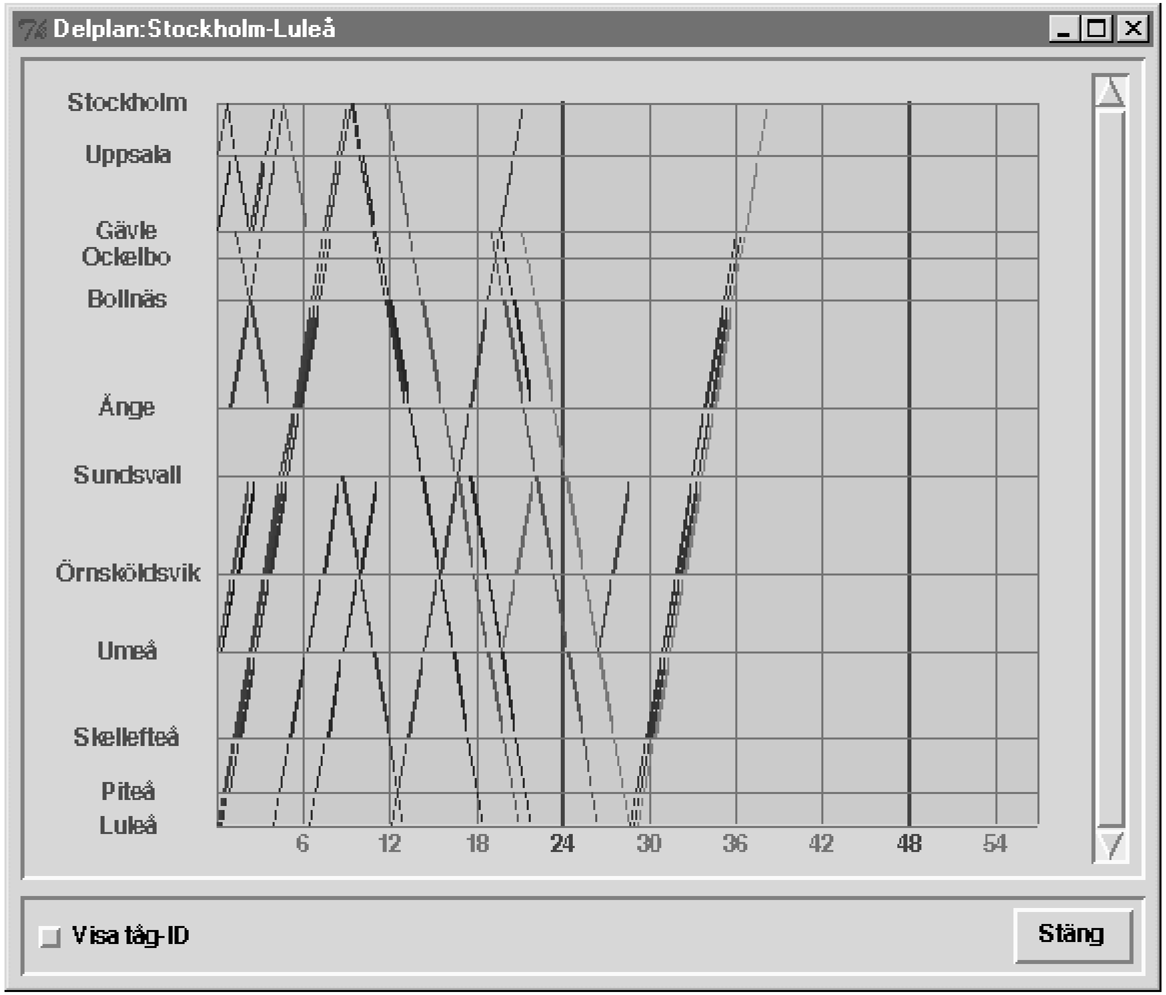




Some Larger Schedules



(Fig. 3)


Delplan:Stockholm-Luleå
Stockholm
Uppsala
Gävle
Ockelbo
Bollnäs
Ånge
Sundsvall
Örnsköldsvik
Umeå
Skellefteå
Piteå
Luleå
6
12
18
24
30
36
42
48
54
Visa tåg-ID
Stäng

(Fig. 4)