Modelling Transition Constraints

Roman Barták

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské námestí 2/25, 118 00 Praha 1, Czech Republic
bartak@kti.mff.cuni.cz

Abstract. Scheduling problems belong to a very successful application area of constraint programming partially thanks to many specialised global constraints defined for scheduling problems. In the paper we propose a concept of the transition constraint that is used to model a state change in resources. We describe and compare three models of the transition constraint. These models have been implemented and tested in the scheduling engine of Visopt ShopFloor system.

Introduction

Scheduling is a successful application area of constraint programming because a scheduling problem can be naturally modelled as a constraint satisfaction problem and many specialised constraints support such modelling and solving. A traditional scheduling problem is defined by a set of activities with precedence constraints and by a set of resources with capacity limits. The task is to allocate the activities to available resources respecting the precedence and capacity constraints (and minimising or maximising a given objective function). In most current scheduling systems, the resources are rather simple, usually only a capacity limit is used to describe the resource. In some systems, set-up times are assumed between the activities, i.e., a time gap must be inserted between two activities allocated to a single resource. In more complex industries, like chemical, pharmaceutical, and food enterprises, the resources are becoming even more complex and resource-centric models are more suitable to describe scheduling problems in such industries [1,2]. In particular, a complex transition scheme is given to describe behaviour of the resource. Such transition scheme defines allowed transitions between the activities and repetitions of the activities. For example, an activity of type B may follow an activity of type A but not the activity of type C and at least two activities of type B and at most five activities of type B can be processed in a sequence. Note that such transition scheme cannot be fully modelled using the standard concept of set-up times, where a long set-up time (or large cost) is used to describe the forbidden transition. Long set-up time does not forbid the transition, it makes it only less usable (depending on the objective function). Moreover, set-up times do not provide a mechanism to describe limited repetition of activities. Therefore, we propose a special transition constraint that is able to model fully the above transition scheme. This constraint has been implemented in the Visopt ShopFloor system [5] and as far as we know, it is the only

system that can cover such complexity of resources. In [6], Beck and Fox proposed a model for alternative activities that could be used to model the above transition scheme. However, this model requires introduction of all the alternative activities in the form of a process plan. As we argue in [2], it is less suitable when the number of alternative branches is large.

In this paper, we describe three models of the transition constraint. First, we sketch the problem area in more details. Then we explain the basic ideas behind the slot representation of resources and finally, we describe and compare the models of the transition constraints. Due to time reasons, we do not provide a precise comparison of the models - the informal comparison is based on our experience with real-life data.

Problem Area

Visopt ShopFloor is a generic scheduling system designed to address complex problems where traditional scheduling techniques failed [1,5]. A typical feature of our problem area is using *resources (machines) with complex behaviour*. This behaviour is described using *states* and *transitions* between the states. At each time, the resource can be at one state only or the resource is in the transition between two states (we allow transition time to be assigned to each transition). The transition scheme is typically described by a directed graph (Figure 1) or by a transition table (Figure 3). Note that using this general concept we can model set-ups, changeovers etc. either using transition times or using states. The set-up states are useful, if these states are connected to other resources. For example, if some by-product is produced during the set-up or if a worker (another resources) is required to do the set-up. Still, at the modelling level set-up states are handled like all other states. The number of states in the resource is not limited; some resources have just one state, in other resources the number of states can be very large (tens to hundreds).

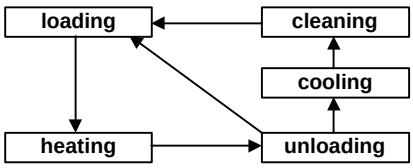

Fig. 1. A simple transition scheme for the resource.

Currently, we concentrate on batch production primarily so the schedule of the resource is described by a sequence of non-overlapping batches[1]. Each batch belongs to one of the resource states. The user may also restrict the length of the batch sequence in given state. For example a minimal number of five batches and maximal number of ten batches of some state S can be in sequence. It means that we cannot change the state S of the resource until at least five batches of this state are processed and we have to change the state S to another state (following the transition scheme) if ten batches of the state S have been processed.

[1] Continuous process can be decomposed into a sequence of batches.

Slot Representation

Traditional scheduling systems use a task-centric model of the problem where the activities are grouped per task. Because, the resources in our problem area are more complex than the traditional "capacity-only" resources, we prefer a resource-centric model [1] in the Visopt ShopFloor system, i.e., the activities are primarily grouped by the resource. We will use the word batch instead of activity in the following text.

The resource centric model is realised via *slots.* Slot is a shell filled by a batch during scheduling. For each resource we have a chain of slots and during scheduling these slots are being filled by batches. The difference from slots in timetabling is time location of slots. In timetabling, the slots represent fixed time intervals. In the Visopt solver, the slots may slide in time. Still, the order of slots is fixed but the slots may be shifted in time, e.g., if the slot is moved to later time then all the successive slots must be moved as well (Figure 2).

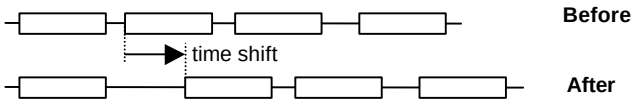

Fig. 2. Slots can move in time provided that the ordering of slots is preserved

The details about slot representation can be found in [2,5], we extract here only the slot parameters necessary to model the transitions. The batch type - state - that can be filled to the slot is described by a finite domain (FD) variable *state*. The constraint binding the state variables of two consecutive slots describes naturally the allowed transitions. In complex transition schemes, we need to restrict repetition of states in the slots. Therefore, we introduce a FD variable *serial_number* that indicates a relative position of the batch (in given slot) in the longest continuous sequence of the batches of the same state (see Figure 3).

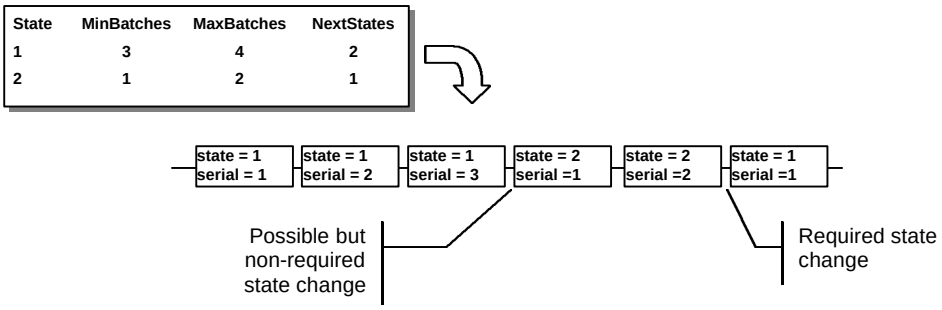

Fig. 3. Serial numbers in slots indicate the position of the batch (slot) in the sequence of slots of the same state.

In many problems, the maximal number of batches is not restricted for some states. Typically, set/up, cleaning, maintenance, and similar states have upper limit for the number of batches (usually, it equals to one), while processing states have no upper

limit (but usually they have the lower limit greater than one for technology reasons). Because, we are generating schedules for a fixed time horizon [5], we can estimate the maximal number of batches of given state in the schedule, e.g., by dividing the schedule duration by the batch duration. Thus, we can expect that the maximal number of batches is finite (i.e. different from supreme) for all the states.

It may seem that if we assume the maximal number of batches to be either one or supreme[2] then we can use a simple transition scheme without constraints restricting repetitions. Simply, the minimal number of batches of state S can be modelled as a single batch of newly introduced state S' such that S must be preceded by S'. Maximal number of batches of this newly introduced state S' equals one. After batch joining, the minimal number of batches equals one in all the states. If maximal number of batches equals one then the transition to a different state is forced.

However, such batch joining works just for one resource. Note that the batches of different resources are also connected e.g. using supplier-consumer dependencies. If we introduce a new "joined" batch then it is not clear how the batches should be connected (see Figure 4). Thus, we really need to model a transition scheme in its generality.

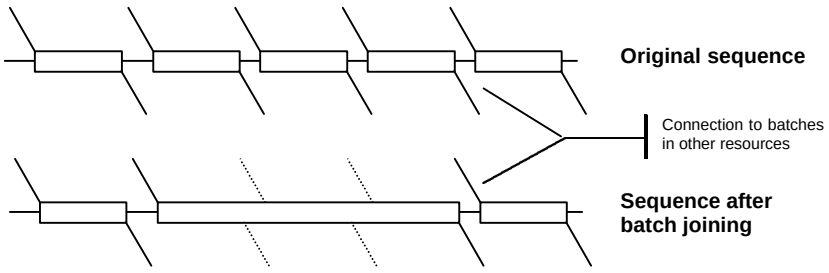

Fig. 4. Batch joining cannot substitute fully the general transition scheme because it is not clear how the joined batch is connected to batches in other resources (dashed lines).

Constraint Models

Basically, the transition constraint connects the state variables and the serial numbers of two consecutive batches (slots) in such a way that the transition scheme is fulfilled. Because the user defines the transitions between the states we must be ready to cover an arbitrary state transition relation. In [3,4], we proposed filtering algorithms for general binary constraints, we call them tabular constraints, and these algorithms are used in the transition constraint as well. Note also that the tabular constraint is used to model the relation between the state and the minimal number of batches and between the state and the maximal number of batches.

In the following paragraphs, we describe three different models of the transition constraint. All of these models use some form of the tabular constraint.

[2] This is a frequent case but still many real resources do have a general transition scheme.

A basic logic model

The simplest model of the transition constraint describes the state transition and the repetition restriction separately using the tabular constraints. For simplicity reasons, we use a meta-formulation of the constraints where the tabular constraint is integrated to existing primitive constraints like implication and comparison. Nevertheless, it is not a problem to separate the constraints using auxiliary variables, i.e., to follow the syntax of a particular underlying solver.

First, we define the state transition constraint via the tabular constraint. Basically, it is a general binary constraint with the domain defined using the table NextStates. The index of the variable indicates the ordinal number of the batch (slot).

$$state_{i+1} \text{ in } \{state_i\} \cup NextStates(state_i)$$

Now, we can define how the serial number changes in the next batch (slot). Simply, if the states in the batches (slots) are identical then the serial number is increased, otherwise the serial number is set to one (we are starting to count batches of another state). Because exactly one of the preconditions of the following implications holds, one of the conclusions must hold as well (constructive disjunction can be used there).

$$state_i = state_{i+1} \Rightarrow serial\_number_{i+1} = serial\_number_i + 1$$
$$state_i \neq state_{i+1} \Rightarrow serial\_number_{i+1} = 1$$

Finally, we need to connect the serial number and the state variable in each batch to model the repetition restriction. The serial number cannot be greater than the maximal number of batches of given state, this can be modelled using a tabular constraint, where the table MaxBatches describes the relation. A similar table MinBatches describes the lower limit for the number of batches. If the serial number in the batch (slot) is lower than the minimal number of batches then the state in the next batch must be identical to the state in the current batch.

$$serial\_number_i \text{ in } 1..MaxBatches(state_i)$$
$$serial\_number_i < MinBatches(state_i) \Rightarrow state_i = state_{i+1}$$

The above model is a straightforward formulation of the transition constraint and if we have the tabular constraint, such model can be implemented easily. However, this model suffers from weak domain filtering, i.e., there remain inconsistent values in the domain of variables. In particular, setting a wrong state in the batch is undesirable.

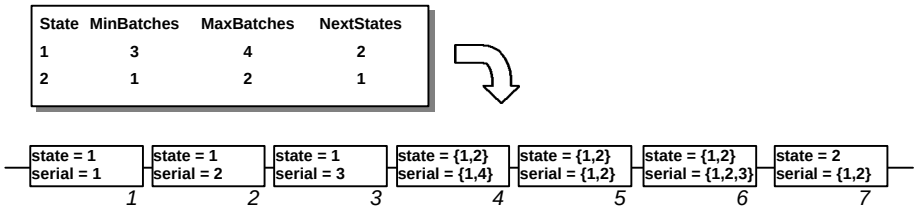

Fig. 5. A locally (arc) consistent sequence of batches that is not globally consistent - the seventh batch must be in the state 1.

A simple tabular model

The reason for weak propagation in the basic logic model is hidden in separation of the state and serial number variables. If we look at Figure 5, we can see that in the batch 6 there is an inconsistent serial number 3. This value is deduced from the value 2 of the serial number in the batch 5 for the state 1. Locally, this is a correct decision, i.e., all the constraints are locally consistent. The problem is that the serial number 2 in batch 5 belongs to state 2. To keep information about the connection between the serial numbers and the states we propose to encode the state and the serial number into a compound serial number. The following formulas can be used to define the encoding:

$$\text{separator} = 10^{\left\lceil \log_{10} \max\{MaxBatches(i) \mid i \in states\} \right\rceil}$$

compound_serial = separator*state + serial

The complete transition scheme can then be converted into a simple transition table describing the transitions in terms of compound serial numbers. As we can see at Figure 6 such table describes state transitions as well as MinBatches/MaxBatches restrictions. For example the compound serial number 13, that represents a third batch of state 1, can either go to 14, i.e., to the fourth batch of state 1, or to 21, i.e., to a first batch of state 2. But, if the compound serial number is 11 or 12 then we can continue with 12 or 13 respectively so the state is not changed. Note also that the domain filtering is now complete, i.e., all inconsistencies are removed (Figure 6).

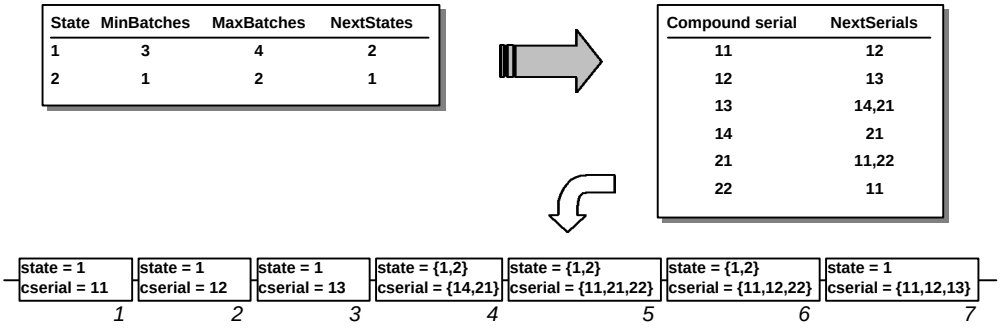

Fig. 6. Conversion of the transition scheme into a transition table for compound serial numbers.

Now, the transition constraint is modelled using a single tabular constraint between two consecutive compound serial numbers. The state can be decoded from the compound serial number easily[3].

```
compound_serial_{i+1} in NextSerials(compound_serial_i)
state_i = integer(compound_serial_i / separator)
```

[3] States play role in other scheduling constraints so the state variable should be preserved. The serial numbers are used just to model transitions so it is not necessary to decode them from the compound serial number.

A compound model

The simple tabular model provides a complete domain filtering for the transition constraint. It is also easy to implement it provided that we have a tabular constraint. Unfortunately, in large-scale real-life problems, the size of the induced transition table for the compound serial numbers can be very large. Assume that we have 300 states and the maximal number of batches per state is slightly less than 1000. Then, the induced transition table has about 300 000 rows. Moreover the structure of such table is not very compact so we cannot use the rectangular representation of the domain as proposed in [4]. Thus we decided to combine the representation using the compound serial numbers with the intentional model of the transition constraint.

The new filtering procedure uses the original transition scheme, i.e., the tables NextStates, MinBatches, and MaxBatches. For efficiency reasons, the values in MinBatches and MaxBatches tables are expressed as compound values, i.e., they include the state. The filtering procedure computes the induced transition table during runtime so it does not need to keep the table in memory. Thus, we can achieve the same pruning as the simple tabular model (all inconsistencies are removed) while keeping reasonable memory consumption. The following code describes the basic structure of the filtering algorithm for the transition constraint.

```
procedure TransitionConstraint(FromState,FromSerial,ToState,ToSerial)
    ToSerial := relevant_serials(ToState,ToSerial)
    NewFromState := NewToState := NewFromSerial := NewToSerial := {}
    for each State from FromState do
     Serial := relevant_serials({State},FromSerial)
     if nonempty Serial then
          NextSerial := increase(Serial,MaxBatches(State)) ∩ ToSerial
          if nonempty NextSerial then
          // transition to identical state
             NewFromState := NewFromState ∪ {A}
             NewToState := NewToState ∪ {A}
             NewFromSerial := NewFromSerial ∪ decrease(NextSerial)
             NewToSerial := NewToSerial ∪ NextSerial
          end if
          NextSerial := start_serial(NextStates(State)) ∩ ToSerial
          if nonempty NextSerial & MinBatches(State)≤max(Serial) then
          // transition to different state
             NewFromState := NewFromState ∪ {A}
             NewToState := NewToState ∪ NextStates(State)
             NewFromSerial := NewFromSerial ∪
                              (MinBatches(State)..MaxBatches(State))
             NewToSerial := NewToSerial ∪ NextSerial
          end if
     end if
    end for
    FromState in NewFromState
    ToState in NewToState
    FromSerial in NewFromSerial
    ToSerial in NewToSerial
end procedure
```

It may seem that the efficiency of the dedicated filtering algorithm is not as good as efficiency of the simple tabular model. However note, that the main complexity of the

tabular model is hidden in the tabular constraint. Because, the new filtering algorithm mimics behaviour of the filtering algorithm for tabular constraints described in [3], there is no significant decrease of efficiency. Additional work done during filtering is balanced by using smaller and more compact tables compared to the tabular model.

Discussion and conclusions

In the paper, we propose and discuss three models for a special transition constraint that is useful in modelling complex resources. All these models have been implemented using the clpfd library of SICStus Prolog [7] and tested in the real-life scheduling system Visopt ShopFloor [5]. First, we defined the transition constraint using a general concept of tabular constraints. This concept provides very good domain filtering but for resources with many states (and many slots), the memory consumption of such model is unacceptable. Therefore, we returned to a basic logic model of the transition constraint. Unfortunately, this model does not filter as good as the simple tabular model (see Figure 5). In fact, missing propagation led to "infinite" solving times of some real-life problems with restrictive transition scheme. Therefore we proposed a special transition constraint that mix the advantages of both logic and tabular models: good domain filtering and reasonable memory consumption. This compound model is used in the current version of Visopt ShopFloor system. We also use a variant of the basic logic model there to describe counters over the sequence of batches [5]. Counter is a generalisation of the transition scheme forcing some transitions after a given number of batches (perhaps of different states). Thus, the counter can be used to model maintenance and cleaning batches that typically appear after a specified number of production cycles; its detail description is out of scope of this paper.

References

1. Barták, R.: Conceptual Models for Combined Planning and Scheduling. *Electronic Notes in Discrete Mathematics*, Volume 4, Elsevier (1999).
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).
3. Barták R.: A General Relation Constraint: An Implementation. *Proceedings of CP2000 Post-Workshop on Techniques for Implementing Constraint Programming Systems*, Singapore (2000).
4. Barták, R.: Filtering Algorithms for Tabular Constraints. *Proceedings of CP2001 Workshop CICLOPS*, Paphos, Cyprus (2001), 168-182.
5. Barták, R.: Visopt ShopFloor: On the edge of planning and scheduling. Submitted to CP2002.
6. Beck, J.Ch. and Fox, M.S.: Scheduling Alternative Activities. *Proceedings of AAAI-99*, USA (1999), 680-687.
7. Carlsson M., Ottosson G., Carlson B. An Open-Ended Finite Domain Constraint Solver. *Proceedings Programming Languages: Implementations, Logics, and Programs* (1997).