Constraint Models for Complex State Transitions

Roman Barták
Charles University, Faculty of Mathematics and Physics, Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic

Constraint-based scheduling is an approach for solving real-life scheduling problems by combining the generality of AI techniques with the efficiency of OR techniques. Basically, it describes a scheduling problem as a constraint satisfaction problem and then uses constraint satisfaction techniques to find a solution. In this paper we study three constraint models describing complex state transitions that are going beyond the existing models of resources (machines) used in scheduling. These models can naturally handle any setup/changeover/transition scheme as well as special counter constraints imposed on the sequence of activities. The proposed models have been implemented and tested in the commercial scheduling engine of Visopt ShopFloor system.

Keywords: constraint satisfaction, scheduling, machine setups

1. INTRODUCTION

Scheduling is one of the most successful application areas of constraint programming [1]. This success is given by flexibility of constraint satisfaction technology supporting real-life constraints that are usually neglected in dedicated scheduling algorithms. On the other hand, existing scheduling techniques can be frequently encoded as so called global constraints and hence improve efficiency of constraint-based scheduling. This paper demonstrates the flexibility of constraint satisfaction by presenting three constraint models for complex transition schemes.

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 and time respecting the precedence and capacity constraints (and minimising or maximising a given objective function). In the majority of current scheduling systems, the resources are rather simple, usually only a capacity limit is used to describe the resource. In some systems, set-up or transition times are assumed between the activities. Typically, the transition time is modelled as a gap between two consecutive activities allocated to the same resource. This approach assumes that a transition matrix is given to describe the transition time between any pair of activities. In more complex industries, like chemical, pharmaceutical, and food enterprises, the resources are becoming even more complex. First, some transitions are forbidden while other transitions are forced. For example, it may be forbidden to allocate activity A right after activity B or if activity C is processed then the next activity must be activity D. Forbidden transitions could be modelled by using an infinite transition time in the transition matrix but the scheduling algorithm must be aware of this feature. The second feature of complex industries is using set-up activities rather than an empty gap between the consecutive production activities. For example when changing the shape of a product in the injection machine, it is necessary to change the mould which requires a crane and a person to do it. Hence, the transition from product A to product B cannot be modelled as an empty gap between two production activities but a set-up activity must be inserted between the production activities. The set-up activity is necessary there because it occupies other resources like the crane and the worker and this activity must be scheduled. Both above features, that is, forbidden/forced transitions and using set-up activities rather than transition times, can be handled by some models of sequence-dependent setup times like the one from ILOG Scheduler. Beck and Fox [8] proposed a model for alternative activities that could be used to model the above transition scheme too. However, this model requires introduction of all alternative activities in the form of a process plan. As we argued in [3], this approach with dummy activities is less suitable when the number of alternative branches is large. The main problem is memory consumption that is so large that the approach was not applicable in real-life problems that we were exposed to. Moreover, there is another feature which this approach as well as other existing techniques can hardly tackle and this is counting. Some real-life resources, for example in chemical and food industries, require processing a minimal and maximal number of activities of the same type in a sequence. For example, if we start processing activity A then we must process at least ten activities A and at most twenty activities A in a sequence. Such constraints are given by technological restrictions and we call them *local counters*. This feature is similar to sequential batching as described in [9]. In some application areas, *global counters* are also necessary. For example, in diaries it is necessary to clean pipes after processing a given quantity of milk. We can describe formally this feature as forcing some transition after processing a given number of activities of predefined types. The difference from a local counter is that activities of more than one type are counted.

In this paper, we propose constraint models to describe the above mentioned features of resources. These models are based on the resource-centric view of the problem [10] and on our slot-based architecture [3]. We use constraint satisfaction technology because of its flexibility to cover real-life restrictions. The proposed techniques have been implemented and tested within the scheduling engine of Visopt ShopFloor system [6].

The paper is organized as follows. First, we will describe in detail the problem to be solved. We will also include a simple motivation example demonstrating the typical features that we are interested in. Then, we will summarise the basics of constraint satisfaction technology. The main part of the paper will be dedicated to three constraint models describing the state transition scheme with local counters. At the end, we will sketch an extension of these models towards global counters. We will conclude by the discussion on practical applicability of the studied models.

2. PROBLEM AREA

Visopt ShopFloor is a generic scheduling system designed to address complex problems where traditional scheduling techniques failed [6]. A typical feature of our problem area is using *unary 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 a transition time to be assigned to each transition). The transition scheme can be formally described as a *state transition graph* which is a directed graph, where nodes correspond to states of the resource and arcs describe allowed transitions. The arcs can also be annotated by a time interval specifying the transition time between the states (for simplicity reasons this feature is not covered in the paper). Figure 1 shows a simple state transition graph with five states. Note that the state transition graph can naturally describe setups between the activities. We can either use a transition time to model the setup time or we can use a special setup state if a setup activity is necessary. In this second case, the setup states are handled like other states. In this paper we focus on techniques enforcing the given transition scheme. For example, cooling and loading activities are the only activities that can directly follow the unloading activity according to the state transition graph from Figure 1.

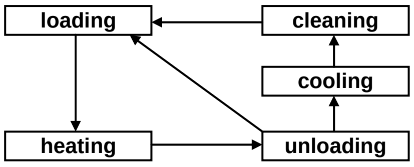


Fig. 1. A simple state transition graph for the resource

The concept of local counters described in the introduction can easily be added to the state transition graph by annotating each node (state) with two natural numbers. One number indicates the minimal number of activities that must be processed in a given state. The second number indicates the maximal number of such activities. Assume that numbers 5 and 10 are assigned to state A. Then, if the first activity of state A is processed then it must be directly followed by at least four and at most nine activities of state A before the state can be changed. Recall that we assume unary resources [1] where the activities do not overlap in time.

To summarize the problem – each unary resource is described by a set of states. Each state has some minimum and maximum number of activities that must be processed in a sequence, and a set of possible following states. The schedule – the sequence of activities – must satisfy both the transition scheme and the minimum/maximum number of activities per state. In the following section we will show a simple example that demonstrates practical applicability of the above concept of state transition graph. As far as we know, there is no scheduling algorithm that can handle the complexity of this problem (note also, that this paper focuses on the state transition schemes only, while the example problem has other features like time windows and resource synchronisation).

2.1. Example problem

Let us consider the following problem where the goal is to plan/schedule production on two machines in such a way that the user demands are satisfied. The machines can run either in a parallel mode or in a serial mode (Figure 2). In the parallel mode, the activities of both machines run in parallel and a worker is required. One final item is outputted from the activity and duration of this activity depends on the experience of the worker (see below). In the serial mode, the first machine pre-processes the item (3 time units) that is finished in the second machine (additional 3 time units). There is no delay for moving the item from the first resource to the second resource.

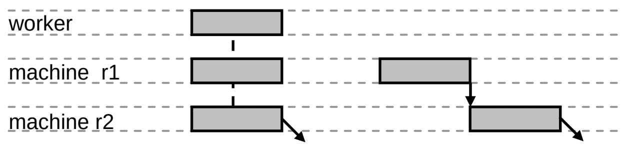


Fig. 2. The final product can be produced either via a parallel production when two machines run in parallel and a worker is required (left) or via a serial production when the item is pre-processed in the first machine and then finished in the second machine (right).

During the parallel production, a by-product is produced. This by-product can be recycled only on the second machine and we need three by-products to get a single final item. Recycling takes 2 time units and it must be done immediately after the three activities of the parallel processing.

The worker, who is necessary for parallel processing, is a beginner. After four production activities, the worker becomes experienced. The parallel production takes 3 time units for the beginner and 2 time units for the experienced worker. Moreover, the worker is available only in the following time windows (0..10), (30..40), (60..70).

Both machines require cleaning after eight production activities or sooner and cleaning must be done synchronously on both machines. Moreover, cleaning cannot be done if there is some non-processed by-product. At the beginning, both machines are clean so the first cleaning activity may appear after first eight activities.

The task is to plan/schedule production starting from time 0 in such a way that 5 final items are ready at time 20 and additional 25 items are ready at time 100.

The requested behaviour of both machines can be easily described as a state transition graph as depicted in Figure 3 (machine r1 is on the left, machine r2 is on the right). Similarly, the behaviour of the worker can be described as a state transition graph with two states: beginner and experienced.

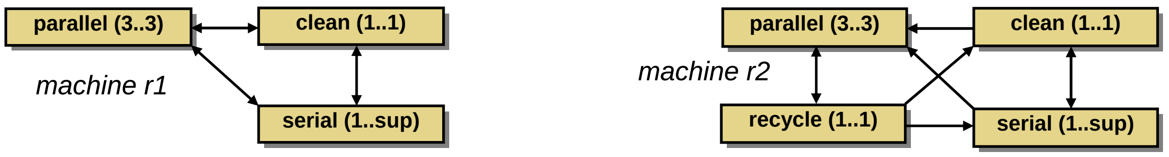


Fig. 3. Behaviour of the machines can be described as a state transition graph with a minimum and a maximum number of activities per state in brackets (sup means no upper limit on the number of activities).

As far as we know there is no scheduling algorithm/system that can cover the complexity of the above described example problem. Visopt ShopFloor solver that implements ideas to be presented in this paper can solve the problem (in less than one second on 1.7 GHz Mobile Pentium 4). Figure 4 shows the Gantt chart of the plan so the reader can verify that all the requirements are satisfied. In particular, recycling and cleaning activities are located as requested and duration of the parallel activities decreases when the worker became experienced (roughly at time 35).

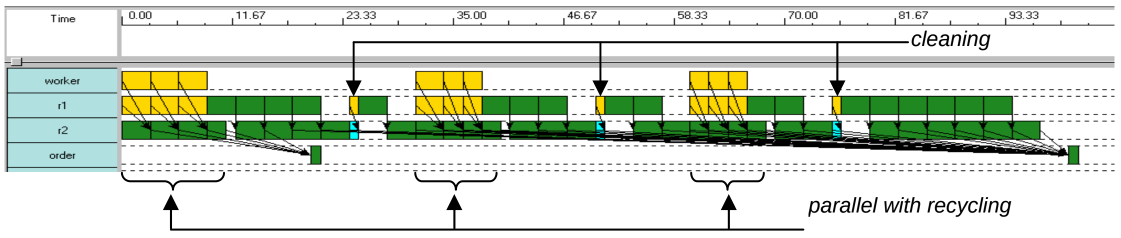


Fig. 4. Allocation of activities to time and resources (the Gantt chart) for the example problem

3. Constraint Satisfaction in a Nutshell

Constraint satisfaction problem [12] is defined as a triple (X, D, C), where X is a finite set of decision variables, D is a set of domains for these variables (domain is a finite set of values), and C is a set of constraints restricting possible combinations of values assigned to variables (constraint is any relation defined over the domains of constrained variables). The task is to find a value for each variable from the corresponding domain in such a way that all the constraints are satisfied.

The mainstream constraint satisfaction technology is based on the combination of domain filtering with search. The idea of domain filtering is to prune values that cannot be assigned to a variable in any solution or, more precisely, that violate some constraint. For example, assume that domains of variables A and B consists of elements {1,..,5}. If there is a constraint $A < B$ then value 5 can be removed from the domain of A and value 1 can be removed from the domain B because these values cannot be assigned to the variables in any solution satisfying the constraint. By removing these values, we made constraint $A < B$ locally (arc) consistent which means that the remaining values can satisfy the constraint (in some combinations). To remove as many inconsistencies as possible such domain filtering is repeated for all constraints until a fix point is reached. This process is called arc consistency or generalised arc consistency if n-ary constraints are assumed. If any domain becomes empty then no solution exists. In all other cases the search procedure splits the space of possible assignments by adding a new constraint (for example by assigning a value to the variable) and the solution is being searched for in sub-spaces defined by the constraint and its negation (other branching schemes may also be applied). Though (generalised) arc consistency can remove many incompatibilities from the problem specification, it usually does not lead directly to a solution and search is necessary.

The first step to solve a problem using constraints is its specification as a CSP – definition of a *constraint model*. Naturally, a single problem has more constraint models and the model that leads to stronger domain filtering is assumed to be better. In this paper we will propose two constraint models and a dedicated domain filtering algorithm for the constraint describing the state transition graphs.

4. SLOT REPRESENTATION

Traditional scheduling systems use a task-centric model of the problem where the activities are grouped per task [10]. Because, the resources in our problem area are more complex than the traditional “capacity-only” resources, we prefer a resource-centric model in the Visopt ShopFloor system, that is, the activities are primarily grouped by the resource.

In [3] we proposed to realise the resource centric model via *slots*. Slot is a shell to be filled by an activity during scheduling. For each resource we have a chain of slots and during scheduling these slots are being filled by activities. 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 and their duration depends on the activity allocated to the slot. Still, the order of slots is fixed but the slots may be shifted in time, for example if the slot is moved to later time then all the successive slots must be moved as well (Figure 5).

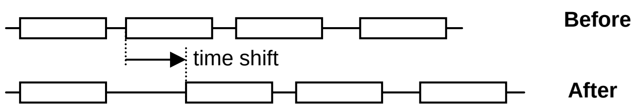
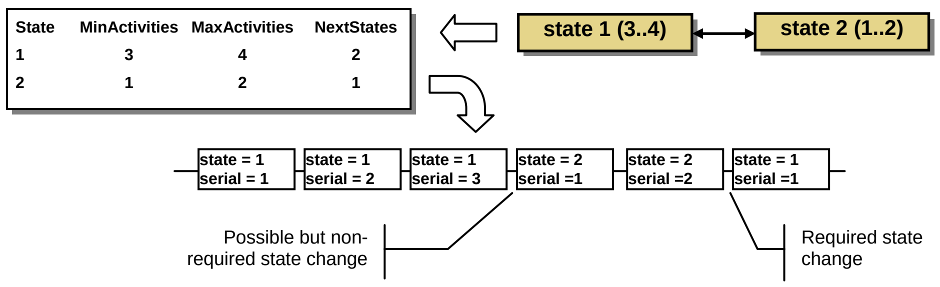


Fig. 5. Slots can move in time provided that the ordering of slots is preserved

The details about slot representation can be found in [3,6], we will extract here only the slot parameters necessary to model the state transitions. The state of activity that can be filled in the slot is described by a finite domain (FD) variable *state*. In particular, the domain of this variable contains identifications of possible states of activities that can be located in a given slot. The constraint connecting the state variables of two consecutive slots describes naturally the allowed transitions. For example, if the transition from state 1 to state 2 is allowed by the state transition graph then the pair (1,2) is allowed by the constraint between the state variables of any two consecutive slots. In complex transition schemes, we also need to restrict repetition of states in the consecutive slots. Therefore, we introduce a variable *serial* that indicates a relative position of the activity (in a given slot) in the longest continuous sequence of the activities of the same state (see Figure 6). The state and serial variables of the first slot can be set to describe the initial situation of the resource.



State	MinActivities	MaxActivities	NextStates
1	3	4	2
2	1	2	1

Fig. 6. Serial numbers in slots indicate the position of the activity (slot) in the sequence of slots of the same state.

5. CONSTRAINT MODELS

In the previous section, we sketched the basic idea of representing the state transition scheme using slots. We will now focus on detail description of constraints modelling the state transitions and local counters – so called *transition constraints*. Basically, the transition constraint connects state and serial variables of each two consecutive slots. Because the user defines the transitions between the states we must be ready to cover an arbitrary relation. In [4,5] we proposed filtering algorithms for general binary constraints, we call them tabular constraints, and these algorithms will be 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 activities and between the state and the maximal number of activities. In the following paragraphs, we will describe three different models of the transition constraint. All of these models use some form of the tabular constraint.

5.1. 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, that is, to follow the syntax of a particular underlying constraint solver.

First, we define the state transition constraint via the tabular constraint. Basically, it is a general binary constraint defined using the table NextStates. The index of the variable indicates the ordinal number of the slot. The constraint says that the next state is either identical to the current state or it is one of the states that can follow the given state according to NextStates table.

$$\mathtt{state}_{i+1} \in \{\mathtt{state}_i\} \cup \mathtt{NextStates}(\mathtt{state}_i) \qquad (1)$$

Now, we can define how the serial number changes in the next slot using two constraints. Simply, if the states in the slots are identical then the serial number is increased by one, otherwise the serial number is set to one (we are starting to count activities of another state). Because exactly one of the preconditions of the following implications holds, one of the conclusions must hold as well.

$$\mathtt{state}_i = \mathtt{state}_{i+1} \Rightarrow \mathtt{serial}_{i+1} = \mathtt{serial}_i+1 \qquad (2)$$
$$\mathtt{state}_i \neq \mathtt{state}_{i+1} \Rightarrow \mathtt{serial}_{i+1} = 1 \qquad (3)$$

Finally, we need to connect the serial number and the state variable in each slot to model the minimal/maximal number of activities per state. The serial number cannot be greater than the maximal number of activities of a given state. This can be modelled using a tabular constraint, where the table MaxActivities describes the relation. A similar table MinActivities describes the lower limit for the number of activities. If the serial number in the slot is smaller than the minimal number of activities then the state in the next slot must be identical to the state in the current slot.

$$\mathtt{serial}_i \in \{1,..,\mathtt{MaxActivities}(\mathtt{state}_i)\} \qquad (4)$$
$$\mathtt{serial}_i < \mathtt{MinActivities}(\mathtt{state}_i) \Rightarrow \mathtt{state}_i = \mathtt{state}_{i+1} \qquad (5)$$

Proposition 1: The basic logic model describes the restrictions imposed by the state transition graph.

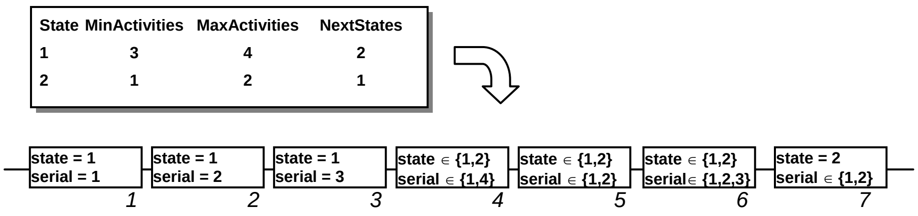
Proof: We will show that any complete instantiation of state and serial variables satisfying the constraints (1) - (5) describes a valid sequence of activities according to a given state transition graph. For simplicity reasons, we can assume that $\text{serial}_1 = 1$.

Let i,…, i+k be a sequence of slots of the same state that cannot be extended in any direction. Formally, $\forall j \in \{0,\dots,k\}$ $\text{state}_i = \text{state}_{i+j}$ and either i = 1 or $\text{state}_i \neq \text{state}_{i-1}$, and either i+k is the last slot or $\text{state}_{i+k} \neq \text{state}_{i+k+1}$. We will show now that $k+1 \leq \text{MaxActivities}(\text{state}_i)$ and if i+k is not the last slot then also $\text{MinActivities}(\text{state}_i) \leq k+1$. In other words, any (non-extendable) sequence of consecutive slots of the same state has the requested length. First, $\text{serial}_i = 1$ because either i = 1 (and $\text{serial}_1 = 1$ according to our assumption) or $\text{state}_i \neq \text{state}_{i-1}$ and $\text{serial}_i = 1$ according to (3). According to (2) we know that $\text{serial}_{i+1} = \text{serial}_i+1$ etc. so by induction we can show that $\forall j \in \{0,\dots,k\}$ $\text{serial}_{i+j} = j+1$. Specially, $\text{serial}_{i+k} = k+1$. By (4) we know that for any j $\text{serial}_j \leq \text{MaxActivities}(\text{state}_j)$, in particular $\text{serial}_{i+k} \leq \text{MaxActivities}(\text{state}_{i+k})$. Together, we get $k+1 \leq \text{MaxActivities}(\text{state}_i)$. Assume that i+k is not the last slot so according to our assumption $\text{state}_{i+k} \neq \text{state}_{i+k+1}$. Hence from (5) we can deduce that $\text{serial}_{i+k} \geq \text{MinActivities}(\text{state}_{i+k})$ so we get $k+1 \geq \text{MinActivities}(\text{state}_i)$.

Assume now that $state_j \neq state_{j+1}$, that is, the state is changed. According to (1) $state_{j+1} \in NextStates(state_j)$ so the state change follows some arc in the state transition graph. Together we showed that any sequence of slots satisfying the constraints of the basic logic model follows the restrictions of the state transition graph, namely state transitions and local counters.

To complete the proof, we need to show that any valid sequence of activities according to the state transition graph corresponds to some instantiation of state and serial variables in slots satisfying the constraints (1) - (5). Let the value of the state variable be set to the state identification and the value of the serial variable be set to the ordinal number of the activity in the sequence of activities of the same state. Then, it is easy to show that this instantiation satisfies all the constraints (1) - (5). □

The basic logic model is sound but it suffers from weak domain filtering. Figure 7 demonstrates a problem that is locally consistent (every constraint is consistent), but there is no complete instantiation of variables satisfying all the constraints. This is an undesirable feature because such inconsistencies must be detected by the search procedure which decreases overall time efficiency.



State	MinActivities	MaxActivities	NextStates
1	3	4	2
2	1	2	1

Fig. 7. A locally (arc) consistent sequence of slots that is not globally consistent - the seventh slot cannot be in state 2.

5.2. A simple tabular model

The weak domain filtering in the basic logic model is due to the separation of information about the state and the serial number into two variables. In Figure 7 there is an inconsistent serial number 3 in slot 6. This value is deduced from value 2 of the serial number in slot 5 for state 1. Locally, this is a correct decision, that is, all the constraints are locally consistent. The problem is that serial number 2 in slot 5 belongs to state 2 only and the maximal number of activities of state 2 is 2. To keep information about the connection between the serial numbers and the states we propose to encode the state and the serial numbers into a single number called a *compound serial number* (cserial) using the formulas:

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

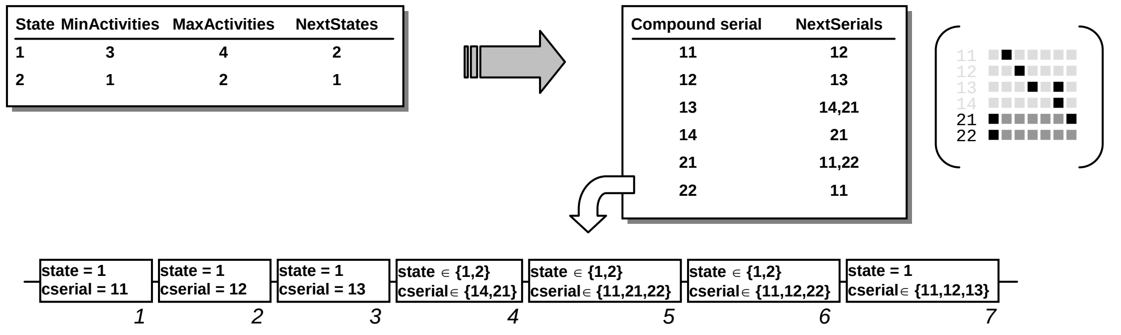
$$\text{cserial} = \text{separator} \,.\, \text{state} + \text{serial}$$

Separator is a constant that describes which part of the compound serial number (cserial) defines the state and which part defines the serial number (we use decimal separators). For example, if the separator is 10 then the compound serial number 13 encodes state 1 and serial number 3 (that is, the third activity of the first state). Note that the above encoding is a bijection from states and serial numbers to compound serial numbers. In particular, the state and the serial number can be decoded from the compound serial number in a unique way. Hence, to simplify reading we will also use cserial “decoded” as (state, serial).

The complete transition scheme can now be converted into a single transition table NextSerials describing the transitions in terms of compound serial numbers. This table covers both state transitions and MinActivities/MaxActivities restrictions.

$$\begin{aligned} NextSerials((s, i)) = &\{ (s, i+1) \mid i < MaxActivities(s) \} \\ &\cup \{ (r, 1) \mid r \in NextStates(s) \wedge i \geq MinActivities(s)\} \end{aligned}$$

Figure 8 shows an example of such a transition table. For example the compound serial number 13, that represents the third activity of state 1, can either go to 14, that is, to the fourth activity of state 1, or to 21, that is, to the first activity of state 2. However, if the compound serial number is 11 or 12 then we can continue with 12 or 13 respectively and the state is not changed. By keeping information about states and serial numbers together we prevent the problem depicted by Figure 7. In fact, this new model achieves global consistency as we will show later.



State	MinActivities	MaxActivities	NextStates
1	3	4	2
2	1	2	1

Compound serial	NextSerials
11	12
12	13
13	14,21
14	21
21	11,22
22	11

Fig. 8. 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.

$$\mathtt{cserial}_{i+1} \in \mathtt{NextSerials}(\mathtt{cserial}_i) \qquad (6)$$

Because the state is also encoded in the compound serial number, the state variable is not necessary in the transition constraint. However, states play an important role in other scheduling constraints so the state variable should be preserved. The state can be decoded from the compound serial number easily:

$$\mathtt{state}_i = \mathtt{integer}(\mathtt{cserial}_i\ /\ \mathtt{separator})$$

Proposition 2: The simple tabular model describes the restrictions imposed by the state transition graph.

Proof: Recall that we have one-to-one mapping between cserial and (state, serial) variables. So a complete instantiation of cserial variables can be mapped to a complete instantiation of state and serial variables and vice versa. Moreover, the constraint (6) over the cserial variables is satisfied if and only if the constraints (1) - (5) over the corresponding state and serial variables are satisfied. This follows directly from the definition of table NextSerials ((s, i) ∈ NextSerials((r, j)) iff all the constraints (1) - (5) over (r, j) and (s, i) are satisfied). Consequently, using Proposition 1 we can deduce that any complete instantiation of cserial variables in slots satisfying the constraints (6) corresponds to a valid sequence of activities according to the state transition graph and vice versa. □

In Proposition 2 we used a solution equivalence of the basic logic model and the simple tabular model. However, it does not imply that the models are equivalent concerning the filtering power of constraints. Notice that five constraints from the basic logic model are encoded in a single constraint in the simple tabular model so we can do some "global" reasoning over these five constraints. Proposition 3 shows that in fact the simple tabular model achieves global consistency, that is, any value remaining in the domain of cserial variables after making the problem arc consistent can be part of some solution.

Proposition 3: If the simple tabular model is made arc consistent then global consistency is achieved.

Proof: The constraint network of the simple tabular model, where nodes correspond to variables and edges to constraints, forms a tree (Figure 9). It is known that arc consistency in the tree structured constraint networks leads to global consistency [12]. □

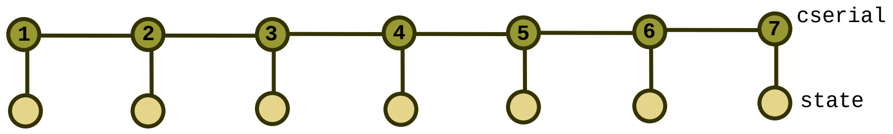


Fig. 9. The constraint network of the simple tabular model forms a tree.

5.3. A compound model

The simple tabular model provides a complete domain filtering for the transition constraint. It is also easy to implement 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 activities 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 [5]. Thus we decided to combine the representation using the compound serial numbers with the intentional model of the transition constraint. Intentional model means that instead of using a table describing the constraint, we propose a dedicated filtering algorithm realising the same constraint.

The new filtering algorithm uses the original transition scheme, that is, the tables NextStates, MinActivities, and MaxActivities. For efficiency reasons, the values in MinActivities and MaxActivities tables are expressed as compound values, that is, they include the state. For example, if separator = 10 then MinActivities(1) = 14 means that there must be at least 4 activities of state 1 in a sequence. The filtering algorithm computes the induced transition table on demand 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,FromCSerial,ToState,ToCSerial)
    ToCSerial ← relevant_serials(ToState,ToCSerial)
    NewFromState ← NewToState ← NewFromCSerial ← NewToCSerial ← {}
    for each State ∈ FromState do
     CSerial ← relevant_serials({State},FromCSerial)
     if nonempty CSerial then
        NextCSerial ← increase(CSerial,MaxActivities(State)) ∩ ToCSerial
        if nonempty NextSerial then
        // transition to identical state
          NewFromState ← NewFromState ∪ {State}
          NewToState ← NewToState ∪ {State}
          NewFromCSerial ← NewFromCSerial ∪ decrease(NextCSerial)
          NewToCSerial ← NewToCSerial ∪ NextCSerial
        end if
        NextCSerial ← start_serials(NextStates(State)) ∩ ToCSerial
        if nonempty NextCSerial & MinActivities(State) ≤ max(CSerial) then
        // transition to different states
          NewFromState ← NewFromState ∪ {State}
          NewToState ← NewToState ∪ relevant_states(NextCSerial)
          NewFromCSerial ← NewFromCSerial ∪
            ((MinActivities(State)..MaxActivities(State)) ∩ FromCSerial)
          NewToCSerial ← NewCToSerial ∪ NextCSerial
        end if
     end if
    end for
    return(NewFromState, NewFromCSerial, NewToState, NewToCSerial)
end procedure
```

The algorithm takes domains of state and cserial variables of two consecutive slots as its input and returns the new (pruned) domains of these variables containing only the consistent values, that is, the values satisfying the transition constraint. It uses the following procedures inside. Procedure `relevant_serials(States, CSerials)` selects from CSerials the compound serial numbers corresponding to any state in States, for example, `relevant_serials({1},{11,12,13,21,21})` returns {11,12,13} (for simplicity, we assume separator = 10 in examples). Symmetrically, procedure `relevant_states(CSerials)` returns the states corresponding to the compound serial numbers from CSerials, for example, `relevant_states({11,12,13,21,21})` returns {1,2}. Procedure `increase(CSerials, Max)` adds 1 to each compound serial number from CSerials respecting the maximal number Max (we assume that only compound serial numbers of a single state are present), for example, `increase({11,12,13},13)` returns {12,13} (14 is not allowed). Similarly, `decrease(CSerials)` decreases by 1 the compound serial numbers taking in account the minimal possible number, for example, `decrease({11,12,13})` returns {11,12} (10 is not allowed). Finally, procedure `start_serials(States)` generates the first compound serial number for each state in States, for example, `start_serials({1,2,3})` returns {11,21,31}.

Basically, the proposed filtering algorithm computes a part of table NextSerials on demand and according to the table it selects only the compatible values from domains of the variables. For each possible state in the first slot we check whether there is some compound serial number for this state. If not then the state is no more assumed, otherwise we start exploring the rows of table NextSerials for this state. First, we try to increase by one the compound serial numbers of the given state to obtain the compound serial numbers for the same state in the next slot. If any of these numbers appears among the compound serial numbers of the next slot then it is possible that the state is not changing and hence we add the state and the corresponding compound serial numbers into new domains. Second, we explore possible changes of the state by checking whether the minimum number of activities has been reached and there is a corresponding "starting" compound serial number in the next slot. Again, if this is possible, we update the new domains accordingly. Figure 10 shows the relation between the filtering algorithm, table NextSerials (expressed as a binary compatibility matrix) and the original state transition graph.

```
procedure TransitionConstraint(FromState,FromSerial,ToState,ToSerial)
init new domains (empty)
for each State in FromState do
        if self transition then
                add State to NewFromState, NewToState
                set NewSerial shift
        if transition to a different state then
                add State to NewFromState
                add NextStates(State) to NewToState
                set NewSerial transition
end for
return new domains
end SP
```

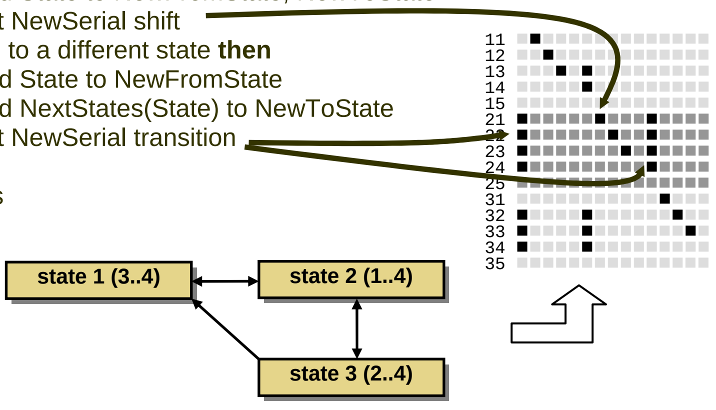


Fig. 10. Filtering algorithm TransitionConstraint explores the table NextSerials by rows (possible transitions from state 2 are highlighted).

It may seem that 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 simple tabular model is hidden in the tabular constraint. Because, the new filtering algorithm mimics behaviour of the filtering algorithm for tabular constraints described in [4], there is no significant decrease of efficiency. Additional work done during filtering is balanced by using smaller and more compact tables compared to the simple tabular model.

6. GLOBAL COUNTERS

The motivation example from the beginning of the paper requires global counters to model the proper location of the cleaning activity. Because of using constraint satisfaction technology, the concept of global counters can easily extend the state transition graph.

Recall that a global counter counts activities of some states and when a given limit is reached, the counter forces a specific transition. This idea can be formally described in the following way. For each global counter we decompose the set of states into three disjoint sets: counter states, inert states, and reset states. Each state belongs to one of these sets. Moreover, for each global counter we a have natural number describing its limit. The semantic of global counter is as follows. At the beginning, the global counter has a value zero (or some other predefined number). The global counter is increased by one each time any activity of the counter state is processed. The global counter is not changed if any activity of the inert state is processed. Finally, the global counter is set to zero each time an activity of the reset state is processed. The role of the counter limit is following. As soon as the counter reaches the limit, the next activity in the sequence must be some activity of the reset state. In any case, the counter is not allowed to exceed the counter limit. For example, if we want to force cleaning after three load-heat-unload production cycles in Figure 1, we can use the global counter with the limit 3 that has unloading as the only counter state, cooling as the only reset state, and all other states are inert states.

Global counters can be added to all presented models of state transitions. It is enough to include a new variable $counter_i$ specifying the value of the global counter in the slot i (to be more precise, this variable indicates the value of the counter right after processing the activity allocated to slot i). The constraints modelling the global counter can be defined in a similar way like in the basic logic model. The following three constraints describe the update of the counter (for simplicity reasons, we define $counter_0 = 0$):

$$\mathtt{state}_i \in \mathtt{CounterStates} \Rightarrow \mathtt{counter}_i = \mathtt{counter}_{i-1} + 1$$
$$\mathtt{state}_i \in \mathtt{ResetStates} \Rightarrow \mathtt{counter}_i = 0$$
$$\mathtt{state}_i \in \mathtt{InertStates} \Rightarrow \mathtt{counter}_i = \mathtt{counter}_{i-1}$$

The forced transition after the counter reaches its limit is described by the following constraint:

$$\mathtt{counter}_{i-1} \geq \mathtt{Limit} \Rightarrow \mathtt{state}_i \ \mathtt{in} \ \mathtt{ResetStates}$$

Let us conclude this section by two notes. First, it is possible to define as many global counters as the user requires. For each global counter we will use its specific $counter_i$ variable and the set of constraints of the above form. Second, by adding the global counters into the simple tabular model we lose global consistency because the constraint network is changed. Nevertheless, this happens almost always when additional constraints are present in the system. For example, if there are more resources with some relations between them (like in the motivation example) then consistency techniques are not enough to solve the problem and search is necessary.

7. DISCUSSION AND CONCLUSIONS

In the paper, we proposed and discussed 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 [11] and tested in the real-life scheduling system Visopt ShopFloor [6]. The idea of presented constraint models has been first published in [7].

Rather than providing a detail empirical evaluation we describe how the idea of the transition constraints evolved. First, we implemented 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. Actually, we were not able to solve some real-life problems because of large memory consumption. Therefore, we returned to the basic logic model of the transition constraint that does not filter as good as the simple tabular model (see Figure 7) but its memory consumption is much smaller. Unfortunately, missing propagation led to "infinite" solving times of some real-life problems with restrictive transition scheme because the system was "lost in backtracking". Therefore we proposed a dedicated filtering algorithm for the transition constraint that mixes the advantages of both logic and tabular models, namely good domain filtering and reasonable memory consumption. This compound model is used in the current version of Visopt ShopFloor system. In the paper, we also proposed a variant of the basic logic model to describe global counters over the sequence of activities. Such counters can be used to model maintenance and cleaning activities that typically appear after a specified number of production cycles. Global counters are also implemented within the Visopt ShopFloor system.

We are not aware about another approach that can cover generality of the state transition scheme with local and global counters studied in this paper and hence no comparison to other approaches can be given. We used the concept of constraint satisfaction rather than writing a particular solving algorithm because this concept can be easily extended and combined with other constraints (we demonstrated this extendibility by adding the global counters). This is particularly important for real-life applications where there are many "side" constraints that must be covered and that a dedicated solving algorithm can hardly accommodate. Note finally, that the studied problem is strongly motivated by requirements of real-life applications and it is not an academic only problem. We demonstrated it by using a simple motivation example that includes several features required by real-life complex scheduling problems.

8. ACKNOWLEDGEMENTS

The research is supported by the Czech Science Foundation under the contract no. 201/07/0205.

REFERENCES

[1] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, Kluwer Academic Publisher, 2001.

[2] R. Barták, Conceptual Models for Combined Planning and Scheduling. *Electronic Notes in Discrete Mathematic*, Volume 4, Elsevier, 1999.

[3] R. Barták, Dynamic Constraint Models for Planning and Scheduling Problems. *New Trends in Constraints.* LNAI 1865, 237-255. Springer Verlag, 2000.

[4] R. Barták, A General Relation Constraint: An Implementation. *Proceedings of CP2000 Post-Workshop on Techniques for Implementing Constraint Programming Systems*, 30-40, 2000.

[5] R. Barták, Filtering Algorithms for Tabular Constraints. *Proceedings of CP2001 Workshop CICLOPS*, 168-182, 2001.

[6] R. Barták, Visopt ShopFloor: On the edge of planning and scheduling. *Principles and Practice of Constraint Programming - CP2002*. LNCS 2470, 587-602. Springer Verlag, 2002.

[7] R. Barták, Modelling Resource Transitions in Constraint-Based Scheduling. *Proceedings of SOFSEM 2002: Theory and Practice of Informatics*. LNCS 2540, 186-194. Springer Verlag, 2002.

[8] J. Ch. Beck and M.S. Fox, Scheduling Alternative Activities. *Proceedings of AAAI-99*, 680-687. AAAI Press, 1999.

[9] P. Brucker, *Scheduling Algorithms*. Springer Verlag, 2001.

[10] V. Brusoni, L. Console, E. Lamma, P. Mello, M. Milano, P. Terenziani, Resource-based vs. Task-based Approaches for Scheduling Problems. *Proceedings of the 9th ISMIS96*, LNCS Series, Springer Verlag, 1996.

[11] M. Carlsson, G. Ottosson, B. Carlson, An Open-Ended Finite Domain Constraint Solver. *Proceedings Programming Languages: Implementations, Logics, and Programs*. 1997.

[12] R. Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.