A CONSTRAINT MODEL FOR A CYCLIC TIME PERSONNEL ROUTING AND SCHEDULING PROBLEM

PER KREUGER, MARTIN ARONSSON

Abstract. We present a crew scheduling problem with time windows in the context of scheduling train personnel, which encompasses not only assignment of resources to tasks but also the introduction of extra tasks, passive journeys, depending on the problem instance. The representation of the problem is made as a constraint program, which relies heavily on some global constraints, notably a constraint for expressing non-overlap between rectangles on a surface. A search algorithm is described and we also point out some problems and deficiencies of the current model and its computational behaviour.

Acknowledgement. The research reported in this paper was funded by the Swedish government agencies of NUTEK and Vinnova as part of the Complex systems programme during 1999-2001 and working on the problem domains of and in cooperation with the Swedish State Railway (SJ) and Green Cargo AB.

The authors wishes to thank Jolanta Drott who acted as our main contact and sponsor within the rail industry for several years.

Keywords: Crew scheduling, time windows, cyclic time, resource assignment

SICS Technical Report T2001:04
ISSN: 1100-3154
ISRN: SICS-T–2001/04-SE

Date: September 2001
Contact: martin@sics.se.

1. Introduction

This paper presents a finite domain model for the construction of circuits of tasks to be performed by resources, here traveling personnel. The model centers around a set of tasks constrained by time and location parameters, together with constraints between different types of tasks such as distance between breaks etc. The problem arises in the area of traveling personnel within the rail industry.

This problem is related to some of the vehicle routing problems studied by e.g. Solomon [Sol87], but differs in some respects. One large additional constraint is that some extra tasks, passive tasks, sometimes must be introduced to move the resource, i.e. some person, from a location A to another location B. This happens if there are more resources coming to A than departing from A, and there are more resources departing from B than arriving to B. The person must travel with some other train inside the scope of the current plan (together with some other resource), or some other kind of transport. In the first case there is some synchronization between trains and passive tasks. Since trains are decomposed into sub tasks, the introduction of passive tasks may lead to that the constructed circuits of tasks are tied together by synchronization relations.

The core model has been implemented in SICStus Prolog [SICStus], and it is shown that the implementation behaves correctly. However, the propagation of the constraint primitives used are too weak and/or the search too naive to handle large problem instances. In order to scale up to realistic problem sizes, better propagation must be achieved, and some of the main causes of inefficiency are pointed out.

The rest of the paper is organized as follows. Section 2 gives some terminology, section 3 defines the problem and section 4 outlines the constraint model. Section 5 describes the search algorithm while section 6 gives a short example. In section 7 and 8 we summarize our conclusions and point out some problems with the present model and implementation.

2. Terminology

The problem studied in this paper occurs in the railway transportation industry. To describe this domain, we use the following terminology.

A *trip* is a transportation task (train), departing at a specific location and arriving at another location, with a given path through a net consisting of tracks and locations. The trip has as parameters its departure time interval (or time window) and a fixed traversal time.

The schedule should be repetitive, i.e. the schedule could be for a typical day or week that starts all over again when reaching the end of the schedule. The time period that the schedule is made for is called the *cycle time period*, and the time point that is set equal to the start of the schedule is called the *cycle time*. In our presentation the start time of the schedule will be set to time 0.

Tasks are parts of trips, possibly whole trips. Tasks typically start and end at stations where personnel can change from one task to another, or where a trip starts or ends. Tasks cannot be too long since there are union regulations that puts an upper limit on the length. The tasks can be of four kinds:

- *working tasks*,
- passive journey tasks inside the scope of the current plan, here called *passive tasks*,
- passive journey tasks performed by external transports outside the scope of the current plan, here called *foreign transports*,
- *break tasks*, which have certain constraints on placement, both in time and relative to each other as well as other types of tasks.

Passive journey tasks occur when the resource (person) travels together with some working task, without working on it (i.e. there are already enough resources assigned to that task).

A foreign transport denotes some other kind of transportation than a passive journey task. Typically this can be a taxi, a bus etc, something that is not part of the current scheduling problem. This may happen for several reasons, e.g. if the net of working tasks is unbalanced or the waiting time is too long at some station. With unbalanced we mean for example that there is a task ending at a location where there is no task leaving, or the extension of that which means that there is a working task going from one sub-net of tasks to another sub-net of tasks but there is no (working) task going back to the original sub-net. A foreign transport is modeled as a passive task which has to be mapped onto one of several given external transports (see section 4.4).

Breaks are tasks that are stationary, i.e. they are performed at the same location. There are certain regulations about the distribution of breaks and work tasks, e.g. the time between two breaks and regulations for the sum of the durations of the working tasks for each resource. We will not in this paper model the complete picture on this issue, only sketch a general form and give a simple example.

A *turn* between a task A and a task B is when the resource (person) is assigned B after A, without any working task, passive task or foreign transport in between.

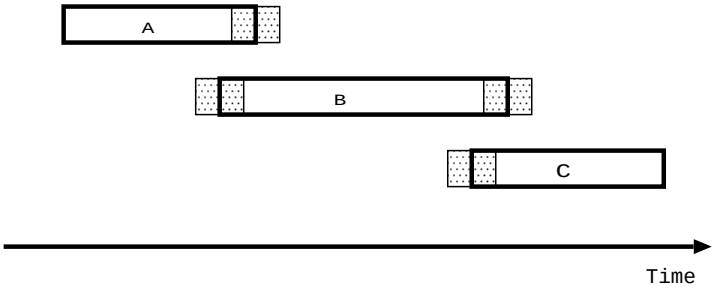
A *circuit* is a sequence of tasks, starting and ending at the same location and with a fixed time duration which is a multiple (larger than zero) of the cycle time period. Circuits must be closed, i.e. all tasks has exactly one task that they turn into, and each circuit has finite number of tasks.

3. Problem Formulation

The overall goal is to assign traveling resources (personnel) to tasks (transports) such that all given tasks are assigned sufficient resources. The assignments must be done such that union regulations and other restrictions are obeyed. The first issue is to split the trips given in the problem formulation into tasks, which we assume that there is a suitable algorithm for. This algorithm gives us a set of tasks together with precedence relations that relates tasks from the same trip to each other (typically restricting a task B following a task A to depart after A, but at most 5 minutes after A). Since the departure times and thereby the arrival times are not fixed, the turns in the solution must satisfy some time constraints, mainly that trips that are tied together by turns of tasks do not overlap in time in such a way that the turn is not possible to perform. In addition to that, law and union regulations regarding breaks, rests etc must be obeyed.

That the length of the circuits is limited makes this problem much more complicated than a classic flow problem, well studied in operations research (see for example [HL95] or [AMO]). The solution to handle limited circuit length in OR is to split the process of finding a solution into two steps, first apply a so called pairing algorithm to generate a set of individually feasible circuits and then apply an optimization algorithm to choose an optimal subset of circuits that assign sufficient resources to each task (see for example [AHKW]). This solution method has also been used in the railway industry (see for example [AHD97]). But if the time table is not fixed, turns may constrain the departure times and thereby influence if other turns are possible to make or not, hence the choice of a turn is not local and therefore the pairing algorithm must not only generate a set of feasible circuits but also a set of relations stating under which conditions the circuit is possible to apply. For example, let the time window for the arrival time of a trip A partially overlap the time window for trip B, and let the time window for the arrival time

of trip `B` partially overlap the departure time window of a trip `C`. Making the turn from trip `A` to `B` may destroy the possibility of making the turn from `B` to `C`.



The problem can be divided into three main sub-problems:

- *Location continuity*, which states that the task's locations must fit, i.e. for each task in a circuit the successor task's start location must be the same as the previous task's end location.
- *Task cover*, i.e. every task must be a member of a circuit.
- *Time continuity*, i.e. no task may overlap in time with another task in the same circuit.

4. Formal model

This section presents the formal model. The model assumes that a number of primitive predicates are given (constraints in a constraint programming language), in which the model is defined. For a detailed presentation of the underlying language see [SICStus]. Generally we will use the letter W to denote vectors related to working tasks, P for vectors related to passive tasks and T for vectors related to both working tasks and passive task, where the passive tasks have higher index values than the working tasks. For example, the vector of working task start locations is called Wls, the corresponding vector for passive tasks is called Pls, and the joint vector is called Tls and is formed by

$$Tls = \langle Wls_1, \ldots, Wls_n, Pls_1, \ldots, Pls_K \rangle$$

so e.g. Tls_{n+i} corresponds to Pls_i.

4.1. Problem parameters.

The following parameters characterize the problem:

CT: Cycle time: encodes the length of the period for the cyclic time
M: The number of possible resources (circuits)
B: The number of break tasks in a circuit of length CT
N: The number of work tasks
K: The number of passive tasks (no-duty tasks)
Dom: Initial domains for start times and duration; Bounds of these domains are required to fall in the interval $\{0, ...CT-1\}$
Wls: Work task start locations, $Wls = \langle Wls_1, ..., Wls_N \rangle$ where each Wls_i is a location identifier
Wle: Work task end locations, $Wle = \langle Wle_1, ..., Wle_N \rangle$ where each Wle_i is a location identifier
Bd: Break task durations, $Bd = \langle Bd_{i,j} \rangle$ where each $Bd_{i,j} \in \{1, \ldots, CT-1\}$. In the example given, a break task's duration will be 60 minutes for a lunch break, and 600 minutes for a nights rest.

4.2. **Foreign transports.** We assume that there is a finite set E given, which contains the possible external transports that are available. Foreign transports are then modeled using this set E. The elements of E are tuples with four elements,, and each element is on the form $\langle Els_i, Ele_i, Es_i, Ed_i \rangle$ where

Els_i**:** is the external transport start location for transport i,
Ele_i**:** is the external transport end location for transport i,
Es_i**:** is the external transport start time for transport i,
Ed_i**:** is the external transport duration time for transport i.

4.3. **Problem variables.** This section defines the problem variables, i.e. the variables that we wish to find values to. The variables are collected into vectors as described above. Work tasks are initially constrained by the domains of their start times and durations. Passive tasks are constrained to take either one of the work task's start time and duration, or the value 0. Since the number of passive tasks that should be used is not known in beforehand, and since it is not possible to incrementally add passive tasks as they are needed, a pool of possible passive tasks is created. A passive task can either be unused, in which case it is assigned the resource 0 (a dummy resource) and turned into itself, or be mapped onto a working task in which case a resource is transported with the same working task as the passive is mapped onto, leading to the resource being available somewhere else in the net. This is illustrated in the example section later.

4.3.1. *Passive Task Start Locations.*

$$Pls = \langle Pls_1, \ldots, Pls_K \rangle$$

where each $Pls_i \in Wls$.

Each passive task must begin at a place from where a working task starts. If a passive task i is not used, then Pls_i is arbitrary.

4.3.2. *Passive Task End Locations.*

$$Ple = \langle Ple_1, \ldots, Ple_K \rangle$$

where each $Ple_i \in Wle$.

Each passive task must end at a place from where a working task end. If a passive task i is not used, then Ple_i is arbitrary.

4.3.3. *Task turns.*

$$Wn = \langle Wn_i \rangle_{i \leq N} = \langle Wn_1, ..., Wn_N \rangle$$

where each $Wr_i \in \{1, ..., N + K\}$, and

$$Pn = \langle Pn_i \rangle_{i \leq K} = \langle Pn_1, ..., Pn_K \rangle$$

where each $Pr_i \in \{0, ..., N + K\}$.

Wn and Pn encodes the successor task.

4.3.4. *Task resources.*

$$Wr = \langle Wr_i \rangle_{i \leq N} = \langle Wr_1, ..., Wr_N \rangle$$

where each $Wr_i \in \{1, ..., M\}$, and

$$Pr = \langle Pr_i \rangle_{i \leq K} = \langle Pr_1, ..., Pr_K \rangle$$

where each $Pr_i \in \{0, ..., M\}$.

Note that Pr_i may be assigned the value 0. This is used for the passive tasks that are not used in the problem, and are moved outside the "real" resources that are numbered from 1 and up.

4.3.5. *Task start times.*

$$Ws = \langle Ws_i \rangle_{i \leq N} = \langle Ws_1, ..., Ws_N \rangle$$

where each $Ws_i \in Dom(Ws_i)$,

$$Ps = \langle Ps_i \rangle_{i \leq K} = \langle Ps_1, ..., Ps_K \rangle$$

where each $Ps_i = \{0, ..., CT - 1\}$.

4.3.6. *Task durations.*

$$Wd = \langle Wd_i \rangle_{i \leq N} = \langle Wd_1, ..., Wd_N \rangle$$

where each $Wd_i \in Dom(Wd_i)$,

$$Pd = \langle Pd_i \rangle_{i \leq K} = \langle Pd_1, ..., Pd_K \rangle$$

where each $Pd_i = \{0, ..., CT - 1\}$.

4.4. **Constraints.** This section presents the constraints used in the model. The complete description of these constraints is given in [SICStus], of which some are explained in some detail below.

- $alldiff$, a constraint taking a vector of finite domain variables. It ensures that all the values assigned to the variables of the vector are distinct. It is syntactically defined as `all_distinct([X1, X2, ..., Xn])` in [SICStus].
- $disjoint2$, a surface constraint that assures that rectangles packed onto a surface (also a large rectangle) do not overlap. It is syntactically defined as `disjoint2([q(X1, Y1, D1, H1), q(X2, Y2, D2, H2), ..., q(X1, Y1, D1, H1)])` in [SICStus], where each `Xi` is the x coordinate and `Yi` is the y coordinate of the lower left corner of rectangle i, and `Di` is the length along the x-axis and `Hi` is the length along the y-axis of rectangle i. `Xi`, `Yi`, `Di` and `Hi` are all integers or variables, restricted to finite integer domains. By identifying the right side of the large surface rectangle with the left side, a cylinder is created, which is used to model the circular time described in the model. This is accomplished by using the parameter `wrap/4` of the $disjoint2$ primitive. For an in-depth description of the *disjoint2* constraint see [BC00].
- reified constraints, i.e. it is possible to form logical formulas from primitive constraints.
- *element*, a constraint often used to express index functions ($X_i = j$, where $1 \leq i \leq n$ is expressed as `element(I,[X1, X2, ..., Xn], J)`)

We now present the formulas that build the constraint model.

4.4.1. *Resource constraints.*

Condition 4.1. Resource identity

$$(\forall i, j \leq N + K)\,(Tn_i = j \Rightarrow Tr_i = Tr_j)$$

Resource identity guarantees that if a turn is made from task i to j, then they must be assigned to the same resource.

Condition 4.2. Task non overlap

This non overlap constraint guarantees that no task is overlapping any other task in time and resource, i.e. if tasks are assigned to the same resource, then they shall not overlap in time, and vice versa, if they overlap in time, then they must be assigned to different resources. It also assures that if a turn is made from task i to j, then there may not be any other task in between task i and j. This is accomplished by having a rectangle for each task i, whose height is 1 and whose

length TD_i is from the start of task i to the start of task j. Since the turns are not known, the duration TD_i is the union of the durations from i to all possible j initially.

$$disjoint2(\langle Ts_1, TD_1, Tr_1, 1\rangle, \ldots, \langle Ts_n, TD_n, Tr_n, 1\rangle)$$

where $\forall i, j \leq N + K : Tn_i = j \rightarrow TD_i = dist\,(Ts_i, Ts_j, CT)$.

The function *dist* calculates the distance in time between T_1 and T_2 and is needed since the time wraps around the cycle time. The definition is:

$$dist(T_1, T_{2,}, CT) = \begin{cases} T_2 - T_1 & if\, T_2 \geq T_1 \\ CT - T_1 + T_2 & otherwise \end{cases}$$

where T_1 and T_2 are time points and CT is the cycle period.

This constraint is sufficient to ensure the *task cover* property, as described in section 3.

Condition 4.3. Location continuity

$$\forall\,(i, j \leq N + K)\,(Tn_i = j) \Rightarrow Tle_i = Tls_j$$

This condition states the location continuity property as described in 3. Turning task i to task j implies that the end location of task i and the start location of task j must coincide.

4.4.2. *Scheduling constraints.*

Condition 4.4. Passive task elimination

$$\forall\,(i < K)\,(Pn_i = i + N) \Leftrightarrow (Ps_i = 0 \wedge Pd_i = 0 \wedge Pr_i = 0)$$

Passive tasks not used should not be part of the schedule. They are encoded by turning them into themselves, and zeroing the time, resource and duration.

Condition 4.5. Passive task synchronization

$$\begin{gathered}\forall\,(i < K)\,\forall\,(j < N + K + M)\,(Pn_i = j \wedge j \neq i + N) \Leftrightarrow \\ (\exists l \leq N(Wls_l = Pls_i \wedge Wle_l = Ple_i \wedge Ws_l = Ps_i \wedge Wd_l = Pd_l)) \vee \\ (\exists l(\langle Els_l, Ele_l, Es_l, Ed_l\rangle \in E \wedge Els_l = Pls_i \wedge Ele_l = Ple_i \wedge Es_l = Ps_i \wedge Ed_l = Pd_l)\end{gathered}$$

This condition ensures that if a passive task is used, it is mapped onto one of two possible transports, either a working task or an external transport given by the tuples in E. In both cases the time data (departure times and task durations) are synchronized.

Condition 4.6. Break tasks non overlap

$$disjoint2(\begin{array}{l}\langle Ts_1, Tr_1, Td_1, 1\rangle, ..., \langle Ts_N, Tr_N, Td_N, 1\rangle, \\ \langle Bs_{11}, 1, Bd_{11}, 1\rangle, ... \langle Bs_{MB}, M, Bd_{MB}, 1\rangle\end{array})$$

This constraint introduces breaks. Breaks are assigned to each resource by fixing their Tr variable to a value. It is implemented by the $disjoint2$ constraint, using resource on one axis and the time on the other axis. One may think of this as first submitting the breaks for each resource, and then try to place all the resources in the surface spanned by resource and time, without overlapping and without violating the break constraints.

Note that this constraint is similar to 4.2, but this constraint encodes each circuit with its breaks. Without breaks, it is sufficient with the condition 4.2.

Condition 4.7.

Break time constraints relate start times and durations for break tasks belonging to the same resource. They may also constrain the minimum and maximum distance between the time variables of the break tasks. The general form involves all breaks for the same resource (we assume that breaks do not have to be related between resources).

$$Bt_1(Bs_{11}, Bd_{11}, ..., Bs_{1B}, Bd_{1B})$$
$$\vdots$$
$$Bt_M(Bs_{M1}, Bd_{M1}, ..., Bs_{MB}, Bd_{MB})$$

The following constraint can serve as an example of a break constraint, but many more may be defined in order to fulfill law, union regulations etc.

$$\forall i \leq M : \forall j \leq B : dist\left(Bs_{ij} + Bd_{ij}, Bs_{i,(j+1 \bmod B)}, CT\right) > SmallPass$$

where $SmallPass$ is the least time a working period may be and $dist$ is as defined in condition 4.2. In the example below, $SmallPass$ is set to 120 minutes.

4.4.3. *Additional (redundant) constraints.*

Condition 4.8.

$$alldiff(Tn_1, ..., Tn_{N+K})$$

Each successor for each working task and passive task must be unique. This constraint is sufficient to encode that the tasks form circuits, but not to restrict the number or length of the circuits, and is subsumed by condition 4.2. The reason for this is that condition 4.2 contains all working tasks exactly once, and if they can be placed into the rectangle formed by the time-resource plane, then also the successor for each working task can easily be seen in the rectangle. However, this condition adds a lot of propagation during search, and it is also convenient to use it since the turns are explicit here, not implicit as in condition 4.2.

Condition 4.9.

$$(\forall i, j \leq N)\,(Ts_i \leq Ts_j < Ts_i + Td_i \Rightarrow Tr_i \neq Tr_j)$$

If the work tasks overlap, they cannot be allocated to the same task resource. This constraint is also subsumed by 4.2, but adds some propagation during search.

5. Search algorithm

The search algorithm described here is simple, and works in two steps: first generate the circuits and then fix the departure times. The latter step is implemented by a standard enumeration technique from the underlying SICStus system, and the first step is done by a specialized algorithm as described below in 1. The idea is to build the circuits one after another, starting with circuit number 1. The advantage is that the propagation on the variables in the Tr vector is good since actual values are given to each circuit. The main (and large) disadvantage is that the last circuits formed are very poor (large waiting times and few active tasks). The main procedure `search` iterates through the resources, and for each one of them picking a start task to perform and then makes a circuit by calling `make_circuit`. `make_circuit` in turn constructs the circuit by iteratively by picking a successor task to the last chosen task, by calling `choices` to get a list och possible choices sorted according to some cost function, the cheapest turn first. `make_circuit` then tries each one of the choices starting with the cheapest one. `make_circuit` stops when the circuit is closed. If the search goes beyond the circuit length, several constraints will become violated (4.6 among others) and backtracking will occur. Thus the circuit must be within the stated circuit length. `choices` first picks the subset `S` of `Tn` that are

Algorithm 1 circuit formation

```
proc search(Tn, Ts, Td)
              for R = 1 to M do
                  i = start(Tn, Ts, Td) % Pick one not assigned yet
                  Tri = R
                  make_circuit(i, i, Tn, Tr, R)
              end
          end
proc make_circuit(start, i, Tn, Tr, R)
              if   i = start
              then finish(R, Tr) % Restrict other elems in Tr
                                 % not to be R
              else Choices = choices(i, Ts, Td, Tn)
                   foreach j in Choices try
                       element(i, Tn, j) % Tn_i = j
                   make_circuit(start, j, Tn, Tr, R)
              end
          end
fun choices(i, Ts, Td, Tn)
             S = possible(i, Tn)
             VS = value_all(S, Ts, Td)
             VSS = sort(VS)
             return VSS
          end
```

possible candidates, evaluates them all and sorts them, returning a sorted list with the cheapest turn first.

Passive journey tasks are introduced at an high cost. When they are introduced, they must be mapped onto a working task going from the location where the resource currently is. The choice of waht to map this passive task onto (some other working task, or some foreign transport) is a crucial choice, since it determines where the resource will be in the next cycle of `make_circuit`. Currently, this choice is speculative in the sense that the first transport from the current location is chosen, which gives no guidance to a location where the resource is needed and may lead to more passive tasks than actually needed (see section 7).

6. Example

The following example is a very simple one. Nevertheless it demonstrates the model and some steps in the search process.

There is one trip going from Stockholm (CST) to Göteborg (G), and one back. We call these 15 and 21 respectively. Both these trips passes Hallsberg (HPBG), which is a resource depot and thus each trip that passes there is split into two tasks. These tasks are related to each other in two ways: the second task must start after the first arrives, since they are part of the same trip, and the second task must start after just a short time period (since the passengers on the train do not want to wait a long time for a train driver). The tasks are numbered 1 and 2 for trip 15 and 3 and 4 for trip 21. There is also one trip going from Hallsberg to Stockholm, which is called 39. Since this trip never passes a resource depot, it forms a single task, which is called 5. The data for the tasks is given in table 1. All times are given in minutes.

TABLE 1. Task data

Task	Trip	Duration	Departure	Arrival	From	To
1	15	77	900..929	977..1006	CST	HPBG
2	15	97	977..1018	1074..1115	HPBG	G
3	21	91	345..374	436..465	G	HPBG
4	21	70	436..495	506..565	HPBG	CST
5	39	83	1215..1244	1298..1327	HPBG	CST

TABLE 2. Mapping of stations to integers

Station name	Integer code
dummy	0
CST	1
HPBG	2
G	3

TABLE 3. Initial state of working tasks

Working task vectors	1	2	3	4	5
Wr	1..3	1..3	1..3	1..3	1..3
Wn	{2}\/(4..7)	{3}\/(6..7)	{2}\/(4..7)	{1}\/(6..7)	{1}\/(6..7)
Ws	900..929	977..1011	345..374	436..470	1215..1244
Wd	77	97	91	70	83
Wls	1	2	3	2	2
Wle	2	3	2	1	1

We will in the following use the notation `X..Y` to denote a domain of integers between `X` and `Y` inclusive.

The cycle period is set to 1440 which is the number of minutes in a day.

Each circuit shall also have one large break of 600 minutes, and two shorter breaks of 60 minutes duration. There must be at least 120 minutes between each break task. To remove symmetries between the two shorter breaks, an ordering constraint is added:

$$\forall i \leq M : Ts_{i,1} + 180 \leq Ts_{i,2}$$

Each location is also mapped to an integer, in this case as in table 2. There is also a dummy station, with number 0, which is used by passive transports that are not used by the solution.

After posting all the conditions presented in section 4.4, but before searching, the state of the main problem vectors is as presented in table 3, table 4 and table 5. The first table shows the state of the working tasks. Each resource may serve each task. Each turn Tn_i is restricted to tasks that departure from the same location that task i arrives at. As explained in section 4 vectors corresponding to passive tasks are appended to the corresponding vectors for working tasks, and thus the passive tasks 1 and 2 are represented as number 6 and 7 in the joint problem vectors. Therefore, members of Tn can have values larger than 5, for example Tn_4 and Tn_5 may turn to working task 1 or passive task 1 or 2, denoted by 6 and 7 in vector Tn. Note that the departure time for the passive tasks should be one of the departure times for the working tasks, or for the passive task 1 take the value 6 if passive

TABLE 4. Initial state for passive tasks

Passive task vectors	1	2
Pr	0..3	0..3
Pn	1..7	1..7
Ps	{1}\/(345..374)\/(436..495)\/ (900..929)\/(977..1018)\/(1215..1244)	{2}\/(345..374)\/(436..495)\/ (900..929)\/(977..1018)\/(1215..1244)
Pd	0..97	0..97
Pls	0..3	0..3
Ple	0..3	0..3

TABLE 5. Initial state for break tasks

Breaks	1,1	1,2	1,3	2,1	2,2
Bs	0..1259	180..1439	0..1439	0..1259	180..1439
Bd	60	60	600	60	60

Breaks	2,3	3,1	3,2	3,3
Bs	0..1439	0..1259	180..1439	0..1439
Bd	600	60	60	600

TABLE 6. Working tasks after search

Working task vectors	1	2	3	4	5
Wr	1	2	2	2	1
Wn	5	3	4	6	1
Ws	900..929	977..1011	345..374	436..470	1215..1244
Wd	77	97	91	70	83
Wls	1	2	3	2	2
Wle	2	3	2	1	1

task 1 is not used (analogously for passive task 2). We will later see that passive task 2 is not used in the solution.

The search for circuits is then performed, as described in section 5. After the turns are made we get the state presented in table 6, table 7 and table 8. Each working task has been assigned to a specific circuit, and all turns are fixed, but that the departure times are not fixed yet. Note that the passive task 1 has been mapped onto a working task (number 1), and therefore it has inherited the time data from that task. Passive task 2 is not used, and has therefore been turned into itself ($Tn_7 = 7$ which is the position of passive task 2 in the problem vector Tn) and assigned to the special circuit 0 with a duration of 0. The domains of the break tasks' time variables has also been pruned somewhat.

A complete solution to this problem is presented below as a Gantt schema, with time on the x axis and the circuit numbers on the y axis.

TABLE 7. Passive tasks after search

Passive task vectors	1	2
Pr	2	0
Pn	2	7
Ps	900..929	0
Pd	77	0
Pls	1	0
Ple	2	0

TABLE 8. Break tasks after search

Breaks	1,1	1,2	1,3	2,1	2,2
Bs	0..1184	180..1439	0..1439	507..689	687..869
Bd	60	60	600	60	60

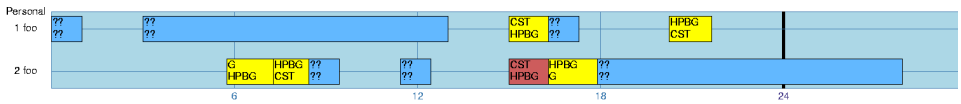
Breaks	2,3	3,1	3,2	3,3
Bs	1074..1214	0..1259	180..1439	0..1439
Bd	600	60	60	600

TABLE 9. Time table

Vector	1	2	3	4	5	6	7
Ts	900	977	345	437	1215	900	2

TABLE 10. Break schedule

Breaks	1,1	1,2	1,3	2,1	2,2	2,3	3,1	3,2	3,3
Bs	0	977	180	507	687	1074	0	180	360



The yellow bars (light grey) are working tasks, the blue bars (also marked with question marks) are break tasks, and the red bar (darker grey) is a passive tasks. One can see that the passive task maps onto the yellow task above. The corresponding departure times is given in table 9 and in table 10.

7. IDENTIFIED PROBLEMS

Two main problems has been recognized, one having with poor propagation in the `disjoint2` constraint and one modeling problem.

The `disjoint2` constraint is a source of inefficiency in our implementation of the model in SICStus prolog. This has at least two causes. The first cause has to do with synchronization of rows and departure times (X values) as stated in condition 4.5. Without any passive transports, each row in the *disjoint*2 constraint in condition 4.2 and condition 4.6 is independent and as tasks are assigned to resources (rows), the only time constraints that need to be checked are the ones on the same resource. When passive tasks are used, they synchronize rows which requires that information

from one row is propagated efficiently to the synchronized rows. Since passive tasks are needed in almost every problem instance, this propagation is needed to scale up the application. The weak propagation between rows seems to be a particular problem in conjunction with the break tasks.

The second cause has also to do with synchronization but also with interval and domain propagation. When two tasks are assigned the same resource, their resource variables becomes equal in 4.2 and 4.6. But since the information about them becoming equal is not propagated, only values pruned off the domains are propagated, and since values from variables in the Tr vector are not restricted (in the ideal case all resources are equal and thus each circuit may be assigned any resource) no propagation occurs when identifying Tr variables (they often have the same domain). Therefore there may be overlap between tasks having the same resource since the propagation is too weak to detect that the tasks are on the same row but the exact row is not determined yet. The situation would be better if `disjoint2` performed domain propagation on the Y axis, and even better with synchronization information was propagated between the variables for the Y position (the Tr variables in our case).

There are currently work in progress at SICS to improve propagation in the `disjoint2` constraint, with this use of the constraint in mind (forthcoming SICS Technical Report).

The modeling problem is that it is hard to introduce passive tasks in a way that the number of them are minimized, or rather, the total duration is minimized. The net of tasks must be balanced, i.e. there must be as many transportations to a node (station) as there are out from the node, and they must be distributed in time so that they fit into circuits. The problem is to find out where to add passive tasks, and where they should go. If the departure times are fixed, it is possible to determine a minimal set of passive tasks that balances the net. But with the introduction of time windows the problem of determining if a turn is possible or not ceases to be a local property and turns into a global problem. It is quite easy to construct an example where a turn may be a possible turn, if another turn is not made and thereby pushes the task forward or backward within its time limit and thereby making another turn impossible. In some sense, this is another side of the same problem described previously about the synchronization of rows in the disjoint2 constraint. This is so far an open problem, and passive tasks are introduced by heuristics, which may lead to impractical circuits.

8. Conclusions

We have presented a constraint model for creating crew circuits from train trips with time windows, and exemplified the process of finding a solution. The presented search algorithm is simple, and a better search algorithm based on for example the insertion heuristics as described in [Sol87] should improve the results. The implementation of the model pointed out some performance problems that must be addressed in order for the model to scale up. The problem of finding a minimal set of passive journey tasks to add to the set of working tasks must also be addressed for the model to be a usable in more realistic sized problems.

References

[AHD97] E. Andersson, C. Hjorring, J. Drott, *Crew Scheduling for Railways with Carmen Systems*, Technical report, Swedish State Railways, Stab Tågplanering, Stockholm Sweden, and Carmen Systems AB, July 1997

[AHKW] E. Andersson, E. Housos, N. Kohl and D. Wedelin: *Crew Pairing optimization*, in OR In Airline Industry, Eds Gang Yu, Kluwer Academic Publishers

[AMO] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network Flows, Theory, Algorithms, and Applications*, Prentice Hall, 1993

[BC00] N. Beldicieanu, M. Carlsson, *Sweep as a Generic Pruning Technique Applied to the Non-Overlapping Rectangles Constraint*, SICS Technical Report T2001:13

[HL95] F. S. Hillier, G. J. Lieberman, *Introduction to Operations Research*, McGraw-Hill, 1995

[SICStus] SICStus Prolog Users Manual, see http://www.sics.se/isl/sicstus.html

[Sol87] M. M. Solomon, *Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints*, Operations Research vol. 35, No. 2, March-April 1987