A Slot Representation of the Resource-Centric Models for Scheduling Problems

Roman Barták*

Charles University, Faculty of Mathematics and Physics
Department of Theoretical Computer Science
Malostranské námestí 2/25, 118 00 Praha 1, Czech Republic

bartak@kti.mff.cuni.cz
http://kti.mff.cuni.cz/~bartak

Abstract. Conventional approach to solving scheduling problems using constraint technology is based on a static formulation of the problem; in particular all the scheduled activities are expected to be known in advance. Such static formulation is not capable to solve many real-life planning and scheduling problems where the appearance of the activity depends on allocation of other activities during scheduling. In the paper we propose a slot representation of the scheduling problems that allows introduction of new activities during scheduling while preserving the advantages of constraint propagation. The research is motivated by real-life problems in complex process environments like alternative production formulas, transition patterns and non-ordered production.

1 Introduction

The conventional formulation of the scheduling problem is static; the scheduling task deals with allocation of the known activities to available resources over time respecting capacity, precedence and similar constraints. It is also a known wisdom to model the scheduling problem using CP framework in the same way as a constraint satisfaction problem is defined, i.e., first introduce all the variables, then post all the constraints, and finally label the variables respecting the constraints. However, there exist many scheduling problems that do not fit into this static framework because the appearance of the activity in the schedule depends on appearance or allocation of other activities. Consequently new variables describing newly introduced activities are generated and new constraints binding these variables are posted during the course of scheduling.

There exist several problem areas where a dynamic formulation of the model helps to solve the problem. Typically, for many synthesis tasks such as configuration, design, and model composition, the elements of the constraint problem (i.e., variables,

* Supported by the Grant Agency of the Czech Republic under the contract no 201/99/D057 and by InSol Ltd.

domains, and constraints) might change as the search progresses. This was reflected in the formulation of Dynamic Constraint Satisfaction Problem [6] where it is possible to activate/deactivate variables during the course of problem solving. Similar approach to model alternatives in the schedule via deactivating activities in the activity network was proposed in [9]. However, these methods still require all the variables and the constraints to be posted in advance so they are less useful if the number of alternatives is big, i.e., if the ratio between the active variables in the solution and the deactivated variables is low. Therefore, structural constraint satisfaction was proposed in [7] to solve the problems mainly in AI planning [8] where it is not clear in advance what a solution's graph will look like.

In the paper we describe an approach for solving the scheduling problems that need not the full capabilities of AI planning but still it is not possible to generate all the alternatives in advance due to memory consumption. In these problems we know the overall structure of the resulting schedule via estimating the maximal number of activities but we cannot post all the variables and constraints in advance because of large number of alternatives. Therefore we are speaking about scheduling enhanced by planning capabilities.

The research is motivated by real-life problems of industrial scheduling like alternative processing formulas, transition patterns, processing of by-products and non-ordered production. The problem area and the conceptual model to cope with it are overviewed in Section 2, details can be found in [2]. In Section 3, we describe an implementation of the resource-centric conceptual model using slots. Slots, which are well known in timetabling applications, are generalised here so they are not fixed to a particular time but they can be allocated to resources over time during scheduling. The slot behaves like a shell to be filled by an activity during scheduling and it takes over activity's attributes like the start time and duration. This gives us the flexibility of choosing the activities during scheduling while preserving the constraint propagation. In Section 4 we sketch how to do scheduling with such dynamic slots models and we describe the first experience with the implementation. We conclude with outline of future research.

2 Problem Area and Resource-Centric Models

In complex process environments like (petro)chemical, pharmaceutical, or food industries there exist many problems that cannot be tackled by conventional scheduling techniques to full extend. For example there are alternative resources to process given activity and there are even alternative processing formulas to produce the same item. Consequently, we get many *alternative production sequences* to satisfy given demand. This complicates modelling using traditional task-centric approach [4] where we need a list of activities to satisfy given demand. Moreover, the activities cannot be allocated to the resources in arbitrary order but they must follow user-defined *transition patterns*. Typically expensive set-up activities inserted between production activities must be assumed or appearance of the activity depends on previous activities processed by the resource, e.g., a cleaning activity every ten hours. Typically, set-ups are modelled using special set-up duration or simple set-up activity

like in [9] but the problem is when some low quality products, so called *by-products*, are produced during set-ups. In many real-life scheduling projects the by-products are neglected but many customers require processing of the by-products to be included in the schedule because the by-products can be used as a raw material (re-cycling) to decrease the overall production cost and they must be stored somewhere. Last but not least there is a *non-ordered production*, i.e., the production driven by marketing forecast rather than by the real custom orders. It is possible to use some low-preference tasks to model marketing forecast but we think that it is more appropriate to actively postpone the decision [5] about such production to the scheduling stage when more information is available, e.g. about load of resources.

The details of the problem area may be found in [2] where we also give a comparison of constraint models for solving such problems. The result of discussions there is that the resource-centric model is preferable in areas where all above described problems should be modelled. The resource-centric model is activity-based model where the activities are grouped per resources rather than per tasks. This simplifies expressing of the transition constraints that describe the sound order of activities in the resource. Moreover it allows strengthening the planning role of the scheduler necessary for modelling non-ordered production which may achieve about 90% of the overall production in some plants. Last but not least, the resource-centric model is easier to maintain because it may be implemented as a plug-in system where we can simply plug a new resource if necessary. Thus, the scheduling engine has a modular architecture and it can be easily ported to various production environments.

3 Slots in Scheduling

Our first implementation of the resource-centric model was purely dynamic, i.e., all the activities were generated dynamically during scheduling. This corresponds directly to the mixed planning and scheduling framework proposed in [1] that uses separate activity generator (planner) and activity allocator (scheduler). The dynamic implementation simplifies expressing of the constraints because the constraints are posted among known activities (no complicated constraint triggers are necessary). However, such fully dynamic environment has also drawbacks. Probably the main one is suppressing the role of constraint propagation. Because the constraints are posted among already generated activities only, the look-ahead techniques cannot be exploited to full extent and a special mechanism for deciding about new activities is necessary. Moreover, some activities may be generated twice or more times, e.g., a supplier generates consuming activity that is found later to be already present in the system. Thus, a mechanism for merging corresponding activities is necessary.

Because of above described difficulties with fully dynamic representation we turned our attention to a semi-dynamic representation that combines advantages of constraint propagation while still keeping the constraints in a relatively simple form. We generalise the slot representation for this purpose. Slot representation is not new to the scheduling community, however it is used mainly in timetabling and personal management applications where the slots are part of the original problem formulation. In [2] we also studied a time-line model with fixed slots but we argued against it

because it leads to huge, i.e., intractable models. However, if slots are generated per activities rather than per time slices with fixed duration then the memory consumption is comparable to the fully dynamic model.

We use slot as a shell to be filled by an activity during scheduling. Because we are generating schedules for a fixed period of time (rather than minimising makespan) we can estimate the number of activities processed by the resource. More precisely, we can compute the maximal number of the activities processed by the resource in given time period (by dividing the duration of the scheduled period by the duration of the shortest activity). Currently we are not using overlapping activities but we are still able to schedule cumulative resources by allowing single activity to process different quantities of items. To summarise it, we may generate a chain of consecutive slots for each resource and these slots will be filled by activities during scheduling. Moreover, we usually know the activity in the first slot that corresponds to the initial situation of the resource so by using the constraint propagation via transition constraints among consecutive slots (see later) we may further decrease the maximal number of slots.

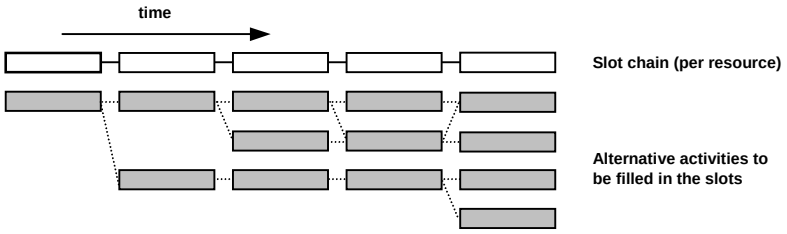

Fig. 1. In the slot representation, the slots in the chain are being filled by the alternative activities during scheduling.

Naturally, because the slots are introduced in advance it may happen that some slots are pushed after the schedule end during scheduling. In the resource-centric model this is not a problem because we may expect that the production continue after the schedule end. So, these superfluous slots describe the possible future production and we need not care about them during scheduling (even if the constraint propagation still reduces their domains).

3.1 Attributes and Constraints

Slots are not just empty shells but we propose to move attributes of the activities to the slots. Typically, all the activities use start time, duration, and completion time attributes so we may shift these attributes to slot. Because these attributes are used by all the activities in all the slots we can introduce them when the slot is generated, i.e., before the scheduling starts. Therefore we call them *static attributes*. There is one more static slot attribute common to all the slots and this is an *activity attribute* whose domain specifies which activities can be filled in the slot.

Naturally, the activity attribute is connected with other attributes via constraints, for example the constraint between the activity and the start time attributes may express the time windows for the activity, the constraint between the activity and the

duration attributes describe the duration of particular activity etc. We call these constraints that bind single slot attributes the *slot constraints* and they correspond to what we call resource constraints in [2].

By binding the activity attributes of the consecutive slots we can naturally describe the transition patterns. We call the constraints binding attributes of different slots of single chain the *chain constraints* and they correspond to what we call transition constraints in [2].

In some problem areas, the activities may use different sets of attributes, for example each activity may process different sets of items and we need a special attribute describing the quantity of each item. Because such attributes are specific for given activity we cannot introduce them before we know the activity in the slot. Therefore we call these *attributes dynamic* and they are introduced as soon as the activity in the slot is known, i.e., when the activity attribute becomes ground.

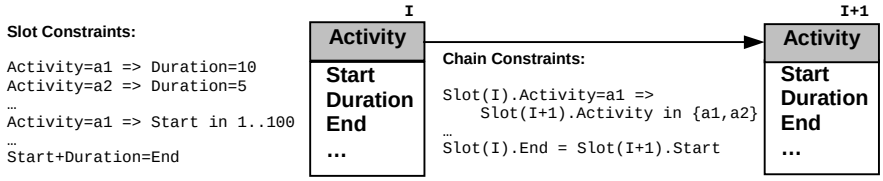

Fig. 2. Static slot attributes and constraints are generated in advance before the search starts. Thus, it is possible to exploit the power of constraint propagation.

A nice feature of the slot model is that it preserves the advantages of the constraint propagation because many constraints may be posted in advance. Moreover, by connecting the activity attributes to the constraint network we have planning capabilities for free. We mean that the decision about which activities are filled in the slots is automatic by means of standard constraint programming technology without necessity to use a dedicated planning module.

3.2 Posting Dynamic Constraints

The existence of dynamic slot attributes implies the existence of *dynamic constraints*, i.e., the constraints that are posted during the scheduling. It is clear that if there is a dynamic attribute among the constrained variables then the constraint cannot be posted before all the dynamic attributes in the constraint are generated. Nevertheless, there is no big problem with such dynamic constraints because they can be posted immediately after all the dynamic attributes in the constraint are generated.

Unfortunately, there exists another form of dynamic constraints when all the constraint variables are present in the constraint network but it is not known which variables should be connected by the constraint. This is the case of so called *inter-chain constraints* that bind attributes of slots in different slot chains. These constraints typically bind supplying and consuming activities in different resources, in [2] we call them dependencies. The main problem here is that neither the slots are fixed in time nor the activities in the slots are known. Therefore until we restrict “enough” the

domains of slot attributes we do not know which slots are connected and, thus, we cannot post the inter-chain constraints. We call the method of posting inter-chain constraints after filling and allocations of the slots in time the *lazy method*. The advantage of the lazy method is that only the necessary constraints are introduced, however, these constraints serve as tests only and there is no constraint propagation through them. Therefore, we propose to utilise more *eager method* of posting inter-chain constraints that uses a disjunctive expression of the conditional constraints. More precisely, instead of posting a single constraint connecting two slots we post the disjunction of similar conditional constraints binding several pairs of slots. The right constraint is selected (activated) during the search stage when conditional part of the constraint is satisfied. The big advantage of this method is that we can exploit the constraint propagation if the disjunctive constraint is implemented as a global constraint (the constraint propagation is weak in pure disjunction of constraints).

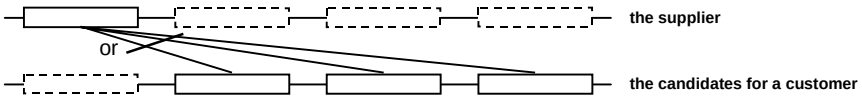

Fig. 3. The eager method of posting inter-chain constraints uses the disjunction of conditional constraints. When the domains of slot attributes are restricted "enough" the right constraint from this disjunction is selected.

4 Algorithm and Implementation

We have implemented the slot model in a CLP framework that provides a natural environment for posting the constraints during search. To post the dynamic constraints we use the same event mechanism that initiates the constraint propagation after the change of the attribute's domain. The overall structure of the program is as follows:

```
GENERATE ALL THE SLOTS
   in left to right order including all the static slot
   and chain constraints

ACTIVATE THE TRIGGERS
   for generating the dynamic attributes and posting
   the dynamic constraints including the inter-chain
   constraints

SEARCH THE SCHEDULE
   i.e. label the slot attributes in left to right order
```

Even if the above-described structure of the program is similar to the conventional method of solving CSP problems, it should be noted that we do not post all the constraints in advance but we allow introduction of new constraints during search (using the trigger mechanism).

We use left to right ordering both for generation of slots and for labelling. This allows us to decrease further the upper estimate of the number of slots[1] and it prevents "deep" backtracking as well[2].

5 Conclusions and Open Problems

In the paper we presented a slot representation for scheduling problems requiring some planning capabilities. The implementation of the proposed model prove itself to be enough general to cover most industrial scheduling problems. Probably the main difficulty of the current implementation is still big memory consumption. This is partly caused by using Prolog as an underlying solving platform, which saves all the data (including attributes' domains) for possible future backtracking. We are studying using of cuts to free the memory. Another reason for big memory consumption is using large disjunctive representation of the dynamic constraints. Our current research is oriented to design of more sophisticated triggers that postpone a bit the posting of the dynamic constraints and thus it decreases the size of the disjunctive expression.

References

1. Barták, R.: On the Boundary of Planning and Scheduling: A Study. *Proceedings of the Eighteenth Workshop of the UK Planning and Scheduling Special Interest Group*, Manchester, UK (1999) 28-39.
2. Barták, R.: Dynamic Constraint Models for Planning and Scheduling Problems. *Proceedings of the ERCIM/CompulogNet Workshop on Constraint Programming*, LNAI Series, Springer Verlag (2000), to appear.
3. Beck, J.Ch. and Fox, M.S.: Scheduling Alternative Activities. *Proceedings of AAAI-99*, USA (1999) 680-687.
4. Brusoni, V., Console, L., Lamma. E., Mello, P., Milano, M., Terenziani, P.: Resource-based vs. Task-based Approaches for Scheduling Problems. *Proceedings of the 9th ISMIS96*, LNCS Series, Springer Verlag (1996).
5. Joslin, D. and Pollack M.E.: Passive and Active Decision Postponement in Plan Generation. *Proceedings of the Third European Conference on Planning* (1995)
6. Mittal, S. and Falkenhainer, B.: Dynamic Constraint Satisfaction Problems. *Proceedings of AAAI-90*, USA (1990), 25-32.
7. Nareyek, A.: Structural Constraint Satisfaction. *Proceedings of AAAI-99 Workshop on Configuration*, 1999.
8. Nareyek, A.: AI Planning in a Constraint Programming Framework. *Proceedings of the Third International Workshop on Communication-Based Systems* (2000), to appear.
9. Pegman, M.: Short Term Liquid Metal Scheduling. *Proceedings of PAPPACT98 Conference*, London (1998), 91-99.

[1] The information about the initial activity can be propagated in the slot chain. This reduces the domains of the activity variables and, thus, it increases the minimal slot duration.

[2] We may expect that the time distance between the dependent slots is not very large and that there is a good propagation between close slots.