

TASK STRUCTURE ABSTRACTION

PER KREUGER, MARTIN ARONSSON, SIMON LINDBLOM

Swedish Institute of Computer Science (SICS)

Contact: Per Kreuger <piak@sics.se>

ABSTRACT. This paper explores the idea of using formalised abstraction as means of describing various operations on structured representations of planning, scheduling and resource allocation problems.

We define a notion of structures on tasks, relations and resources. Each task consists of task parameters and an inductively defined substructure. The task parameters denote domain variables constrained by the relations of the structure while resources formalises resource constraints, typically presupposing the semantics of some global constraint. A notion of consistency of such structures is briefly outlined.

We go on characterise properties of safeness and certain conservation properties on operations. We claim that such properties are relevant to problem domains where tasks on several different types or levels of resources has to be scheduled simultaneously or by partitioning the problem into several subproblems that has then to be coordinated.

Finally we present, in some detail, a model of such a problem from the rail transport industry formalised as several task structures. We show how operations on these can be seen as solvers for the various subproblems but also used to transform a specification or a solution of one subproblem into a specification of an other one. We claim that properties of such operations can be used to characterise reasoning in coordination problems.

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 co-operation with the Swedish State Railway (SJ) AND GREEN CARGO AB.

The authors wish to thank Juan Alonso and Jan Ekman of the DSPS group at SICS for valuable comments on the manuscript and Jolanta Drott who acted as our main contact and sponsor within the rail industry for several years.

SICS Technical Report T2001:5

ISSN: 1100-3154

ISRN: SICS-T-2001/5-SE

Date: July 2001.

Key words and phrases. Formal abstraction, task scheduling, resource management, hierarchical planning, approximate methods, problem coordination, constraint programming, formal models.

1. INTRODUCTION

Abstraction as mechanism to simplify complex planning problems have been studied in artificial intelligence [Yan97, Kno94]. However as a heuristic to solve scheduling and resource allocation problems we have not so far seen any general mechanism described in the literature. The authors of [CLS99] describe an approach to automatically generate meta heuristics using a particular notion of abstraction but do not consider it as a general modelling device.

This paper attempts to lay the the basis of a class of formal systems that can be used to describe scheduling and combined scheduling and resource allocation problems at several levels of abstraction in which operations on the structures of the system define mechanisms to transform a problem description into several subproblems or combine several into one etc. As examples we present a formal model of a train scheduling and vehicle routing problem.

We will proceed top down to define an abstraction hierarchy on a structure of problem parameters, variables, linear relations and resources. A problem is encoded using the notion of a task structure which consist of tasks, task relations and resource descriptors. Tasks are inductively defined data structures used to represent problems at several levels of abstraction, task relations are simple linear arithmetic constraints on the the parameters and variables of the tasks, and resources represent information about the resources allocated to or available to process the tasks.

2. STRUCTURES

Definition 2.1. Let a TASK STRUCTURE be defined as a triple

$$\langle \{Tsk_i\}_{0 \leq i < l}, \{Rel_j\}_{0 \leq j < m}, \{Res_k\}_{0 \leq k < n} \rangle$$

where each Tsk_i is a TASK, each Rel_j a RELATION and each Res_k a RESOURCE as defined below. Note that l, m, n may all be 1 in which case the structure is said to be EMPTY.

For a given task structure Str use the following notation to denote its components:

Notation 2.2. If

$$Str = \langle \{Tsk_i\}_{0 \leq i < l}, \{Rel_j\}_{0 \leq j < m}, \{Res_k\}_{0 \leq k < n} \rangle$$

let $tsk(Str)$ denote $\{Tsk_i\}_{0 \leq i < l}$, $rel(Str)$ denote $\{Rel_j\}_{0 \leq j < m}$ and $res(Str)$ denote $\{Res_k\}_{0 \leq k < n}$ and furthermore, for $0 \leq i < l \wedge 0 \leq j < m \wedge 0 \leq k < n$, let $tsk_i(Str)$ denote Tsk_i , $rel_j(Str)$ denote Rel_j and let $res_k(Str)$ denote Res_k .

2.1. Tasks.

Definition 2.3. Let a TASK, Tsk be a pair

$$Tsk = \langle \langle \tau(Tsk) \rangle_{\tau \in \Sigma(Tsk)}, sub(Tsk) \rangle$$

where we will refer to $\Sigma(Tsk)$ as the SIGNATURE of the task and where

$$\langle \tau(Tsk) \rangle_{\tau \in \Sigma(Tsk)}$$

denote its TASK PARAMETERS. Its SUBSTRUCTURE $sub(Tsk)$ is a task structure as defined above. We refer to $tsk(sub(Tsk))$ as the SUBTASKS of Tsk and the cardinality of the set of subtasks $|tsk(sub(Tsk))|$ as its ARITY. We will refer to tasks with empty substructure (i.e. with arity 0) as PRIMITIVE TASKS. We restrict in what follows our attention to tasks with finite signatures that are inductively built over primitive tasks.

Note. Task parameters typically include identifiers, resource requirements, release time points and various durations. These may be given as constants or as domain variables, i.e. variables whose values varies over a finite set or real interval. We do not in general require all the tasks in a structure to have identical signature.

For a given task use the following notation to denote its components:

Notation 2.4. Let $dom(\tau(Tsk))$ denote the DOMAIN of each task parameter $\tau(Tsk)$ of a task Tsk . In the case of a constant parameter we assume that dom returns the constant. We will identify a task Tsk for which $\Sigma(Tsk) = \{\tau_1, \dots, \tau_m\}$ with the $(m+1)$ -tuple $\langle \tau_1(Tsk), \dots, \tau_m(Tsk), sub(Tsk) \rangle$.

When referring to the components of the substructure of a given task $tsk_i(Str)$ of some structure Str we will for conciseness use the notation $tsk(tsk_i(Str))$ to denote its subtasks $tsk(sub(tsk_i(Str)))$, $rel(tsk_i(Str))$ to denote the relations $rel(sub(tsk_i(Str)))$ of its substructure and $res(tsk_i(Str))$ to denote the resources $res(sub(tsk_i(Str)))$ of its substructure.

We will use subscripts for the initial operator to denote individual elements of the substructure and finally use the even more concise notation $tsk_{ij}(Str)$ to denote each individual subtask $tsk_j(tsk_i(Str))$ in $tsk(tsk_i(Str))$.

2.2. Task relations.

Definition 2.5. Let a TASK RELATION, $rel_j(Str)$ of some structure Str be a (linear) relation of the following form:

$$c_1 x_1 + \dots + c_l x_l + c_0 = 0$$

where each x_k is either a task parameter $\tau(tsk_i(Str))$ for some $tsk_i(Str)$ in $tsk(Str)$ and some $\tau \in \Sigma(tsk_i(Str))$ or a local domain variable or constant and each c_k is similarly either a constant numerical task parameter $\tau(tsk_i(Str))$ for some $tsk_i(Str)$ in $tsk(Str)$ and some $\tau \in \Sigma(tsk_i(Str))$ or an arbitrary numerical constant.

Note. Note that any linear *inequality* can also be expressed in this form by introducing a local domain variable with lower and upper bounds.

Relations are typically used to state relations between a task and its subtasks and (partial) orders between the tasks of a structure or the subtasks of a task. The local variables and constants occurring in the relation may e.g. include resource parameters (as defined below) of a structure or in the case of relations occurring in a substructure of a particular task the task parameters of that (super) task.

Definition 2.6. Let an INSTANTIATION OF A DOMAIN VARIABLE be a relation that constrains it to a constant value.

Instantiations are of the form $\tau(tsk_i(Str)) = c$, i.e. a special case of a task relations.

2.3. Resources.

Definition 2.7. Let a RESOURCE, Res be defined as a pair

$$Res = \langle \langle \tau(Res) \rangle_{\tau \in \Sigma(Res)}, sub(Res) \rangle$$

with SIGNATURE $\Sigma(Res)$ and where we let $\langle \tau(Res) \rangle_{\tau \in \Sigma(Res)}$ denote its RESOURCE PARAMETERS and an ordered sequence $sub(Res)$ its SUB-RESOURCES. We will refer to the length of the sequence as the ARITY of the resource and resources that lack sub-resources, i.e with arity 0, as PRIMITIVE RESOURCES. We restrict in what follows our attention to resources with finite signatures that are inductively built over primitive primitive resources.

Notation 2.8. Identify a resource Res with signature $\Sigma(Res) = \{\tau_1, \dots, \tau_m\}$ with the $(m+1)$ -tuple $\langle \tau_1(Res), \dots, \tau_m(Res), sub(Res) \rangle$.

Resources typically model various disjunctive constraints¹. Its parameters generally include an identifier, various limits, a selection of elements of the signature of the tasks to which the resource is allocated and a type: e.g. unary disjunctive resource, unary disjunctive resource with setups, cumulative resource with limit, set covering etc. We will not at this point further restrict the class of constraints that resources encode but see section 5.1.3 and 5.2.1 below for typical uses of the construct.

3. TASK STRUCTURE CONSISTENCY

We will distinguish between three types of consistency for a task structure.

Definition 3.1. A task structure is said to be **LINEARLY CONSISTENT** if and only if there exists at least one set of instantiations for all the domain variables occurring in the structure such that all task relations are satisfied.

This property is typically decidable in polynomial time.

Definition 3.2. A task structure is said to be **RESOURCE CONSISTENT** if and only if there exists at least one set of instantiations for all domain variables occurring in the structure such that none of the resource limits is violated.

Formally, this property depends on the semantics of the constraints encoded by the types of the resource of the structure. Neither the complexity of or existence of a procedure to determine this property for some particular resource will concern us further here.

Definition 3.3. A task structure is said to be **consistent** if and only if there exists at least one set of instantiations for all domain variables that occurs in the structure such that all task relations are satisfied and none of the resource limits are violated.

Note. Note that the consistency of a task structure have not yet formally been related to the consistency of the substructure of its tasks. Instead we let such relations be defined through certain properties of operations on task structures defined below.

4. TASK STRUCTURE OPERATIONS

Task structure operations are functions on task structures. Although there are many conceivably useful such operations the following classes of operations are currently our prime concern.

4.1. Linear operations.

4.1.1. Weakening.

Definition 4.1. A function ρ is a **WEAKENING** of a task structure Str if and only if $tsk(\rho(Str)) = tsk(Str)$, $res(\rho(Str)) = res(Str)$ and each relation in $rel(\rho(Str))$ is implied by the relations in $rel(Str)$.

Note. The linear consistency of a task structure implies the linear consistency of all its weakenings.

4.1.2. Strengthening.

Definition 4.2. A function σ is a **STRENGTHENING** of a task structure Str if and only if $tsk(\sigma(Str)) = tsk(Str)$, $res(\sigma(Str)) = res(Str)$ and each relation $rel(Str)$ in is implied by the relations in $rel(\sigma(Str))$.

Notation 4.3. Strengthening operations that preserve consistency are referred to as **CONSISTENT STRENGTHENINGS**. Weaker properties of operations include being **LINEARLY** or **RESOURCE CONSISTENT** with the obvious interpretation.

¹For a systematic exposition of such resource constraints see [Bel00]

Definition 4.4. A special case of consistent strengthening operations are those that consistently instantiates all domain variable task parameters in the structure. Such strengthening operations are referred to as CONSISTENT ASSIGNMENTS. concretions

Note. Strengthening operations are dual to weakening operations in the sense that the linear consistency of any strengthening $\sigma (Str)$ of a task structure Str implies the linear consistency of Str itself.

Example. A scheduling algorithm can be seen as consistent assignment of a task structure representing a scheduling problem.

4.2. Structural operations.

4.2.1. Abstraction.

Definition 4.5. A function α is an ABSTRACTION of a task structure Str if and only if every task in $tsk (Str)$ occurs as a subtask in exactly one of the tasks in $tsk (\alpha (Str))$.

We consider in particular abstraction operations where the consistency of a task structure Str is implied by the consistency of $\alpha (Str)$. Such an abstraction is referred to as a SAFE ABSTRACTION of Str . Not all interesting abstractions will have this property however. Abstractions that do not are referred to as APPROXIMATIONS. Some approximations will fulfil weaker properties such as being LINEARLY SAFE, i.e. where the *linear* consistency of a task structure is implied by its the consistency of its abstraction.

Note. We do not in general state requirements on how an abstraction α on Str act on the relations or resources of Str but it is often possible to ensure safeness of the operation only if relations and resources of the Str are somehow translated to relations and/or resources of $\alpha (Str)$.

Example. Partitioning operations are examples of abstractions. In contrast a set covering is not an abstraction at all since each task must occur as subtask of a unique supertask.

Other examples include abstractions that aggregate tasks and resources to obtain a simpler problem. In some cases strong resource requirements in the abstract structure can ensure that the operation is safe, which is of course a very attractive property. For an a relatively elaborate example see section 5.2.1 below.

4.2.2. Concretion.

Definition 4.6. A function κ is an CONCRETION of a task structure Str if and only if the tasks in $tsk (\kappa (Str))$ are all subtasks in exactly one of the tasks in $tsk (Str)$.

We consider in particular concretions κ such that for a structure Str , the consistency of $\kappa (Str)$ is implied by the consistency of Str . Such a concretion is referred to as a CONSERVATIVE CONCRETION of Str .

Note. Concretions are dual to abstractions in the sense that for any abstraction operation α of a structure Str there exists a concretion κ_α such that $\kappa_\alpha (\alpha (Str)) = Str$

4.3. Other operations. We will also consider compositions of the operations defined above. It is often possible to prove stronger properties of composite operations than of simple ones. E.g. that all consistent instantiations of a particular concretion of an abstraction of a structure are also consistent instantiations of the original structure. Such properties are very useful when considering complex problems with several interrelated subproblems or when using approximations to restrict the search for a large problem.

An other class of operation include merging and splitting operations on structures and structure generating operations in general.

5. EXAMPLES

To illustrate the intended use of the task structures defined in section 2 we will now give an example of a train planning problem encoded as task structures.

The train planning problem consists of the following two² subproblems:

- (1) A track time slot scheduling problem
- (2) A vehicle routing and scheduling problem

The first problem amounts to fixing departure times and traversal times on tracks and arrival times and location usage durations at various locations in a rail network. This problem is subject to resource limitations on tracks and at locations and shares certain variables with the vehicle problem that consists of determining routes for vehicles travelling in the network.

5.1. Trip scheduling.

Definition 5.1. Represent a TRIP SCHEDULING PROBLEM as a task structure Ts where each of the tasks $tsk_i(Ts)$ is a trip task with signature $\{id, dep, trv\}$ where for any given task Trp , $id(Trp)$ encodes a unique identifier and $dep(Trp)$ and $trv(Trp)$ encodes the *departure time* and *traversal duration* of the trip respectively while its subtasks $tsk(sub(Trp))$ represent tasks on the track and location resources. For a pure trip scheduling problem there are generally no interesting relations between the trips so we will assume that the set of linear relations $rel(Ts)$ is empty.

The structure of the problem is instead given by the substructure of each trip and the the resources of the structure. The resources of a trip scheduling problem is given in section 5.1.3 below.

5.1.1. *Trip substructure.* In the simplest case the substructure of each trip encodes the resource requirements of each individual track traversal and location usage. This can be represented by encoding the individual track traversals and location usages as primitive tasks.

Definition 5.2. Let each TRACK TRAVERSAL TASK $tsk_{ik}(Ts)$ of Ts have signature

$$\{id, trk, dep, trv\}$$

and arity 0 where $id(tsk_{ik}(Ts))$ encodes a unique identifier and $dep(tsk_{ik}(Ts))$ and $trv(tsk_{ik}(Ts))$ encode the *departure time* and *traversal duration* of the trip $tsk_i(Ts)$ on the track $trk(tsk_{ik}(Ts))$.

Let each LOCATION USAGE TASK $tsk_{il}(Ts)$ of Ts have signature

$$\{id, loc, usg, arr, stp\}$$

and arity 0 where $id(tsk_{il}(Ts))$ encodes a unique identifier and $usg(tsk_{il}(Ts))$ the *resource requirement* of the task while $arr(tsk_{il}(Ts))$ and $stp(tsk_{il}(Ts))$ encode the *arrival time* and *duration of the stop* of the trip $tsk_i(Ts)$ at the location $loc(tsk_{il}(Ts))$.

Note. The trip is represented as a task that is related to its subtasks a job to its tasks in traditional scheduling problems.

5.1.2. *Linear relations.* Recall that $tsk_{ij}(Ts)$ denotes the subtasks for each trip task $tsk_i(Ts)$ in the structure Ts and require that for each trip, that the first of its subtasks $tsk_{i0}(Ts)$ represents a track traversal task and that each track traversal task is followed by a location usage task. Assume furthermore that the order of the indexes of the subtasks of each trip task encode the path traversed by the trip over the network and that each trip task $tsk_i(Ts)$ has arity is $2n_i$. Then the

²There is also a third subproblem in this domain that is described in a separate paper [AK01]. The level of detail required to represent this problem falls between the two described here.

following relations must hold between the subtasks of each trip $tsk_i(Ts)$ and thus occur among the relations $rel(sub(tsk_i(Ts)))$ of its substructure:

Condition 5.3.

$$\forall (0 \leq j < n_i) (dep(tsk_{i(2j)}(Ts)) + trv(tsk_{i(2j)}(Ts)) = arr(tsk_{i(2j+1)}(Ts)))$$

$$\wedge$$

$$\forall (0 < j < n_i) (arr(tsk_{i(2j-1)}(Ts)) + stp(tsk_{i(2j-1)}(Ts)) = dep(tsk_{i(2j)}(Ts)))$$

This condition simply states the relation between the arrival and stop time of an odd task (a location usage) to the departure and traversal times of any preceding and following track traversals.

Condition 5.4.

$$dep(tsk_i(Ts)) = dep(tsk_{i_0}(Ts))$$

$$\wedge$$

$$dep(tsk_i(Ts)) + trv(tsk_i(Ts)) = arr(tsk_{i(2n_i-1)}(Ts))$$

This condition relate the departure and arrival times of the trip with those of the first and last subtasks.

Note. Note how the signature elements are used as projection functions to access the parameters of the individual tasks.

The task parameters of the trip $dep(tsk_i(Ts))$ and $trv(tsk_i(Ts))$ are examples of *local* domain variables in the context of the substructure of the trip.

5.1.3. *Network resources.* Each subtask of a trip places a requirement on a specific resource. The resource constraints of the trip scheduling problem are subject to non overlap and cumulative constraints which are, of course, not linear. A straight-forward model encode individual tracks as unary resources with non overlap constraints in time and locations as cumulative resources representing e.g. capacity of the switching system at the location. Partition the resources $res(Ts)$ of Ts into two types: track and a location resources.

Definition 5.5. Let a TRACK RESOURCE have signature

$$\{serialize, id, rel, dur\}$$

and arity 0 where *serialize* encodes the fact that the resource can be used by at most one task at any given time. Furthermore, for any given track resource Trk , let $id(Trk)$ encode a unique identifier and let for each task $tsk_{ij}(Ts)$ served by Trk :

$$rel(Trk, tsk_{ij}(Ts)) = dep(tsk_{ij}(Ts))$$

$$\wedge$$

$$dur(Trk, tsk_{ij}(Ts)) = trv(tsk_{ij}(Ts))$$

encoding that the release time and duration of each track traversal task is represented by its *dep* and *trv* parameters respectively.

Let a LOCATION RESOURCE have signature

$$\{cumulative, id, lim, rel, dur, utl\}$$

and arity 0 where *cumulative* encode the fact that the resource can accommodate a limited number of simultaneous usages at any one time. Furthermore, for any given location Loc , let $id(Loc)$ encode a unique identifier, $lim(Loc)$ the limit on simultaneous usage and let for each task $tsk_{ij}(Ts)$ served by Loc :

$$rel(Loc, tsk_{ij}(Ts)) = arr(tsk_{ij}(Ts))$$

$$\wedge$$

$$dur(Loc, tsk_{ij}(Ts)) = stp(tsk_{ij}(Ts))$$

$$\wedge$$

$$utl(Loc, tsk_{ij}(Ts)) = usg(tsk_{ij}(Ts))$$

encoding that the release time, duration and resource utilisation of each track traversal task is represented by its *dep*, *trv* and *usg* parameters respectively.

Note. Note that *rel*, *dur* and *utl* are elements of the signature of the resources while *dep*, *trv*, *arr*, *stp* and *usg* are elements of the signature of the tasks that use the resources.

5.1.4. *Solutions to a trip scheduling problem.* A solver for trip scheduling problems is realised as a strengthening operation that fixes the departure and traversal times of each track traversal subtask and the arrival time and stop duration for each location usage subtask such that the resulting structure is consistent.

A weaker but often useful notion of a solution to a trip scheduling problem is a strengthening that implies that an arbitrary (further) instantiation of the resulting structure will be consistent. Such a structure can e.g. be used to do conservative rescheduling efficiently.

5.2. **Network abstraction.** The individual tracks in real network are relatively short and numerous. In addition many of the locations connected by these tracks are irrelevant to the scheduling problem since trains never stop there and the resource limit is one. In order to reduce the problem size we would like to be able to aggregate several tracks and locations into a compound resource. Such an aggregation can be realised as an abstraction operation on the task structure defined in section 5.1.

5.2.1. *Track abstraction.* Several consecutive tracks and intermediary locations on a path can be aggregated by representing them as an composite track resource. To achieve this we will map the corresponding subtasks of all trips that traverse the the sequence of tracks onto individual super-tasks with a more complex resource requirement. This is an example of an abstraction operation. Given certain properties of the parameters of the tasks allocated to this new resource and that the speed of the trip over such an composite track resource can be seen as constant, it is possible to prove that it is also a *resource safe abstraction*.

Definition 5.6. Let a sequence

$$\langle res_{l_0}(Ts), \dots, res_{l_{(2k)}}(Ts) \rangle$$

be a TRACK RESOURCE AGGREGATE of a structure Ts , if its elements are track and location resources from $res(Ts)$ and $res_{l_0}(Ts)$ is a track resource and the sequence represent a continuous path in the network, so that each track resource but the last one is followed by a location resource and each task $tsk_i(Ts)$ in Ts that use *some* part of this sequence also use *the whole* sequence.

We can then represent these resources as an composite track resource as follows:

Definition 5.7. Let a COMPOSITE TRACK RESOURCE over a track resource aggregate

$$\langle res_{l_0}(Ts), \dots, res_{l_{(2k)}}(Ts) \rangle$$

for a structure Ts have signature

$$\{serialized_headway, id, rel, dur, stp\}$$

and arity $2k + 1$ and let for any given composite track resource Cmp , $id(Cmp)$ encode a unique identifier and for each task $tsk_{ij}(Ts)$ served by Cmp :

$$\begin{aligned} rel(Cmp, tsk_{ij}(Ts)) &= dep(tsk_{ij}(Ts)) \\ &\quad \wedge \\ dur(Cmp, tsk_{ij}(Ts)) &= trv(tsk_{ij}(Ts)) \\ &\quad \wedge \\ stp(Cmp, tsk_{ij}(Ts)) &= hdw(tsk_{ij}(Ts)) \end{aligned}$$

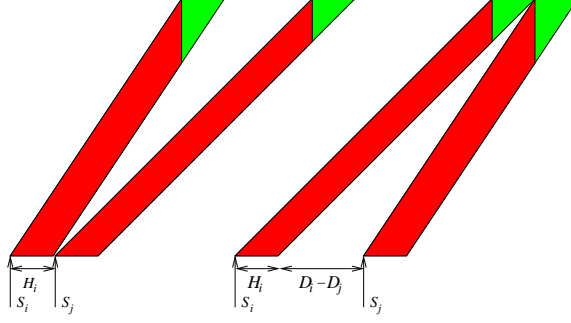


FIGURE 5.1. serialised_headway constraint

encode that the release time, duration and headway (a setup time parameter) of any composite track traversal task to which Cmp is allocated is represented by its dep , trv and hdw parameters respectively. Let furthermore the sub-resources of Cmp be

$$sub(Cmp) = \langle res_{l_0}(Ts), \dots, res_{l_{(2k)}}(Ts) \rangle$$

Let *serialized_headway* encode the fact that the resource can be used by several aggregate track traversal tasks at any given time as long as the following condition holds:

Condition 5.8.

$$\begin{aligned} S_i + \max(H_i, H_i + D_i - D_j) &\leq S_j \\ \vee \\ S_j + \max(H_j, H_j + D_j - D_i) &\leq S_i \end{aligned}$$

where $S_i = dep(Tsk_i)$ and $S_j = dep(Tsk_j)$ are the release times, $D_i = trv(Tsk_i)$ and $D_j = trv(Tsk_j)$, the traversal durations and $H_i = hdw(Tsk_i)$ and $H_j = hdw(Tsk_j)$, the headway parameters of each pair of tasks Tsk_i and Tsk_j .

Note. We assume here a new parameter hdw of each task scheduled on the composite resource. This parameter encodes a temporal headway between each pair of tasks. The headway is used to enforce a minimal time distance (a setup time) between any two trips traversing any part of the composite resource. The setup time between any two tasks is a function of both this headway and the traversal time of both tasks.

As can be seen from figure 5.1 this encoding guaranties a minimal time distance is between any two tasks even if the traversal times are different. As stated the condition is actually a bit stronger than necessary. It would be sufficient if the red areas in the figure did not overlap.

If our network include single track segments we need to, in addition to the above condition, for each pair of tasks traversing the track resource in opposite directions, ensure also a classic non-overlap condition involving the departure time and traversal duration for the (composite) single track traversal tasks (see figure 5.2).

These conditions are both modelled with a single global constraint (*serialised with setup*) in the underlying constraint system used in our implementations [C⁺95].

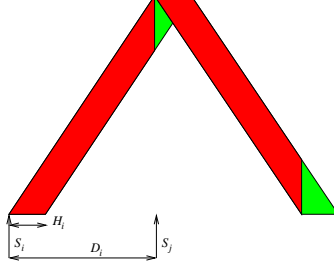


FIGURE 5.2. Non-overlap on single tracks

Definition 5.9. Let the COMPOSITE TRACK ABSTRACTION $Trp_i^\alpha = \alpha(Trp_i)$ of a trip task Trp_i with arity n_i over an composite track Trk map Trp onto itself unless

$$(\exists m) \left(\begin{array}{l} trk(tsk_m(Trp_i) = sub_0(Trk)) \\ loc(tsk_{(m+1)}(Trp_i) = sub_1(Trk)) \\ \wedge \\ \vdots \\ \wedge \\ trk(tsk_{(m+2k)}(Trp_i) = sub_{(2k)}(Trk)) \end{array} \right)$$

in which case it is instead a task with signature

$$\{id, trk, dep, trv\}$$

and arity $n_i - 2k - 1$ where $id(Trp_i^\alpha) = id(Trp_i)$, $dep(Trp_i^\alpha) = dep(Trp_i)$, $trv(Trp_i^\alpha) = trv(Trp_i)$. Let m_i represent the index of the primitive track traversal task such that

$$trk(tsk_{m_i}(Trp_i) = sub_0(Trk))$$

and map the subtasks $tsk(Trp_i)$ of the original trip onto $tsk(Trp_i^\alpha)$ such that

$$\forall (0 \leq j < m_i) (tsk_j(Trp_i^\alpha) = tsk_j(Trp_i))$$

and

$$\forall (m_i < j < n_i - 2k) (tsk_j(Trp_i^\alpha) = tsk_{(j+2k)}(Trp_i))$$

and let $Tsk_{im_i}^\alpha = tsk_{m_i}(Trp_i^\alpha)$ be a new COMPOSITE TRACK TRAVERSAL TASK with

$$\Sigma_{trv'} = \{id, trk, dep, trv, hdw\}$$

and arity $2k + 1$ where $id(Tsk_{im_i}^\alpha)$ denotes a unique identifier and $trk(Tsk_{im_i}^\alpha) = Trk$. Furthermore let its subtasks be the tasks originally using the aggregated resources, i.e.:

$$tsk(Tsk_{im_i}^\alpha) = \{tsk_{m_i}(Trp_i), \dots, tsk_{(m_i+2k)}(Trp_i)\}$$

and its relations $rel(Tsk_{im_i}^\alpha)$ analogous to those relating these task in the original trip task:

$$\forall (0 \leq j < k) \\ dep(tsk_{2j}(Tsk_{im_i}^\alpha) + trv(tsk_{2j}(Tsk_{im_i}^\alpha)) = arr(tsk_{2j+1}(Tsk_{im_i}^\alpha))$$

and

$$\forall (0 < j \leq k) \\ arr(tsk_{(2j-1)}(Tsk_{im_i}^\alpha) + stp(tsk_{(2j-1)}(Tsk_{im_i}^\alpha)) = dep(tsk_{(2j)}(Tsk_{im_i}^\alpha))$$

and $dep(Tsk_{im_i}^\alpha)$ and $trv(Tsk_{im_i}^\alpha)$ are new local variables related to the parameters of its substructure as follows

Condition 5.10.

$$dep(Tsk_{im_i}^\alpha) = dep(tsk_0(Tsk_{im_i}^\alpha))$$

and

$$trv(Tsk_{im_i}^\alpha) = trv(tsk_0(Tsk_{im_i}^\alpha)) + stp(tsk_1(Tsk_{im_i}^\alpha)) + \dots + trv(tsk_{2k}(Tsk_{im_i}^\alpha))$$

and let $hdw(Tsk_{im_i}^\alpha)$, finally be new a constant task parameter.

Note. The choice of this parameter will influence if this operation is a safe abstraction or not. If

$$hdw(Tsk_{im_i}^\alpha) \geq \max \left(\begin{array}{l} \max_{l=0}^{k-1} (trv(tsk_{(2l)}(Tsk_{im_i}^\alpha)) + stp(tsk_{(2l+1)}(Tsk_{im_i}^\alpha))) \\ trv(tsk_{(2k)}(Tsk_{im_i}^\alpha)) \end{array} \right)$$

and the speed of the trip over the composite track resource can be regarded as constant, then it is straightforward to prove that for each consistent assignment of the new variables $dep(Tsk_{im_i}^\alpha)$ and $trv(Tsk_{im_i}^\alpha)$ in the abstract structure there will exist a consistent assignment of the concrete structure fulfilling the relations of each composite track traversal task in the abstract structure.

Definition 5.11. Let a mapping α from a task structure Ts and a track resource aggregate

$$\langle res_{l_0}(Ts), \dots, res_{l_{(2k)}}(Ts) \rangle$$

of Ts to a structure $\alpha(Ts)$ be a TRACK RESOURCE AGGREGATE ABSTRACTION if an only if $rel(\alpha(Ts)) = rel(Ts)$ and

$$res(\alpha(Ts)) = \{Trk'\} \cup res(Ts) \setminus \{res_{l_0}(Ts), \dots, res_{l_{(2k)}}(Ts)\}$$

where Trk' is a composite track resource over

$$\langle res_{l_0}(Ts), \dots, res_{l_{(2k)}}(Ts) \rangle$$

for Ts and

$$tsk(\alpha(Ts)) = \{\alpha(tsk_i(Ts)) \mid tsk_i(Ts) \in tsk(Ts)\}$$

5.3. The vehicle routing and scheduling problem.

Definition 5.12. Represent a VEHICLE ROUTING AND SCHEDULING PROBLEM (c.f. e.g. [DHKK97, Sol87]) as a task structure Rt where each of the VEHICLE TASKS $tsk_i(Rt)$ have signature $\{nv, vt\}$ and represent a (cyclic) *route* taken by a number $nv(tsk_i(Rt))$ of vehicles of type $vt(tsk_i(Rt))$ through the network over $nv(tsk_i(Rt))$ successive periodic cycles of a predetermined length. Let furthermore the relations $rel(Rt)$ consist of one linear relation:

Condition 5.13.

$$\left(\sum_{vt(tsk_i(Rt))=vt(res_j(Rt))} nv(tsk_i(Rt)) \right) \leq nv(res_j(Rt))$$

where $vt(res_j(Rt))$ represents the vehicle type and $nv(res_j(Rt))$ a corresponding limit on the number of vehicles available of that type for each resource $res_j(Rt)$ in $res(Rt)$.

Note. Note that each vehicle task represents the route taken by several vehicles on successive periods. E.g. if the period length is one day and the $nv(tsk_i(Rt)) = 2$, then the time taken to complete a cycle is two days and the vehicle task can be served by two vehicles, one serving the first part of the route on the first day and the second part on the second day while an other vehicle is serving the second part of the route on the first day and the first part on the second. See figure 5.3.

5.3.1. *Vehicle task substructure.* We will use the subtasks $tsk_i(Rt)$ of a vehicle task $tsk_i(Rt)$ represent the individual TRIP TASKS served by the vehicles allocated to the vehicle task and augment the signature defined for trips in 5.1 with additional parameters relevant to the routing aspect of the problem:

Definition 5.14. Let a TRIP TASK $tsk_{ij}(Rt)$ have signature

$$\{id, dep, trv, on, of\}$$

with *substructure* and parameters $id(tsk_{ij}(Rt))$, $dep(tsk_{ij}(Rt))$, $trv(tsk_{ij}(Rt))$ as in section 5.1 while $on(tsk_{ij}(Rt))$ and $of(tsk_{ij}(Rt))$ are used to encode the time needed to ready the vehicle before and release it after the trip $tsk_j(tsk_{ij}(Rt))$. Let furthermore the relations be empty and the resource of the task substructure be defined as in definition 5.15 below.

Note. Note that the vehicle tasks here play the role of jobs and the trip tasks that of tasks in a classical sense. This in contrast to the trip scheduling problem formalised in section 5.1 where the role of jobs are played by the trip tasks.

The substructure of the individual trip tasks will not further concern us here but we would like to point out that for the particular formalisation of the routing problem given here no straightforward abstraction of a trip scheduling problem to a routing and scheduling problem will be safe. This means that a particular solution to the routing and scheduling problem may be inconsistent with every solution to the underlying trip scheduling problem.

5.3.2. *Vehicle resources.* If each vehicle task was served by exactly one vehicle the resource requirement on that vehicle could be represented as a precedence relation on the trip tasks served by the vehicle. Since optimal vehicle routes may however be arbitrarily long and need to be served by several vehicles the resource requirement for a given vehicle task becomes more complex. The following formalisation gives a condition that ensures that it is possible for a limited number of vehicles to serve all the trip tasks of a vehicle task.

Since each vehicle task encodes a route taken by several vehicles we will associate with each vehicle task a constraint that allow tasks to overlap in time up to a limit set by the number of vehicles serving the task. However, this is not a classical cumulative but rather amounts to a type of generalised precedence relation that ensures that it is possible to expand the schedule for a single period into a limited number of consecutive periods without any overlap. In figure 5.3 a requirement of this type is illustrated as an expansion of a single period enclosing all trip tasks departure times into two consecutive periods in which a circular route for a vehicle is completed without any overlap of tasks in time.

Definition 5.15. Let the resources $res(tsk_i(Rt))$ of each vehicle task $tsk_i(Rt)$ in a vehicle routing and scheduling problem Rt consist of one unique resource $R_i = res_0(tsk_i(Rt))$ with signature $\{finite_cycle, prd, lngt\}$ which encodes a relation (as outlined above) between all trip task that occur in $tsk(tsk_i(Rt))$. Let $prd(R_i)$ denote the period length after which the trip tasks in $tsk(tsk_i(Rt))$ are to be repeated and $lngt(R_i)$ the time taken by one vehicle to complete one full cycle through the network given in number of the periods required. Assume that the arity of each vehicle task $tsk_i(Rt)$ is n_i and let the resource type *finite_cycle* represent that the following condition must hold for all trip tasks in $tsk(tsk_i(Rt))$:

Condition 5.16.

$$\left(\begin{array}{l} \exists x_1, \dots, x_{n_i} \in \{0, \dots, lngt(R_i) - 1\} \\ (x_{i+1} \leq x_i + 1) \wedge \\ \forall j \in \{1, \dots, n_i\} \left(\begin{array}{l} dep(tsk_{ij}) + dur(tsk_{ij}) + tt_{ij} = \\ dep(tsk_{i((j+1) \bmod n_i)}) + x_i prd(R_i) \end{array} \right) \end{array} \right)$$

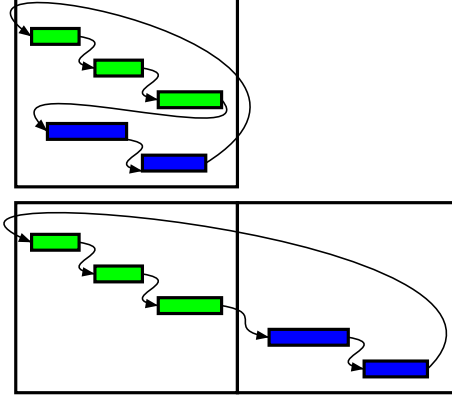


FIGURE 5.3. Cyclic route of trips within a time period and expanded to two periods to illustrate non-overlap condition

where tt_i is the transition time between task i and its successor $(i + 1) \bmod n$ in the cyclic sequence of trip tasks and x_i are discrete variables.

Note. The condition effectively counts the number of times we cross a period end.

5.3.3. Solutions to the vehicle routing and scheduling problem. Initially (before the solution of the problem) the arity of each vehicle task will be 1, as will the number of vehicles allocated to each task.

The relations of the structure may well be inconsistent since solving the problem involves partitioning the set of trip task in such a way that the relations are fulfilled.

Condition 5.17. Solutions represent the route of each vehicle as the sequence of trips served by the vehicle. If each trip is served by exactly one vehicle a solution constitute a set partitioning of the trips, if not, a set covering. In addition a particular non-overlap relation must hold between the subtasks (trips) belonging to any route. If the the departure and traversal times of each trip is fixed then this overlap conditions is straightforwardly represented as a set of linear inequalities, i.e. as relations in the substructure of each route. Such relations can e.g. be encoded by

$$dep(tsk_j(Rt_i)) + trv(tsk_j(Rt_i)) + of(tsk_j(Rt_i)) \leq dep(tsk_{j+1}(Rt_i)) - on(tsk_{j+1}(Rt_i))$$

for every consequent pair of trips $tsk_j(Rt_i)$ and $tsk_{j+1}(Rt_i)$ covered by a route Rt_i .

If the time parameters of the individual trips have not yet been fixed, a more complex (nonlinear) condition handling cyclic times may have to be considered.

The possibility of representing such relations over the timing parameters as a resource constraint on the subtasks of a supertask is a key mechanism in defining more complex operations on task structures. An example is a property on routes that ensures that every instantiation consistent with it will be possible to serve with the vehicle resource allocated to the route.

5.3.4. Transforming routing problems to and from trip scheduling problems. The transformation of a trip scheduling problem or solution to a vehicle routing and scheduling problem is a straightforward abstraction operation. Note however that this abstraction is not resource safe since a particular solution of a vehicle routing and scheduling problem may impose constraints that are not necessarily consistent with the resource constraints of the trip scheduling problem. A dual concretion

is possible to define imposing the overlap conditions of the routing problem as (additional linear or resource) conditions on the trip scheduling problem. Such an operation would be resource safe which is a very desirable property since this would mean that if we can solve the trip scheduling problem without violating the routing constraints then any solution of the resulting flow problem for the vehicles will be no worse than the one we started from.

6. CONCLUSIONS

We have defined a formal language in which a class of scheduling and resource allocation problems can be straightforwardly represented. The advantages of the representation include

- Subproblems can be extracted from the representation and solved individually
- Properties of such solutions can be used to further constrain the original problem
- Transformations of problem formulations and solutions are represented as operations with interesting properties that can be formally proved

The paper describes the basic constructs of the formalism and provides some examples of models from a domain that have been the focus of our group over the last several years. The model has been implemented as a decision support tool for analysing medium sized combined vehicle routing and scheduling and trip scheduling problems for the Swedish train operator Green Cargo. The implementation does not use the formalism defined above in its full generality but allows many useful problem transformations, (linear) relations and resource conditions not reported here.

Future work includes giving a complete formal model of this application and defining additional classes of operations as well as applying the formalism on other problem domains.

REFERENCES

- [AK01] M. Aronsson and P. Kreuger. A constraint model for a cyclic time personnel routing and scheduling problem. Technical report, Swedish Institute of Computer Science, 2001.
- [Bel00] Nicolas Beldiceanu. Global constraints as graph properties on structured networks of elementary constraints of the same type. Research Report R:00:01, SICS, Feb. 2000.
- [C⁺95] Mats Carlsson et al. *SICStus Prolog Users Manual*. Intelligent Systems Laboratory, Swedish Institute of Computer Science, 1995. ISBN 91-630-3648-7. For latest version see <http://www.sics.se/isl/sicstus/docs/>.
- [CLS99] Y. Caseau, G. Laburthe, and G. Silverstein. A meta-heuristic factory for vehicle routing problems. In *Fifth International Conference on Principles and Practice of Constraint Programming CP'99*, LNCS, pages 144–158, Alexandria, Virginia, 1999. Springer Verlag.
- [DHKK97] J. Drott, E. Hasselberg, N. Kohl, and M. Kremer. A planning system for locomotive scheduling. Technical report, Swedish State Railways, Stab Tågplanering, Stockholm, Sweden, and Carmen Systems AB, Jul 1997.
- [Kno94] C Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.
- [Sol87] M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35(2):254–265, March-April 1987.
- [Yan97] Q. Yang. *Intelligent planning — A decomposition and abstraction based approach*. Springer Verlag, 1997.