INTEGRATING PLANNING INTO PRODUCTION SCHEDULING: VISOPT SHOPFLOOR SYSTEM

Roman Barták, Roman Mecl
Charles University, Prague, Faculty of Mathematics and Physics

Abstract: Traditional planning deals with the problem of finding activities to satisfy a given goal while traditional scheduling solves the problem of allocation known activities to limited resources and to limited time. In many real-life problems both tasks should be accomplished together. In the paper we describe a scheduling engine of the Visopt ShopFloor system that integrates planning and scheduling component in a unified framework.

Key words: planning, scheduling, constraint reasoning, applications

1. INTRODUCTION

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 planning problems is usually 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, i.e., the set of activities is generated, and the scheduling problem is solved next, i.e., the activities are allocated 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. In (Barták, 1999b) we argued for a more

tighten integration of planning and scheduling where time and resource constraints play an important role in guiding the planner. The basic idea is to post 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 an approach bridging the gap between planning and scheduling from the scheduling side. Basically, it means that we are solving a scheduling problem where the activities can be introduced (planned) during the scheduling process. We use the concept of tight integration of planning and scheduling proposed in (Barták, 1999b). This concept utilises the underlying constraint satisfaction technology that was slightly modified to allow dynamic changes of the constraint store. The planning component uses mixed forward and backward chaining and it is fully integrated into the scheduler. Because the used planning technique is not very complicated we are speaking rather about enhancing schedulers by planning technology.

This is an application paper describing the solver behind the Visopt ShopFloor commercial scheduling system. The main purpose of the paper is not to present the GUI of the Visopt ShopFloor system with all its possibilities to model scheduling problems in the factories but to present the ideas that are used in the underlying solver. This solver is based on the above surveyed ideas of enhancing scheduling by adding planning capabilities, i.e., we concentrate on the problems where the structure of activities is not known in advance but it must be planned during the process of scheduling. The system is intended to production scheduling in real-life complex industries. Thus, the solved problem does not fit to any “standard” academic view of scheduling problems like job-shop, flow-shop, or open-shop. Probably the closest formal problem is the resource-constrained project scheduling problem but Visopt ShopFloor system covers even more complex problems, e.g. complex transition schemes of resources, including modelling set-ups with by-products and recycling etc. The system has already been tested in real-life factories and its development still continues.

The paper is organised as follows. First, we describe the problem area and we give some examples where introduction of activities during scheduling is necessary. Then, we show how the problems can be specified formally, i.e., what modelling tools are available for the users. After that we briefly sketch the system architecture. The main part of the paper is about the solver. We describe the basic technology, the constraint model, and the scheduling strategy. The paper is concluded with the results of some real-life projects and with discussion of future development of the system.

2. PROBLEM AREA

Visopt ShopFloor is a general scheduling system applicable to production scheduling. It means that the system is not designed for a particular factory but it can be applied to various scheduling problems.

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. Demands are specified by ordered items with ordered quantity and delivery time. The demand is satisfied if the ordered quantity of the item is ready for delivery at the delivery time. It means that the item must be produced, purchased, or already stored in the factory. Production of the item is done on resources called producers. The items are produced in batches (the resource schedule is described as a sequence of batches); the batch specifies a quantum of item that is produced together. It is possible to produce several items in a single batch (main products and co-products). The batch can also describe various processing formulas, i.e., the same item can be produced using different input items. If one batch produces some item then there must be another batch or delivery that consumes the item. In general, there could be many-to-many relation between producers and consumers (Figure 1). It means that the quantity of the item produced in a single batch can be split to several consuming batches of various resources and vice versa. The feasible schedule must ensure that the item flow in the factory is correct in the above sense.

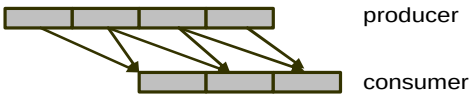

Figure 1. Items are flowing (arrows) between the batches (rectangles).

To summarise the above discussion, the goal of production scheduling is to generate a schedule for each resource in such a way that the item flow between the resources is correct. A schedule of the resource is a sequence of batches. Sequencing of batches can be further restricted by some transition criteria. For example, minimum and maximum number of batches of the same type (same output item) must be processed in a sequence. When the batch type is changed then a changeover batch must be inserted or a cleaning batch must be used when a given number of production batches is processed (Figure 2). Such transition scheme is fully specified by the user (see Section 3.1 for details).

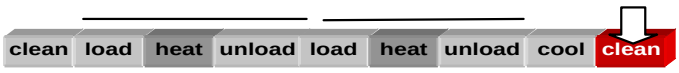

Figure 2. A resource schedule is a sequence of batches with complex sequencing constraints (e.g., after two load-heat-unload cycles there must be cleaning.

In addition to production resources there could be also moving resources that are used to transport items between the production resources. Moving is done in batches as well so the behaviour of the mover is similar to the behaviour of the producer.

Producers and movers are called main resources. There could be also secondary resources that assist to main resources. Tool and worker are examples of the secondary resource. Again, the schedule of the secondary resource is a sequence of batches (activities) - each such batch is synchronised with a batch of the respective main resource, i.e. both batches run in parallel (same start times and same completion times).

2.1 Why do we need planning?

In (Barták 1999a) we gave several examples where traditional static scheduling approach is not sufficient because some activities are not known before scheduling starts. Moreover, these activities cannot be planned in advance because their appearance depends on allocation of other activities. We call these activities the process dependent activities, which mean that these activities might or might not appear in the final schedule depending on some other activities. Nevertheless, appearance of the activity is strictly determined by the activities on which it depends (e.g. its suppliers and consumers) so there is no uncertainty for the activity existence. In the next paragraphs we give some examples of such process dependent activities. All these examples are taken from real-life problems.

In (Pegman, 1998), one of the first examples of the process dependent activity is given. Pegman describes a scheduling system for metal production. The metal blocks must have a particular temperature before they can be processed. Naturally, the temperature of the metal block is decreasing slowly after its heating so if the delay between the end of heating and the start of processing is too long then the temperature of the block might be too low. In such a case, the metal block must be reheated before it can be processed. Because re-heating consumes the resource (the oven), there must be a special re-heating activity introduced. Pegman uses a technique of dummy activity that is either active, if re-heating is necessary, or it is not used if the delay between heating and processing is short enough.

Another example of process-dependent activity is changeover that is necessary to be inserted between two activities of different type processed by the same resource. In constraint-based scheduling, the changeovers are usually modelled using a transition time between two activities. During such transition, the resource cannot be used. However, if some item is produced during changeover, so called by-product, or the changeover requires another resource, e.g. a worker, then we need an activity to model the changeover. Again, the appearance of the changeover activity is dependent on the actual allocation of other activities to resources. Thus the changeover activity cannot be planned in advance.

The appearance of the changeover activity that produces a by-product may lead to adding other activities consuming this by-product. For example, the by-product can be re-cycled and used to satisfy some demand. In general, we may have several process routes to satisfy the demand and these process routes may have a very different structure of the activities. It is possible to select one of the process routes in advance (planning) but in (Barták, 1999b) we argued to postpone this decision into scheduling stage when more information is available (e.g., we can use a by-product produced by some changeover).

Some of the above examples can be modelled using a technique of dummy activities. However, if the number of dummy activities is large then this technique is not applicable. In such a case, it is better to introduce the activities on demand during scheduling, i.e., to enhance scheduling by planning capabilities.

3. PROBLEM FORMALISATION

Visopt ShopFloor uses a general description of the production scheduling problem based on the resource-centric model (Brusoni et all 1996). The problem is specified by a set of resources, a set of dependencies between the resources, and a set of demands. In this section we give more details about these parts of the model. For simplicity reasons, we describe only the key features of the modelling framework.

3.1 Resources

The resource is described as a set of activities. For each activity the user specifies its duration and time windows when the activity is processed. The activity occupies the resource from its start time till its end time. The interruptible activities may run out of time windows but they must start and complete in a time window. The activity also specifies the produced and

consumed items with their quantities. In fact, the set of input and output items (with quantities) forms a lot and it is possible to have several lots in the activity. The number of lots is then restricted by the capacity of the activity.

We use the notion of activity to describe parameters of batches - a batch type; sometime we call it also a state. Thus, the user describes activities, but in the schedule we use batches of a particular activity (Figure 3). The batches are not overlapping. The user may specify the minimal and maximal number of batches of given activity that can be processed in a sequence. It is also possible to describe a transition scheme between the activities with transition times etc. The transition is not allowed until a minimum number of batches of given activity is processed and the transition is forced when the maximum number of batches is reached. The user may also describe a global batch counter, i.e., to specify activities whose batches are counted, and the activity to which we must switch if a given limit is reached.

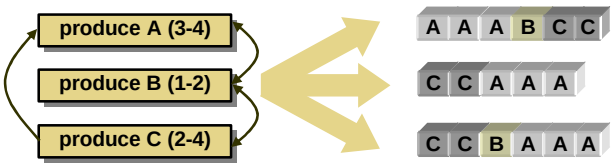

Figure 3. Activities are connected in a transition scheme with minimum and maximum number of batches per activity (left). This scheme restricts the feasible sequences of batches (right).

The above scheme of the resource allows natural description of complex resources. Changeovers and setups can be modelled as standard activities (if they produce some items or if they use secondary resources) or they can be modelled as a transition time between the activities. The global batch counters describe features like insertion of the cleaning batch after N production batches. Different processing formulas can be modelled using different lots in the activity.

The same scheme (Figure 4) can be used to model producers, movers, and secondary resources. Thus we can model rather complex behaviour of the secondary resource like a learning curve, e.g., after ten batches the worker is more experienced = transition to a new state. Or we can capture a recreation schedule, e.g., after three "production" batches the worker needs one "recreation" batch.

Resource
Activity
duration + time windows + interruptible
capacity
Lot
input + output items with quantities
Next activities with transition times

Figure 4. The basic schema of the resource model

3.2 Dependencies

The relations between the resources are called dependencies. Basically the dependency describes an item flow between batches of different resources. For each item, the user specifies a supplying resource with a supplying activity and a consuming resource with a consuming activity. Moreover, the user also describes the delay between the end of the supplying batch and the start of the consuming batch (Figure 5).

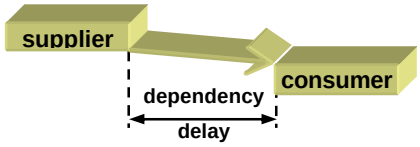

Figure 5. Dependencies express supplier-consumer relations

Note also, that is possible to specify several dependencies for a single item so one consumer may be connected to several suppliers and vice versa.

Basically, the dependencies express supplier-consumer relations but if we use some artificial item and negative delay, we can also model resource synchronisation via dependencies. Thanks to the flexibility of the above model there is no problem to describe alternative production routes, in particular to express many-to-many relations (Figure 1), or to model recycling. It is also possible to use dependencies for modelling simple stores by allowing variable delay (= storing time).

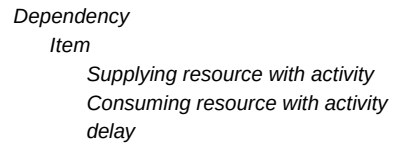

Figure 6. The basic schema of the dependency

3.3 Demands

To start-up production we need some demands. The demand is specified by an ordered item with quantity and required delivery time. The user may allow alternative items to be delivered instead of the ordered item and it is also possible to allow delaying of the delivery. The demands can be specified as consumers in dependencies so we know which resources can supply the final product.

3.4 Purchases

Usually, the items are produced in the factory but some items can also be purchased from external suppliers. It could be raw material, intermediate items, or even the final products. To model such situations, it is possible to specify whether given item can be purchased or not. Then, purchase plays a role of the supplier in dependencies

3.5 The task

When resources, dependencies, demands, and purchases are specified then the goal of the scheduling system is to generate a plan covering the demands and satisfying the production constraints (a feasible plan). There are no batches known in advance, the system has to find out what batches are necessary and to which resources these batches should be allocated. Only the batches describing the initial situation of some resources may be known. It means that we are solving a planning problem under time and resource constraints (Figure 7).

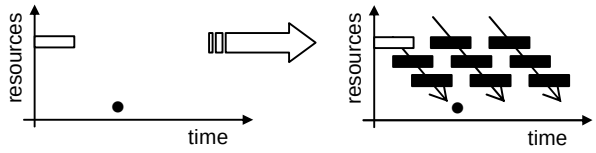

Figure 7. The scheduling task is to find out batches (rectangles) covering the demands (dots) and to allocate the batches to resources.

So far, we discussed only the feasibility problem but we can also model optimisation. It is possible to assign a cost parameter to every object in the schedule and then the task is to minimise the sum of costs. For example, the user may specify penalty for delaying deliveries or for using alternative items. It is possible to assign cost to batches (e.g. energy consumption) or to dependencies (e.g., moving/storing cost). More details about optimisation issues can be found in (Barták, 2002b).

4. SYSTEM ARCHITECTURE

Visopt ShopFloor system consists of two independent parts: the ShopFloor user interface and the solver (Figure 8).

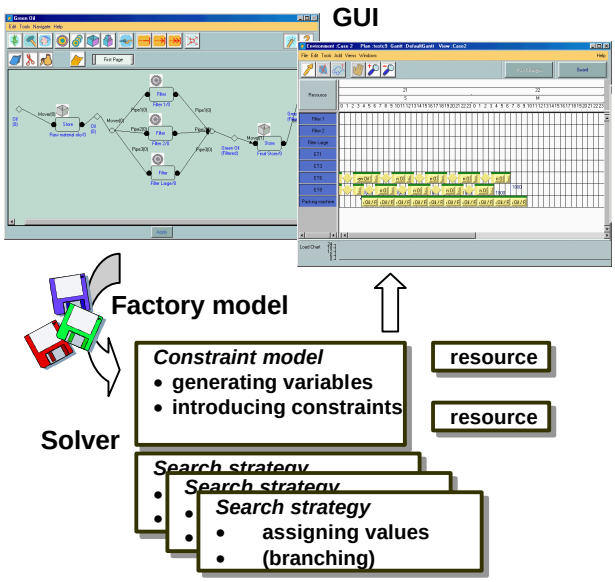

Figure 8. Visopt ShopFloor System Architecture

The *ShopFloor user interface* serves for the problem specification. It provides a graphical modelling environment where the user describes the structure of the factory, i.e., the resources and the production routes (the relations between the resources). It is possible to model the problem from scratch or to upload the problem from an ERP database. In the second case, the ShopFloor visualises the problem description so the user can easily make changes in the model. The ShopFloor user interface uses its own database model that is converted to a factory model before scheduling.

The *factory model* is a structured text file containing a complete description of the problem including the demands. This file is in a human readable form so it is possible to check manually the data or to generate the file using arbitrary text editor (for example to model simple benchmarks without going through the complexity of the GUI). It is even possible to pre-process the model before it is sent to the solver. The factory model serves as an interface specifying the input to the solver so the solver is independent of the user interface. We described the basic components of the factory model in the previous section.

The *solver* is responsible for planning/scheduling. It consists of the constraint model that is generated automatically from the factory model and of the scheduling strategy. We use a modular architecture of the solver so it is possible to add new resource types later. Also the scheduling strategy can be exchanged easily to reflect a particular type of the problem. We give more details about the solver in the next section. The solver produces a plan that is returned to the ShopFloor user interface. The plan is presented to the user in the form of a Gantt chart.

Separating GUI and solver gives us flexibility in designing dedicated user interfaces for the solver or in using the user interface just for data visualisation.

5. THE SOLVER

The solver, or we call it also a scheduling engine, gets the problem specification in the form of a factory model and it generates a feasible plan if it exists. We use a constraint satisfaction technology to implement the solver. First, the solver generates the constraint model using the factory model. Second, the solver tries to find a solution of the constraint model.

Unfortunately, we cannot use the traditional static view of constraint satisfaction problems (CSP) where the variables and constraints are specified first and variable labelling is done next. The problem here is the presence of the planning component; in particular the batches are introduced during

scheduling. Thus the structure of variables and constraints is changing as the search progresses.

In the next section we explain how we use the constraint technology to overcome the difficulty with static CSP. Then we describe the constraint model, i.e., how the variables and constraints are introduced. Finally, we characterise the scheduling strategy.

5.1 Technology

Traditional formulation of CSP is static in the sense that the variables and constraints are defined first and the search is done next. Modelling planning problems as CSP is hard because of the variability of the plans (Nareyek, 2000). It means that the problem cannot be modelled statically (with dummy variables) because of large size of the problem formulation. Thus CSP is used when some planning decisions, e.g., about the plan length, are done. Another possibility is using some generalisations of CSP like Structural CSP (Nareyek, 1999).

The planning sub-problem in our problem area consists of decision about batches necessary to satisfy the demands. The scheduling sub-problem consists of resource allocation and time scheduling. Note that solving the planning sub-problem separately is rather easy, what makes it hard is assuming time and resources. Thus it seems that we can use some simple planning technology like backward chaining combined with look-ahead using constraints. It means that in the first step all alternative batches to satisfy the demands are introduced including the time and resource constraints. Then the constraint-based scheduler selects the necessary batches and allocates them to resources. As soon as the batch is selected, the planner introduces all alternative batches that supply items to this batch etc. Basically the role of the planner is to watch the current partial schedule and when some batch is missing, the planer introduces all the alternative batches for its position.

Constraint satisfaction plays important role in the above process; in particular constraint propagation is used to filter the planning alternatives. Notice that new batches are added when some scheduling decision is done, i.e., when a value is assigned to some variable. It means that variable labelling interleaves with problem formulation, i.e., new variables and constrains are added when the value is assigned to the variable. This could be naturally implemented in constraint logic programming (CLP) where constraints prune the search space.

Recall that the original formulation of CLP is based on exactly the same idea - using constraints to prune the search space without distinguishing the problem formulation and labelling (Gallaire,1985). Recently, when CSP

appeared, the separation of problem definition and labelling was introduced to CLP as well. This second approach is more efficient if all the alternatives can be captured in a static set of variables and constraints (Van Hentenryck and Deville, 1991). As we already mentioned, when planning decisions are involved, this is not necessarily the case. Thus we propose to use CLP where labelling is interleaved with introduction of variables and constraints. Notice finally, that despite the dynamic character of the constraint satisfaction problem we do not need to use frameworks like Structural CSP (Nareyek, 1999) or Dynamic CSP (Mittal and Falkenhainer, 1990) which require a special implementation of the constraint engine. We can use the existing constraint solvers where labelling is seen as a procedure for adding new constraints (and variables) to the system.

5.2 Slot Representation

Traditional scheduling systems use a task centric model of the problem, i.e., the activities belonging to a single task (demand) are connected in a chain with the precedence relations between the activities. However, this model assumes rather simple behaviour of resources, typically, unary and cumulative resources are used. Moreover, it is hard to model sharing of activities between the tasks (many-to-many relations), the changeover activities, or recycling. In our problem area, the transition scheme makes resources more complicated so we decided to use a resource centric model.

The resource centric model is realised via a slot representation. The partial resource schedule is represented here as a sequence of slots, where the slot is a shell to be filled by the batch. This model is similar to timetabling models - the main difference is that the slots are not assigned to time in our representation. In particular, the duration of the slot is not fixed (it depends on the batch that will be filled in the slot) and the slot may slide in time. Still, the sequence of slots cannot be changed so it is not possible to swap position of two slots (but the batches in the slots may be swapped – simply slots are filled by batches in a different order).

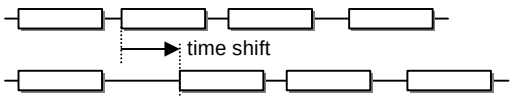

Figure 9. Slots cannot be swapped but they can slide in time.

Each slot is specified by a set of variables describing its position in time (start, end, duration) and specifying what activity can be filled in the slot (a state variable). Note that the slots may be introduced even if we do not know

yet the batches in the slots. Thus we can post the constraints among the slot variables to prune the search space by restricting which batches can be allocated to a given resource in a given time. Typically, time variables are connected to the state variables to describe time windows and duration of batches. We use tabular constraints for this purpose, where a tabular constraint is a general binary constraint with user defined domain (Barták, 2000). Because sequencing of slots is fixed as well, it is possible to model the transition scheme using the constraints posted between two neighbouring slots. To count batches of the same state, we use a special variable called serial that participates in the transition constraints. A detailed description of the transition constraints can be found in (Barták, 2002a).

The maximal number of slots covering the schedule period can be computed in advance so the slots can be introduced before scheduling. The difficulty of this approach is that it may introduce too many slots leading to huge CSP for large-scale real-life problems. Therefore we decided to introduce a minimal number of slots (from left) that can cover the current demands for the resource. Note that the list of demands for the resource may extend as scheduling progresses and the batches are being scheduled in other resources. Thus, we use a guard watching the list of demands for the resource and when the existing slots cannot cover the current demands, then the new slots are added to the end of the slot list. It is also possible to add new slots due to the transition scheme, e.g. to model resources that cannot be interrupted (like big ovens in metal production factories).

The slot representation models naturally the batch resources. Its advantage is that we can post resource constraints early so we can use them to prune the search space even if the batches allocated to the resource are not decided yet. Filling the slots corresponds to batch sequencing. A small disadvantage is that when a batch is filled into the slot then its position in the sequence is fixed. If we want to insert a batch before some batch then this already filled batch must be moved to the next slot so it frees the slot for the inserted batch. Thus decision about batches in slots must be done carefully assuming all the demands for the resource.

5.3 Dependencies

Dependencies are used to propagate demands between the resources. Recall that dependency describes a supplier-consumer relation so it connects the supplying object (batch or purchase) with the consuming object (batch or delivery). Basically, the dependency is a constraint binding the end time of the supplying object with the start time of the consuming object. It also describes what quantity is moved between the objects. At the beginning, we know only the deliveries corresponding to user demands but we do not know

the actual suppliers for the deliveries. Thus we cannot post the dependency constraints. Nevertheless, the factory model specifies what resource and activity may supply the ordered item. Even if the actual batch(es) is not known we can find the slots in possible supplying resources to which the supplying batch(es) can be filled. Then we can post a conditional constraint between the slot and the delivery binding the times if the quantity moved between these objects is non-empty. The problem is that the number of eligible slots may be very large so the number of conditional constraints will also be very large. Thus in real-life problem this eager method is not applicable. We use a more lazy method that connects only the first slot of each possible supplying resource to the delivery. If we find later that the quantity moved between this slot and the delivery is empty and still some quantity must be supplied then the next possible slot is connected with the delivery etc. (Figure 10). This approach is applicable to finding suppliers as well as consumers of batches, i.e., when a batch is filled in some slot then we can start the above process of finding missing suppliers and/or consumers.

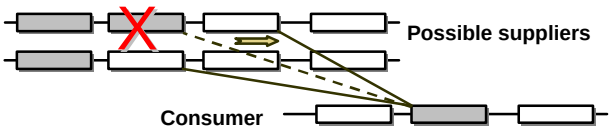

Figure 10. The dependency generator introduces the dependencies to the first possible slot (from left). If the slot is not dependent (X) then the dependency is moved to the next free slot.

The above dependency mechanism realises the backward chaining method of planning. We introduce the alternatives with short look ahead and the decision about the used alternative is postponed to the scheduling stage. Because the constraints are already posted, we are speaking about active decision postponement (Joslin and Pollack 1995).

Each slot keeps a list of dependencies going to the slot and this list is used during labelling to decide which dependencies will really go to the slot (the quantity in the dependency is non-empty). Moreover, it is possible to keep the list of dependencies (demands) for each resource and to apply some ordering constraints to them, like in (Laborie, 2001) and (Baptiste and Le Pape, 1996).

5.4 Scheduling Strategy

As we described above, the constraint model is dynamic but autonomous. It means that the variables and constraints are introduced automatically when

values of other variables are known. The decisions about variables' values are done by the scheduling strategy. The only but significant difference from the traditional constraint satisfaction is that the set of variables is increasing as the search progresses. Moreover, the set of variables may be different in different search branches.

To implement the scheduling strategy we use depth-first search. Naturally, the variable ordering must respect the dynamic nature of the problem so let us look at the scheduling strategy from the scheduling point of view. The goal of the scheduling strategy is to fill the slots by batches, i.e., to decide about the value of the state variable in each slot, and to decide about the connections between the batches. We call this process closing the slot. We know that the slots are introduced in left to right manner and that the dependencies are first started from demands. This observation determines the ordering of slots to be closed (variable ordering). We decided to generate the schedule in slices going from left to right. The slots in each slice are closed in the order from demands/deliveries to purchases (Figure 11). The width of the slice is determined by the user (it is heuristic information). We call the process of closing the slots in the slice a scheduling step.

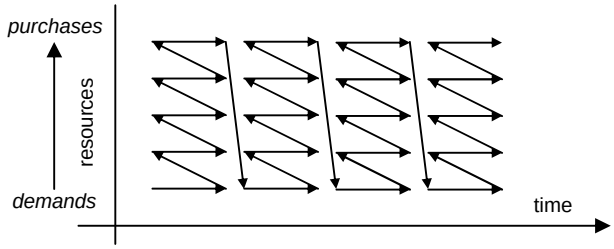

Figure 11. Variable (slot) ordering used by the scheduling strategy.

When the slot is selected, the next question is what batch should be filled in, i.e., what value should be assigned to the state variable. This decision is done using the dependencies going into the slot. The scheduling strategy selects some dependency that can be connected to the slot. Then it sets the quantity in the dependency to be greater than zero, so the dependency is effectively anchored to the slot. This anchoring usually determines the value of the state variable (if not, the scheduling strategy selects one). The process of selecting dependencies going to the slot is repeated until the batch capacity is exhausted. The remaining dependencies are moved automatically to the next possible slot (Figure 12). The selection of dependencies is driven by heuristic that uses information about time and cost; dependencies going to earlier times and leading to less expensive production are preferred.

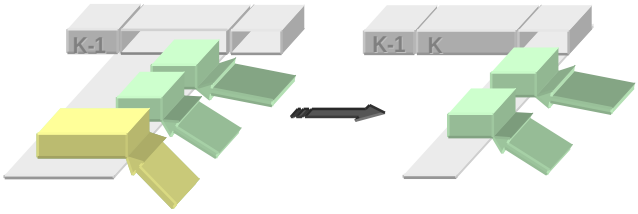

Figure 12. When the dependency is selected for the slot, then the incompatible dependencies are moved to next free slot.

During the first round, the batches are filled to the slots in the slice and the dependencies between the batches are anchored. Thus the planning sub-problem and the resource allocation sub-problem are solved. To complete the scheduling step, the scheduling strategy decides about time allocation of closed batches; the earliest times are preferred.

Notice that the choice of variable ordering ensures that the variables are already present in the system. Unfortunately, the dynamic nature of the problem complicates the direct usage of more advanced search techniques that are basically oriented on static problems like Limited Discrepancy Search (Harvey, Ginsberg 1995). We are currently exploring the possibility of how to apply the good ideas from these more advanced search techniques in our dynamic problems.

6. THE RESULTS

The Visopt ShopFloor has been tested in several pilot projects, in particular in one of the biggest and famous candy producers in The Netherlands, in one of the biggest dairies in Israel, and in a chemical factory in Germany among others. The goal of these pilot projects was to model the most complex production lines in these factories and to provide a feasible schedule for them. At this stage, it is hard to estimate savings when Visopt ShopFloor is applied because these production lines were scheduled manually so far. It is even complicated to evaluate quality of the generated schedules because there are no existing schedules to compare with. Nevertheless, the production experts in the companies agree that the generated plans satisfy the production rules and that they "look good". Note, that in many real problems, the quality of the produced schedule is rather subjective than objective. Even if the solver uses an objective function (cost) to produce “good schedules”, still, it is the user who decides about the quality of the schedule. One of such criterions could be that the plan is optimized enough that the experienced human planner is not able to produce

better plan. The big win is that our system can cover all the features of the complex production lines that the other tested systems like SAP APO cannot model.

Because we cannot compare our system to existing schedulers (other schedulers cannot even model the problems that we are solving) we provide here the results of some of our tests based on real-life data from the above mentioned enterprises. To show capabilities of the solver we report the size of the problem, the size of the solution, and the runtime.

The solver is implemented in SICStus Prolog (3.8.7) and the tests are run on Celeron 500 MHz with 192 MB RAM. Prolog programming language is chosen for its rapid prototyping capabilities and for natural integration of the constraint satisfaction technology.

Table 1 describes the size of the problems. We include the number of resources in the factory, the total number of states (the types of batches), and the number of different items going between the resources. These numbers characterise a given production line. In some sense, Table 1 indicates the size of the planning domain. To describe the size of the actual problem, we also specify the number of demands with the total ordered quantity and the duration of the scheduled period in time units. The quantity and time attributes roughly describe the size of the variables' domains used in the system. For example 10080 time units correspond to one week production with a minute resolution. It means that the scheduler must produce plans for one week where we know what is going on in every minute.

Table 1. Model size for five test problems.

	resources	states	items	demands # / quantity	duration
1	57	704	45	256/196744	840
2	22	677	56	9/7600	3168
3	28	115	34	1/50	8640
4	19	334	47	50/144000	10080
5	34	574	294	45/88485	11520

Table 2 describes the size of the solution and the runtime. We report here the total number of batches and the total number of dependencies in the final schedule. From these numbers we can estimate roughly the number of variables and the number of constraints in the final schedule. For example, each batch is described by at least ten variables (the actual number depends on the number of items in the batch) and each dependency is described by at least six variables. However, note that much more variables and constraints are used during scheduling because many alternative batches and dependencies must be explored. For example, if many-to-many relations are used between the resources then the number of introduced dependencies is very large.

Table 2. Solution size and runtime.

	batches	dependencies	runtime (sec.)
1	990	1428	234
2	651	898	131
3	256	310	221
4	1000	1441	302
5	5807	10175	10095

The above results show that we can solve problems close in size to traditional scheduling problems and much larger than traditional planning problems. The large-scale scheduling problems contain about twenty thousands activities (personal communication to Wim Nuiten from ILOG) and the plans generated by state-of-the-art planners consists of tens of activities (Long and Fox, 2002). The results are not surprising because the plan complexity in our models is not very large. What makes them hard is satisfaction of time and resource constraints. On the other hand, the activities are introduced dynamically in our system according to the production rules. The traditional scheduling technology cannot be applied there due to the size of the static problem formulation. Even if we introduce the batches dynamically, still a lot of memory is necessary to resolve the planning sub-problem. For example, the model 5 is solved using model decomposition to fulfil the current memory limit of SICStus Prolog (256 MB on 32-bit architectures).

7. CONCLUSIONS

In this paper, we show that tight integration of planning and scheduling is possible and that it extends modelling capabilities of the scheduling systems. Constraint satisfaction technology proved itself to be flexible enough for modelling such integrated planning and scheduling problems. However, it is necessary to use a more dynamic view of CSP. In particular the existing global constraints need to be "open" to accept incoming variables. We proposed a general mechanism of dynamic global constraints in (Barták, 2003). Our experience also confirms that large-scale integrated planning and scheduling problems can hardly be modelled in a static way using dummy variables. Large memory consumption is one of the difficulties that we are solving now despite the fact that we are using only a limited number of dummy variables to model planning alternatives.

The presented technology is used in a scheduling engine of the commercial system Visopt ShopFloor. The integrated planning component is the main difference of our system from the traditional schedulers. However, the dynamic nature of our system is different from on-line (reactive)

scheduling - neither the demands nor the factory is changing during the scheduling process. We are working on a rescheduling feature that allows the software to react faster to changes in the production environment.

The unique features of Visopt, which the other scheduling systems cannot cover, include modelling of complex transition schemes for resources, modelling of arbitrary dependency structure of the factory, modelling of set-ups, cleaning, and maintenance including by-products, and modelling of process and item alternatives. Moreover, Visopt ShopFloor attempts to be a general scheduler where the customer describes the problem in a declarative way and the system generates schedules automatically. Other scheduling software is either provided as a toolkit (e.g. ILOG Scheduler), so the particular scheduler must be programmed using this toolkit, or the software solves a particular scheduling problem but it cannot be extended to other problem areas. Opposite to these systems, Visopt ShopFloor (Visopt, 2002) provides intuitive graphical modelling environment independent of the solver, generality covering many scheduling problems, and extendibility via adding new type of resources.

ACKNOWLEDGEMENTS

The research is supported by the Grant Agency of the Czech Republic under the contract no. 201/01/0942 and by Visopt B.V..

REFERENCES

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

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. *Proceedings of the Eighteenth Workshop of the UK Planning and Scheduling Special Interest Group* (PlanSIG), Manchester, UK, 28-39.

Barták, R., 2000. A General Relation Constraint: An Implementation. *Proceedings of CP2000 Post-Workshop on Techniques for Implementing Constraint Programming Systems* (TRICS), Singapore, 30-40.

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

Barták, R. 2002b. 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, 587-602.

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

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

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

Harvey, W.D. and Ginsberg, M.L. 1995. Limited Discrepancy Search. *Proceedings of 14th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 607-613.

Joslin, D. and Pollack M.E. 1995. Passive and Active Decision Postponement in Plan Generation. *Proceedings of the Third European Conference on Planning (ECP).*

Laborie P. 2001. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results. *Proceedings of 6th European Conference on Planning (ECP)*, Toledo, Spain, 205-216.

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*.

Nareyek, A. 2000. AI Planning in a Constraint Programming Framework. *Proceedings of the 3rd International Workshop on Communication-Based Systems (CBS)*.

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.

Van Hentenryck, P., Deville, Y. 1991. The Cardinality Operator: A New Logical Connective for Constraint Logic Programming. *Proceedings of the International Conference on Logic Programming*, 745-759.

Visopt B.V. 2002. http://www.visopt.com