Integrating Planning into Production Scheduling: A Formal View

Roman Barták

Charles University in Prague, Faculty of Mathematics and Physics
Institute for Theoretical Computer Science
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
bartak@kti.mff.cuni.cz

Abstract

The paper describes an approach for integrating planning capabilities into production scheduling. We give an abstract view of production scheduling that motivates the integration and we compare the existing solving technologies in respect to their power for planning and scheduling integration. A new formal model of production scheduling is proposed and its realization in the Visopt ShopFloor system is described.

Introduction

Traditional planning deals with the problem of finding activities to satisfy a given goal. Traditional scheduling solves the problem of allocating known activities to limited resources and to limited time. In many real-life problems both tasks should be accomplished together so integrating planning and scheduling is a hot research topic especially in the planning community. This integration usually means adding time and resource restrictions to the planning problem. Because solving traditional planning problems is hard, adding time and resource constraints may make the problem even harder. Therefore, some researchers propose to keep planning and scheduling separated (Srivastava and Kambhampati 1999). In particular, the planning problem is solved first, which generates a set of activities, and the scheduling problem is solved next, which allocates the activities to resources and time. This is useful, if the planning space is large – if it is hard just to find a valid plan. However, in many real problems it is pretty easy to find a valid plan but it is more complicated to find a good plan in respect to available resources and time. Moreover, sometimes a planning decision – an introduction of an activity – is tightly coupled with a scheduling decision – an allocation of the activities to time and resources. For example, assume a set-up activity whose existence depends on the neighboring production activities. Introduction of such an activity depends directly on the allocation of the production activities. In such a case, the integration of planning and scheduling is inevitable.

In (Barták 1999b) we argued for a more tighten integration of planning and scheduling where the time and resource constraints play an important role in guiding the planner. The basic idea is to post the time and resource constraints as soon as the planner introduces some activity. These constraints then help the planner to decide among the alternative activities in a forward or backward chaining style of planning.

In this paper we describe a new formal model of a tightly integrated planning and scheduling problem. In particular, we deal with the production scheduling problems that require some planning capabilities. Briefly speaking, the problem requires the activities to be introduced during the scheduling process. We formulate the problem in a constraint satisfaction framework that, as we believe, is appropriate to solve such type of problems. We also give some details how to deal with the dynamic formulation of the problem in the constraint satisfaction framework. In particular, we describe how variables and constraints can be introduced dynamically during variable labeling. The proposed techniques have been implemented and tested in the Visopt ShopFloor scheduling system.

The paper is organized as follows. In the next section we will introduce an abstract production scheduling problem and we will explain why integration of planning into scheduling is necessary to solve such a problem. Then, we will overview available technologies for solving planning and scheduling problems and we will highlight their advantages and drawbacks. After that, we will propose a formal constraint model of the problem and we will present two possible ways how to solve this model. Before conclusion we will discuss some difficulties and challenges of the implementation of the proposed formalism.

Motivation

The goal of production scheduling is to generate a plan (a schedule) of production for a specified time period. This plan should satisfy the demands and it should be as profitable as possible. The *demands* describe items that should be produced (including their quantity) as well as time when the item must be ready. Some demands have hard deadlines so the demanded quantity must be ready at a given time. Other demands model a forecast of future demands so it is possible to postpone them if there is not enough resource capacity. The system decides which demands will be satisfied by using information about costs, penalties, and load of resources.

Items are produced on resources with a limited capacity – we call them *main resources*. The production in a

resource is described as a sequence of non-overlapping activities. The sequencing of activities may be further restricted by a *transition scheme* that describes allowed transitions between the activities. The transitions also specify transition times between the activities and they may also specify positioning of some non-production activities like set-up or cleaning activities. Figure 1 shows a transition scheme restricting the sequencing of activities.

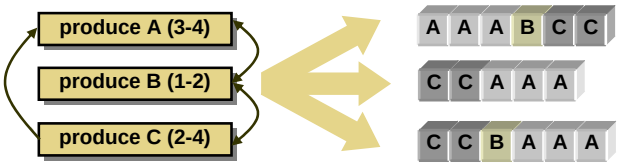

Figure 1. Activities are connected in a transition scheme (left) that restricts the possible transitions as well as a minimal and maximal number of identical activities in a continuous subsequence. This scheme restricts the feasible sequences of activities (right).

The activities produce and consume items. If an activity consumes some item then there must exist another activity that produces this item and vice versa. Note that it is possible to have several consumers and several producers of the item so there is a many-to-many relation between the activities (Figure 2). Note also that the demands can be seen as the final consumers of the items. We call the above producer-consumer relation a *resource dependency* because it describes the dependencies between the resources. The resource dependency specifies which activities can be connected and what is the transport time between the activities. In fact, it is a generalization of the precedence constraints used in the traditional scheduling.

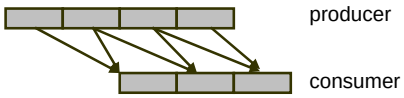

Figure 2. Items are flowing (arrows) between the activities (rectangles). This item flow defines the precedence constraints.

So far, it might seem like a typical scheduling problem. However, notice the following significant difference from the traditional scheduling problems. When specifying the activities, the user does not describe the actual activities to be scheduled, that is the activities satisfying the current demands. The user specifies the activities that can be used in the schedule – this is called a *domain model* in planning. It is the responsibility of the scheduling system to select the appropriate activities which satisfy the demands, form valid resource schedules according to the transition scheme, and form valid item flows according to the resource dependencies. Thus the scheduling system must solve a type of integrated planning and scheduling problem.

For simplicity reasons we use here only the resources processing a single activity at time. However one may assume an extension of the problem where other types of resources appear. An example problem of the above type is described in (Barták 2003b).

Available Technology

One of the strongest trends in planning in the recent years is extending the traditional planning framework by scheduling constraints. This is reflected for example in the recent planning competition (Long and Fox 2002) where the tasks to be solved included several scheduling features. The main approach used in the planning community to model scheduling features is based on attaching some numerical attributes to activities. These numerical attributes can model activity duration as well as resource constraints. While time is modeled explicitly so the system "knows" about the meaning of time attributes, resources are still modeled ad-hoc. It means that the solving algorithm is not aware about the resource nature of the numerical attribute so some general solving approach should be applied. This is a big drawback as the solver cannot exploit fully the scheduling technology. Moreover, the current planners that are able to handle some scheduling constraints are still primarily planners with the capabilities to handle the planning task primarily. As we sketched in the previous section, in some problems the planning task is not very difficult. The problem is made difficult by the time restrictions and resource sharing. Thus, the scheduling technology should play a more important role there.

Surprisingly, the traditional scheduling research seems to be almost untouched by the recent trends in planning and artificial problems like job-shop or open-shop are still of major interest in the scheduling community. There exist generalizations of the academic scheduling problems like Resource Constrained Project Scheduling Problem which are more suitable for the real-life applications. However, these problems are still not going beyond the traditional scheduling task which is allocation of known activities to known resources. The dynamic features are introduced by on-line scheduling but note that this is different from the problem sketched in the previous section. In the on-line scheduling, the activities are coming from the external environment while our problem expects the activities to be planned by the solver. As far as we know, there is no formal scheduling model that covers planning capabilities and the majority of the research papers on such models are written by the practitioners which are exposed to real-life problems. Thus, the papers describe particular applications rather than general frameworks.

If we look into technologies used to solve both planning and scheduling problems we can find out that the common technology applied to both areas is constraint satisfaction. In the traditional formulation of the constraint satisfaction

problem (CSP), the variables, their domains, and the constraints must be specified before the problem is being solved. This fits perfectly the scheduling task so it is not surprising that constraint satisfaction is extensively used there (Baptiste, Le Pape and Nuijten 2001). However, the static formulation of constraint satisfaction complicates its usage in the planning problems which are dynamic in their nature. Thus constraints are used there typically in the plan extraction phase where the constraint satisfaction problem can be formulated statically (Do and Kambhampati 2000). We believe that constraint satisfaction is a good bridge technology especially for solving scheduling problems enhanced by some planning features. Nevertheless, if we choose constraint satisfaction as the integration technology then a more dynamic approach to CSP is necessary.

Constraint satisfaction is a technology that is already used to solve real-life problem so, not surprisingly, there exist extensions of its static formulation to cover dynamic problems. Dechter and Dechter (1998) proposed a concept of Dynamic CSP which is a sequence of traditional constraint satisfaction problems, where every problem is a result of changes in the preceding problem. This concept is appropriate to handle on-line scheduling problems where the problem modifications are coming from the external world but it does not bring many advantages for solving the planning problems.

The original formulation of Dynamic CSP by Mittal and Falkenhainer (1990) motivated by the configuration problems seems more appropriate for the planning problems. Their idea is to use an activation constraint that can activate some variables so these variables will participate in the solution. Variable activation can be seen as selecting some activity in the plan. Nevertheless, this approach is still static in the sense that all the variables are introduced in advance but some of them remain inactive. Thus, this approach is not fully appropriate for general planning problems where the number of such dummy variables can be huge. However, this approach could be useful when the number of dummy variables is low. For example, dummy activities are used in scheduling when there are some alternative schedules/plans that must be selected during the scheduling process (Pegman 1998).

Nareyek (1999) proposed a concept of Structural Constraint Satisfaction to solve the highly dynamic problems typical for planning. His approach is based on the idea of extending the abstract parts of the constraint network into less abstract parts containing new variables and constraints. This approach is similar to hierarchical task networks and hence it is very useful to solve planning problems. However, the Structural CSP should be implemented from scratch so it is very hard to extend the existing constraint solvers into Structural CSP.

Surprisingly relatively small attention is given to using constraint logic programming (CLP) for solving planning problems. The progress in constraint satisfaction push the users to apply CLP in the same way as CSP, that is define the constraint satisfaction problem first and then use search (labeling) for assigning values to the variables. However, recall that CLP was originally proposed as an extension of logic programming where the constraints substituted unification (Gallaire 1985). Constraints are used there to prune the search space but search is done within the logic programming framework not in a special labeling procedure. Thus, it is natural to have a different set of variables and constraints in different search branches. We believe that constraint logic programming is one of the most appropriate frameworks for modeling and solving integrated planning and scheduling problems. It allows a natural dynamic introduction of variables and constraints as search proceeds and there already exist off-shelf CLP systems so one can focus on the problem solving rather than on the implementation of the underlying technology.

A Formal Model

In this section we will give a more precise description of the addressed problem and we will show how it can be encoded using constraints. However, note that the actual implementation does not necessarily follow the constraints presented here. We present the constraints to show what restrictions must be satisfied rather than to introduce a particular implementation of the constraint model.

Resources

A resource is specified as a finite set of activities that can be processed by the resource. Let us denote by $Activities(r)$ the set of activities that can be processed by the resource r. We expect that the sets of activities are disjoint for different resources, formally:

$$r \neq s \Rightarrow Activities(r) \cap Activities(s) = \emptyset.$$

It may seem that the above feature forbids modeling of alternative resources per activity. Such activity is modeled as a set of "identical" activities in alternative resources and these activities are connected to other activities via alternative item flows (see next section). Actually, this is a standard way of modeling alternative resources per activity.

For each activity, the user specifies its duration as a positive integer number. Let us denote by D_i the duration of the activity i. The activity occupies the resource from its start time till its completion time and no other activity can be processed at that time. Such resources are often called unary or disjunctive resources.

The transition scheme is described by a table of transition times between each pair of activities. Let us denote by $T_{i,j}$ the transition time when going from the activity i to the activity j. The transition time is an interval starting with a non-negative integer, for example [3,sup] denotes a minimal transition time 3 and no maximal transition time. If the transition is not allowed then the transition time equals *sup*. Moreover, for each activity i a minimal Min_i and maximal Max_i number of activities that can be processed in a continuous sequence are specified. If $Max_i > 1$ then $T_{i,i}$ specifies the transition time between the activities in a sequence of identical activities. The

possibility to restrict the number of identical activities in a continuous sequence is useful for example for modeling setup, changeover, or transition activities that are processed exactly once between two production activities – Max_i=1. Notice also that these activities are handled like other activities. In particular, they may consume and produce items which is useful to model transition activities producing some low-quality items (for example items with a color that is between two pure colors) or to model changeover activities connected to activities in other resources (for example mould change in the injection machine requires a crane). Such feature is often omitted in the traditional approaches to scheduling where such activity is modeled as a transition time between two activities (this is possible in our framework as well).

As we already mentioned, the task is to generate a schedule for a fix time period. Assume that this period is described by an interval [0,*MaxT*]. A *valid schedule* for the resource *r* is a sequence of activities that can be processed by the resource with start times assigned to integers. Assume that the length of this sequence is *n*, act(*i*) denotes the *i*-th activity in the sequence, and start(*i*) denotes its start time. Then the following *resource constraints* must be satisfied by the valid schedule for the resource *r*:

1. $\forall i$=1…*n*: act(*i*)$\in$*Activities*(*r*)
2. $\forall i$=1…*n*: 0 ≤ start(*i*) ≤ *MaxT*
3. *MaxT* ≤ start(*n*)+$D_{act(n)}$ or
   $\exists$b$\in$*Activities*(*r*) *MaxT* ≤ start(*n*)+$D_{act(n)}$+max($T_{act(n),\,b}$)
4. $\forall i$=1…*n*-1:
   start(*i*)+$D_{act(i)}$ +min($T_{act(i),\,act(i+1)}$) ≤ start(*i*+1)
   start(*i*+1) ≤ start(*i*)+$D_{act(i)}$+max($T_{act(i),\,act(i+1)}$)
5. If *l* is a maximal length of any continuous subsequence of activities *i* then:
   l ≤ Max_i.
   If the subsequence of activities *i* is not the last one in the resource schedule then also:
   Min_i ≤ *l*.

The first constraint ensures that only activities that can be processed by a given resource are included in the resource schedule. The second constraint ensures that the activities start within the scheduled period. It is not possible to schedule the activities before time zero (in past) and we are not interested in the activities starting behind the schedule horizon (after *MaxT*). The third constraint ensures that a complete schedule is produced. It means that either the last activity completes after the end of the scheduled period or there could be another activity after the last activity that can start after the end of the scheduled period. This ensures continuous production if the transition times have a tighten upper bound. The fourth and fifth constraints ensure that the sequence of activities satisfies the transition scheme.

Dependencies

As we already mentioned, the activities may consume and produce items. If an activity produces some item then there must be another activity that consumes the item and vice-versa. Thus the item flow naturally models dependencies between the resources. We denote by $InQ_{Item,a}$ a quantity of *Item* consumed by the activity *a* and by $OutQ_{Item,a}$ a quantity of *Item* produced by the activity *a*. Quantity is a nonnegative integer. If quantity is zero then the item is not consumed or produced by the activity.

Assume that the activity *a* produces *Item* ($OutQ_{Item,a}$>0) and the activity *b* consumes the *Item* ($InQ_{Item,a}$>0). We specify the time necessary for moving the *Item* from *a* to *b* as $Delay_{Item,a,b}$. More precisely, delay is a difference between the start time of the consuming activity (*b*) and the completion time of the producing activity (*a*). Delay can be an interval, an integer number, or a value *sup* indicating that the *Item* cannot be moved between the activities. Note that even if one activity produces the item and another activity consumes the same item, it is still possible to forbid the transport between the activities if there is no transport line between the resources. This is useful to model real connections in factories.

We denote by q(*Item*,*r*,*i*,*s*,*j*) the quantity of *Item* moved from the *i*-th activity of the resource *r* to the *j*-th activity of the resource *s*. We call the set of such moved quantities *item flows* in the schedule. In the following we assume that act(*r*,*i*) denotes the *i*-th activity of the resource *r* and start(*r*,*i*) denotes its start time. These are the same variables as introduced in the previous section; we just attached an identification of the resource to them. The item flows are valid if the following *dependency constraints* hold:

1. q(*Item*,*r*,*i*,*s*,j)>0 ⇒
   start(*r*,*i*)+$D_{act(r,i)}$+min($Delay_{Item,act(r,i),\,act(s,j)}$) ≤ start(*s*,*j*)
   &
   start(*s*,*j*) ≤ start(*r*,*i*)+$D_{act(r,i)}$+max($Delay_{Item,act(r,i),\,act(s,j)}$)
2. $\forall s,j,Item$: $InQ_{Item,act(s,j)} = \Sigma_{r,i}$ q(*Item*,*r*,*i*,*s*,*j*)
3. $\forall r,i,Item$: $\Sigma_{s,j}$ q(*Item*,*r*,*i*,*s*,*j*) ≤ $OutQ_{Item,act(r,i)}$
   if there is no activity *b* such that
   MaxT ≤ start(*r*,*i*)+$D_{act(r,i)}$+max($Delay_{Item,act(r,i),\,b}$)
   then $\Sigma_{s,j}$ q(*Item*,*r*,*i*,*s*,*j*) = $OutQ_{Item,act(r,i)}$.

The first constraint ensures that if there is a non-empty item flow between two scheduled activities than the time distance between these activities is correct according to the specified delay. This is actually a conditional temporal constraint between two activities. The second constraint ensures that the consumed quantity of some item is produced somewhere. The third constraint ensures the same condition on the produced quantity of the item, so this quantity should be consumed somewhere. However, because we generate a schedule for a fix time period, it is not necessary to find consumers of all produced items provided that the consumers may exist in future (after *MaxT*). The third constraint says that if all the consuming activities must be within the schedule period then the produced quantity of *Item* must be consumed completely.

Dependencies describe real item flows in the plant. Decision about a non-empty item flow between activities corresponds to the decision whether there is a temporal

constraint between these activities. The actual temporal constraint is posted within the dependency constraint (1) while the dependency constraints (2) and (3) ensure that necessary temporal constraints are introduced. If the temporal part of the dependency constraint (1) is violated then the item flow is set to zero (constraint propagation). This realizes the idea of active decision postponement.

Note that temporal constraints that are not directly related to an item flow, like resource synchronization, can still be modeled. For example, assume that two activities must run in parallel on two different resources. This can be modeled by introducing a new artificial item produced by one of these activities and consumed by another activity. The delay for moving the item between the activities equals to the negative duration of the activities.

Demands

In the above model we specified only the activities as the objects producing and consuming items. However, the final consumers of the item are demands describing the orders from customers. So the demands can be seen as a special type of consuming-only activities. Let us denote by *Demands* the set of all demands. We expect that the set of demands is disjoint with the set of activities processed by the resources. Each demand $i \in Demands$ is specified by its delivery time DT_i when the demand should be satisfied and by a maximal allowed delay $Delay_i$. The delivery time is an integer within the interval [0,*MaxT*] because otherwise the demand is out of scope of the scheduled period. The delay is a non-negative integer and it specifies how much the delivery can be postponed. The item in the demand is specified in an expected way – using $InQ_{Item,i}$ for demand i. We denote by start(i) the time when the demand i is satisfied. This time must satisfy the following constraint:

$$\forall i \in Demands: DT_i \leq \text{start}(i) \leq DT_i + Delay_i.$$

The demands participate in the dependencies like other activities so they must satisfy the dependency constraint (1). If the demand i is scheduled within the scheduled period – $\text{start}(i) \leq MaxT$ – then the dependency constraint (2) must be satisfied as well. It means that the requested quantity must be available for delivery. Notice that some demands can be postponed after the schedule end, namely the demands i such that $MaxT < DT_i + Delay_i$. For such demands, it is not necessary to produce the requested quantity because these demands can be satisfied in future.

Features and Extensions

The above formal description of resources, dependencies, and demands fully specifies the problem to be solved. The main difference of the proposed formalism from traditional scheduling problems is that the set of activities in the schedule is not known in advance. Notice that the user just specifies the possible activities and their interaction via a transition scheme and dependencies. The set of demands is known in advance and it initiates the production. However, neither the number of activities nor their actual composition is known in advance. The actual activities are decided during the scheduling process. Thus, we are solving a production scheduling problem integrated with planning. Note also that the introduction of the planning capabilities into the scheduling process makes the model significantly more general because the model covers features that the traditional scheduling cannot cover. In particular, the actual activities to satisfy a demand are selected during scheduling so it is possible to model alternative production sequences (via alternative item flows). The traditional scheduling requires the production sequence to be selected before the scheduling starts which could make the schedule less efficient. The proposed model also allows sharing of activities between several demands which influences the choice of the production sequence and makes the schedule more compact. Moreover, it is possible to introduce activities for processing of the by-products produced by some activities. The activities consuming by-products may satisfy the demands more effectively than producing the demanded item from scratch. Thus, recycling is covered by the proposed model while the traditional scheduling approaches cannot model recycling fully. Last but not least, it is possible to schedule activities which are not directly related to the existing demands. For example, it is possible to introduce changeover activities that consume or produce some items. Basically, the transition scheme is responsible for introduction of such activities.

The presented model is intentionally simplified to show primarily the main features of the proposed formalism for integrating planning into production scheduling without overwhelming the reader. For example, it is possible to attach time windows to the activities or to specify the first activity in the resource schedule describing the initial state of the resource. It is also possible to use more general counters in the transition scheme. These counters may force some specific activity, for example cleaning, after processing a given number of counted activities. Note that introduction of counters in the above model of resources is straightforward thanks to the transition scheme. However, the counters cannot be modeled in the traditional task-centric view of the scheduling problems (Brusoni et al 1996). Actually, we are not aware about any scheduling system that can handle the counters or at least the above transition scheme. For details on counters see (Barták, 2002b). Also, for simplicity reasons we did not include any optimization in the above framework. For example, it is possible to attach a cost to each activity and a penalty for delaying satisfaction of the demand. Then the task is to find a schedule minimizing the total cost. Note that using the optimization criteria may force the system to produce items for demands even if the demands may be postponed. If there are no penalties for delaying the deliveries, the system may tend to postpone them because it simplifies the problem – fewer activities are necessary. For details on the cost model see (Barták, 2002a).

Realization

We proposed the above framework using a terminology of constraint satisfaction. It means that we define the decision variables, namely act(i), start(i), and q(*Item*,r,i,s,j), and the constrains restricting the values of these variables. However, notice that the variables are not known in advance because the number of activities is unknown in advance. Thus the nature of the problem integrates planning – introduction of activities – with scheduling – allocation of activities to time and space.

A Static Solver

The difficulty of the unknown set of decision variables can be resolved by using a standard technique of dummy activities that is applied in a less extent for example by Pegman (1998). The idea is as follows. For each resource we can estimate the maximal number n of activities by dividing *MaxT* by the smallest activity duration (it is possible to compute a more precise estimate by assuming the transition scheme). This number (n) defines the length of the resource schedule. Because the resource schedule will not be probably fully filled by the activities we should introduce a dummy activity that fills the empty end of the resource schedule. The duration of the dummy activity equals to the duration of the shortest activity (it is possible to use any duration, for example one) and the dummy activity produces and consumes no items. The dummy activity can be processed at any time that is even after the schedule period (*MaxT*). Thus, there is no upper bound defined for the start time of the dummy activities – the resource constraint (2) is modified to relax the upper bound of the dummy activities in the following way:

$$\forall i=1\ldots n:$$
$$0 \le \text{start}(i) \;\&$$
$$\text{act}(i) \neq dummy \Rightarrow \text{start}(i) \le MaxT$$

Moreover, a transition with a zero transition time is allowed from any activity to the dummy activity but there is no transition from the dummy activity to another activity – the dummy activity represents the dead end in the sequence of the activities. It is allowed to have an arbitrary number of dummy activities in a sequence (Max_{dummy}=sup) and the transition time between the dummy activities is zero ($T_{dummy,dummy}$=0). In a valid resource schedule, the dummy activities are collected at the end of the schedule and they do not bring any ambiguity into scheduling. We mean that the start times of the dummy activities are fully specified by the start time of the last "real" activity in the schedule.

The above method is very close to the timetabling approach where the slots are defined in advance and these slots are filled by activities during scheduling. We can use the metaphor of slots in our framework as well. The only difference is that the slots in our approach are not fixed in time so in addition to the activity variable there is a time variable for each slot. Nevertheless, the order of slots is fixed thanks to the resource constraint (4) that requires the slots to form a non-overlapping sequence in time.

For discussion on the timetabling approach to scheduling see (Barták 1999a). The advantage of the timetabling approach is that we get a standard constraint satisfaction problem where all the variables and constraints are known. Thus it is possible to use any constraint satisfaction technique to find a solution of the problem. On the other side, this approach is impractical for large problems because it requires a huge number of variables and constraints. In particular, a lot of variables are necessary to model the dependencies between the slots. We present this approach mainly to demonstrate the constraint satisfaction nature of the formal framework.

Note finally that instead of the dummy activities it is possible to deactivate the variables in the non-used slots at the end of the schedule similarly to the technique proposed by Mittal and Falkenhainer (1990). However, it does not solve the problem with memory consumption because all the slots (variables) must be introduced and the deactivated variables will not be assigned during variable labeling.

A Dynamic Solver

Because the static view discussed in the previous section is practically unusable, we have developed a dynamic solver as part of the Visopt ShopFloor system for solving the production scheduling problems (Barták, 2002a). The basic idea of the dynamic solver is quite simple – the variables and constraints are introduced on demand to minimize the memory consumption and unnecessary computation. We will discuss now what variables and constraints should be introduced and when they should be introduced.

If we summarize the input to the system then we know the set of demands and we know the description of the activities and their interactions (a transition scheme and dependencies). Thus, before we start solving the problem we can introduce the variables specifying the actual delivery time for the demands – the variables start(*demands*,i). Also, for each resource r we can introduce the first slot – the variables start(r,1) and act(r,1). We expect that the demands should be satisfied in the schedule so the variables q(*Item*,r,*1*,*demands*,j) are introduced together with the dependency constraint (1). For simplicity reasons, we call these variables q variables. The dependency constraint (2) is also introduced for the demands but note that we did not introduce yet the q variables going to slots other than the first slot. Thus the dependency constraint (2) is open in the sense that other q variables can be added to the constraint later. This is realized by the following mechanism. If the demand is not fully covered by the existing q variables (their sum is smaller than *InQ* for a given demand) then additional q variables connecting the demand to other slots are introduced. However, these slots are not present in the system so they should be added. In particular, if there is a variable q(*Item*,r,i,*demands*,j) then at least i slots of the

resource *r* should be present in the system. If this is not true then the necessary slots are introduced to the system.

The mechanism of introducing *q* variables can be generalized to the slots as well. If a slot *i* of the resource *r* is filled by an activity, which corresponds to assigning a value to the variable act(*r,i*), then we know the consumed and produced items in the slot. Thus, we can introduce the variables q(*Item,r,i,s,j*) and q(*Item,s,j,r,i*) that are not yet present in the system. These variables connect the slot with slots in other resource which may force introduction of new slots and so on. This mechanism significantly decreases the number of *q* variables in the system because only the *q* variables for relevant items are introduced. Again, together with the *q* variables the corresponding dependency constraints are posted.

When deciding about the value of the variable act(*r,i*) the existing variables q(*Item,r,i,s,j*) and q(*Item,s,j,r,i*) are assumed. It means that we are trying to fill the slot by an activity which satisfies a request from demands or other activities. In particular, if there is a request to produce some item then the activity producing the item is preferred. Usually there are several such requests so one of them is selected (a choice point in the labeling procedure is introduced) and the other incompatible requests are made zero. When the variable q(*Item,r,i,s,j*) is made zero by filling the *i*-th slot of the resource *r* by some activity incompatible with the request then the variable q(*Item,r,i+1,s,j*) is introduced . This corresponds to shifting the request to the next free slot as Figure 3 shows. Note that the incompatible *q* variables are made zero using the dependency constraints (2) and (3) that are posted when the activity in a given slot is known.

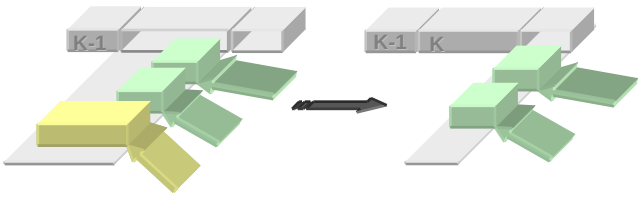

Figure 3. When the dependency is selected for the slot, then the incompatible dependencies are moved to next free slot.

So far we described the mechanism of introducing new slots when there is some request represented by the *q* variables. However, the slot may be also introduced due to a tighten transition scheme which requires activities even if there are no requests to produce or consume an item. We use the following mechanism to introduce a new slot according to the transition scheme. Assume that currently the last slot in the resource *r* has an index *i*. If $\max(\text{start}(i)+D_{\text{act}(i)})<MaxT$ then we know that the activity in the slot *i* completes before the end of the schedule period so there is a space for at least one more activity. In such a case we introduce the (*i*+1)-th slot to the system together with the resource constraints (3), (4), and (5) connecting *i*-th and (*i*+1)-the slots. Note that the above process can be realized even if the activity in the *i*-th slot is not known yet.

In the above paragraphs we sketched the basic principles of dynamic introduction of new variables and constraints to the system. In practice, this could be realized using agents attached to the existing variables. These agents are evoked when the variable domain changes and they are checking some predefined condition on the domain, for example whether there is exactly one value in the domain. If the condition is satisfied then the agent introduces new variables and constraints (perhaps together with the new agents attached to the variables) and silently disappears. Note, that the variables and constraints introduced by the above mechanism are removed when the respective condition is revoked during backtracking. This is natural in the CLP framework where decisions are revoked automatically during backtracking.

We have already mentioned how to decide about the value of the variables act(*r,i*). Actually, this decision is a part of the labeling procedure which works like labeling in standard constraint satisfaction problems. The goal of labeling is to assign values to the variables and the only difference from the standard labeling is that the set of the variables to be labeled extends as the labeling proceeds. Adding new variables naturally influences the variable ordering in the labeling process so the schedule is being built in the order from demands to activities (introduction of dependencies) and from past to future (introduction of slots). This corresponds to the order of gradual introduction of variables. Note also that some variables may be left unassigned, in particular the variables describing the slots and demands that are postponed after the schedule end.

Notice finally that we do not distinguish between planning and scheduling in the above solving process so it is a total integration of planning into scheduling. In some sense, we can say that deciding the values of the activity and *q* variables corresponds to planning while deciding the values of the start time variables corresponds to scheduling. It is possible to interleave labeling of these variables or to decide the values of the activity and *q* variables first (planning and resource scheduling) and then to assign values to the start time variables (time scheduling). This depends on the selected scheduling strategy. Naturally all decisions are revocable via backtracking in case of reaching an infeasible state.

Discussion on challenges

Because the proposed framework is different both from existing planning and scheduling approaches, it brings many challenges. The main challenge is how to exploit the existing planning and scheduling technologies there. Because planning is relatively restricted in the proposed framework, we see the main challenge in the integration of existing global scheduling constraints like edge-finder (Baptiste and Le Pape 1996) to our framework. Note that

these constraints play a very important role in scheduling because they prune the search space significantly (Baptiste, Le Pape, Nuiten 2001). In our opinion, one of the biggest drawbacks of the current planning systems that integrate some scheduling features is impossibility to use these global scheduling constraints. Because in our framework we are working with the requests for activities rather than with the activities assigned to the resource, we cannot use the existing global constraints directly as well. Nevertheless, we believe that it is possible to reformulate either the global scheduling constraints to fit our framework or to extend our framework to exploit these constraints. Another difficulty is the dynamic introduction of the requests so the global constraints should be open in the sense of accepting new variables. We have studied such open global constraints in (Barták 2003a) and it seems that many global constraints can be opened in the above sense. The trade-off for some open global constraints is the decreased filtering power (less values are pruned). Thus, another challenge is to find out (automatically) when the constraint can be closed (no more variables will be coming) so the stronger domain filtering can be applied.

Conclusions

The main contribution of this paper is a new formal framework of production scheduling problems integrated with planning. We showed how such problems can be formulated by means of constraint technology and we presented both static and dynamic approach to solve the constraint model. We argued for the less memory demanding dynamic approach that dynamically extends the constraint model by adding new variables and constraints as the solving process progresses. This dynamic approach has been implemented in the scheduling engine of the Visopt ShopFloor system.

Despite the fact that we focused on the production scheduling, we believe that the proposed techniques are applicable to other integrated planning and scheduling problems. In particular, the problems where complex resources and activity dependencies play a significant role are highly relevant for the proposed framework.

Acknowledgments

Research is supported by the Czech Science Foundation under the contract no. 201/04/1102. The system Visopt ShopFloor has been developed for Visopt B.V.

References

Baptiste, P. and Le Pape, C. 1996. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group*.

Baptiste, P., Le Pape, C., Nuijten, W. 2001. *Constraint-based Scheduling: Applying Constraints to Scheduling Problems*. Kluwer Academic Publishers, Dordrecht.

Barták, R. 1999a. Conceptual Models for Combined Planning and Scheduling. *Electronic Notes in Discrete Mathematics*, Volume 4, Elsevier.

Barták, R. 1999b. On the Boundary of Planning and Scheduling. In *Proceedings of the Eighteenth Workshop of the UK Planning and Scheduling Special Interest Group* (PlanSIG), 28-39, Manchester, UK

Barták, R. 2002a. Visopt ShopFloor: On the Edge of Planing and Scheduling. In P. van Hentenryck (ed.): *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, LNCS 2470, Springer Verlag, Ithaca, 587-602.

Barták, R. 2002b. Modelling Resource Transitions in Constraint-based Scheduling. In W.I. Grosky, F. Plášil (eds.): Proceedings of SOFSEM 2002: Theory and Practice of Informatics, LNCS 2540, Springer Verlag, pp. 186-194.

Barták, R. 2003a. Dynamic Global Constraints in Backtracking Based Environments. *Annals of Operations Research 118, 101-119*, Kluwer.

Barták, R. 2003b. Real-life Manufacturing Problems: A Challenge. In J. Hoffmann and S. Edelkamp (eds.) Proceedings of ICAPS'03 Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks. Trento, p. 38-42.

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

Dechter R. and Dechter A. 1998. Belief maintenance in dynamic constraint networks. In *Proceedings of AAAI-88*, pp. 37-42.

Do M.B. and Kambhampati S. 2000. Solving planning-graph by compiling it into CSP. Proceedings of the Fifth International Conference on Artificial Planning and Scheduling.

Gallaire, H. 1985. Logic Programming: Further Developments, in: IEEE Symposium on Logic Programming, Boston, IEEE.

Long D. and Fox. M. 2002. International Planning Competition 2002. Toulouse, France.
http://www.dur.ac.uk/d.p.long/competition.html

Mittal, S. and Falkenhainer, B. 1990. Dynamic Constraint Satisfaction Problems. *Proceedings of AAAI-90*, USA, 25-32.

Nareyek, A. 1999. Structural Constraint Satisfaction. *Proceedings of AAAI-99 Workshop on Configuration.*

Pegman, M. 1998. Short Term Liquid Metal Scheduling. *Proceedings of PAPPACT98 Conference*, London, 91-99.

Srivastava B. and Kambhampati S. 1999. Scaling up Planning by teasing out Resource Scheduling. Technical Report ASU CSE TR 99-005, Arizona State University.