Visopt ShopFloor: Going Beyond Traditional Scheduling

Roman Barták

Charles University, 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. Visopt ShopFloor is a generic scheduling system for solving complex scheduling problems. It differentiates from traditional schedulers by offering some planning capabilities. In particular, the activities to achieve the goal are planned dynamically during scheduling. In the paper, we give a motivation for the integration of planning and scheduling and we describe how such integration is realised in the scheduling engine of the Visopt ShopFloor system.

1 Introduction

Planning and scheduling are closely related areas but usually the problems from these areas are solved separately using a different technology. The planning task is to generate activities to achieve some goal while the scheduling task is to allocate the known activities to available resources and time. When both tasks are included in the real-life problem then usually the planning component generates the activities in advance and the separate scheduling component allocates the activities to the resources and time [18]. As we argued in [2], such separation is not appropriate if the problem environment is more complex and if the planning decisions are strongly influenced by the scheduling decisions (like the introduction of set-up activities with by-products). Our proposal how to solve the problems on the edge of planning and scheduling is based on the integration of planning and scheduling in a single solver [3].

In [6] we described our realisation of the integrated planning and scheduling system called Visopt ShopFloor. In this paper, we present the unique capabilities of this system using a particular example of the problem going beyond traditional scheduling.

The paper is organised as follows. First, we highlight the main features of the problem area and we describe one particular benchmark problem that can be solved by our system and that the conventional schedulers cannot handle. Then we present the technology and the basic ideas behind the solver. The paper is concluded with the results of the benchmark problem and we also show some results of real-life models.

2 The problem

Traditional scheduling deals with the problem of allocating known activities to available resources and time. Usually, the resources are rather simple; they define a limited capacity for processing the activities. Either we have a unary resource, where only one activity can be processed at a time - this is sometimes called disjunctive scheduling. Or we have a cumulative resource where more activities can be processed in parallel provided that the resource capacity is not exceeded - this is called cumulative scheduling. Distinction of unary and cumulative resources is important because a resource constraint with stronger filtering can be defined for unary resources [1]. Despite the widespread use of unary and cumulative resources in traditional scheduling applications, neither one cares about alignment or sequencing of activities in the resource (we explain these notions later in Section 2.1).

In addition to the resource constraints restricting the allocation of activities, the traditional schedulers allow the definition of precedence constraints between the activities. Usually, the activities are grouped into tasks, where a prescribed sequence of activities must be followed. Therefore we are speaking about the task-centric models [9,2]. Job-shop scheduling [7] is a typical example of the task-centric view of the scheduling problem. Constraint-based scheduling [20] is more general by allowing precedence relations between arbitrary activities but it still requires knowing the activities in advance.

In the following sub-sections we show that the real-life problems are more complex than the above pure schema of the scheduling problems. In particular we give examples of the resources with more complex behaviour going beyond the unary/cumulative classification. We also explain why a fixed task schema is not appropriate to model some production processes. The section is concluded by a description of an example problem that contains some of these features.

2.1 Complex resources

Unary (machine) and cumulative (store) resources are typical representatives of resources but in some production environments like process industries the behaviour of resources is more complex. In particular, alignment and sequencing of activities is important. In Visopt ShopFloor we are modelling batch production with a complex transition scheme.

Batch production means that the activities can be processed in parallel but if two activities overlap in time, they must start and finish at the same times. Such overlapping activities form a batch. In addition to the capacity restriction we also have a compatibility restriction, i.e., the activities are tagged by a type and only the activities of the same type can be processed in parallel.

In addition to batch production we can model a *complex transition scheme*. The resources are described using states and transitions between the states. At any time, a resource is in exactly one state and the state can be changed only according to the transition scheme. Moreover, the number of batches processed at each state can be limited. We now give some examples how the transition scheme is used to model behaviour of a real resource.

Let us consider a resource with two modes of production, parallel and serial. There is no restriction about the number of batches processed in the serial mode but exactly three batches are processed in the parallel mode. The restricted number of batches in the parallel mode is due to the following technological reason. Some by-product is outputted during the parallel production and this by-product is temporarily stored close to the machine. The temporal storage is full after three production batches and thus a recycling batch must be processed before the production can continue.

To make the transition scheme even more complex, we can consider that from time to time there must be a cleaning batch inserted. Moreover, cleaning cannot be done while some by-product is stored in the resource. We discuss the rules about insertion of the cleaning batch later in the paragraph about batch counters.

The above transition scheme can be easily described via a state transition graph where each state is tagged by a minimum and a maximum number of batches processed in this state (Figure 1).

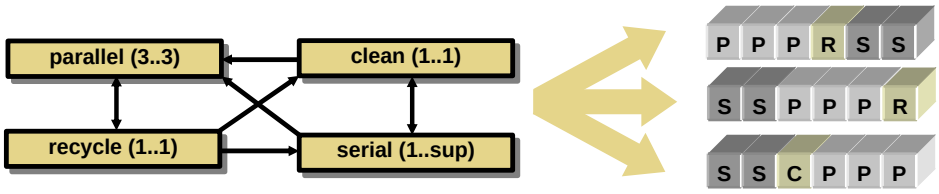

Fig. 1. Behaviour of many resources can be described using states with a minimum and a maximum number of batches per state (in brackets) and using a transition scheme between the states (left). This transition scheme must be followed during batch sequencing (right).

The transition scheme with the minimum and the maximum number of batches per state provides a flexible framework for modelling real-life resources. For example, it is easy to describe a learning curve of the worker. Let us assume that the worker needs first four batches to learn how to use the machine, i.e., duration of these batches is longer than duration of all following batches. We allow tagging the states by attributes, like duration and time windows, and these attributes are then applied to all batches of the state. Thus, the above worker can be modelled via a state transition scheme with two states: beginner and experienced (Figure 2). The batches processed in the beginner state are longer than the batches processed in the experienced state.

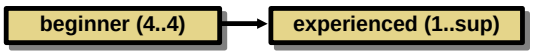

Fig. 2. State transitions can describe evolution of the resource, e.g., after a sequence of batches of given state, the resource irreversibly changes its state.

The above described transition scheme allows counting the batches of the same state. However, in many situations the users need to count batches of different states, e.g. to model insertion of the cleaning batch after a specified number of production batches. Thus, in Visopt ShopFloor we introduce the concept of a general batch counter that counts the batches across several states (Figure 3). This counter restricts further the sequencing of batches.

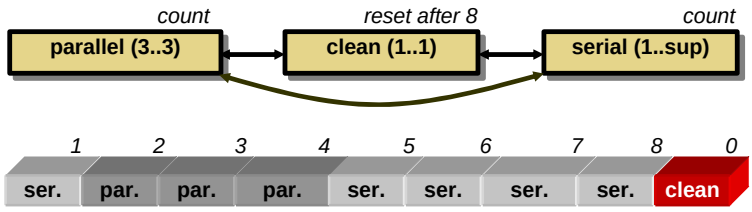

Fig. 3. Batch counters count batches across more states to model situations like forced cleaning after eight production (parallel or serial) batches.

It is hard or even impossible to model the above-described resources in the conventional scheduling. The main difficulty here is the transition scheme with the batch counters that forbid some transitions while force other transitions. It means that sequencing of batches is not arbitrary and the appearance of the batch depends on the allocation of other batches [2,17]. Thus the batches cannot be introduced in advance and it is more convenient to plan the batches dynamically during scheduling, i.e., to integrate planning and scheduling as we proposed in [3].

2.2 Resource dependencies

In the conventional scheduling systems, the direct relations between the activities are described via precedence constraints. These precedence constraints can be seen as an abstraction of the item flow between the activities - the item must be produced before it can be consumed. However, a simple precedence relation is not enough to model many real-life dependencies. The item must be produced before it is consumed but sometimes the delay between the end of production and the start of consumption should not be too long. For example, the item is cooling after its production and some minimal temperature is required when the item is consumed. This can be modelled easily in constraint-based scheduling where the simple precedence relation is substituted by tighter constraints:

$$\texttt{min\_delay} \leq \texttt{consumer\_start - supplier\_end} \leq \texttt{max\_delay}.$$

The problem is when we do not know which activities are connected using the above precedence constraints, e.g., when there are several process routes for a single item. For example, assume that either the item is produced in a parallel mode when two machines co-operate and a worker is necessary (Figure 4 left), or the item is produced in a serial mode when the item flows from the first machine to the second machine and no worker is necessary (Figure 4 right). The structure of the production route is different in the above cases, namely different batches are used with different relations between them. Conventional scheduling requires one production route (task) to be chosen before scheduling (i.e. during planning). We propose to postpone this decision to the scheduling stage when more information about the batches is available [3].

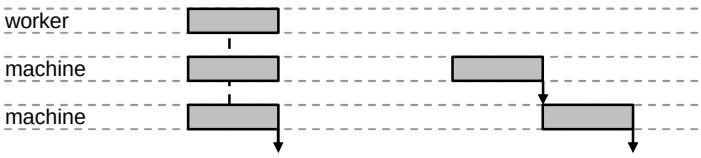

Fig. 4. In the real-life factories, the item can be typically produced using more processing routes, e.g. via a parallel production when two machines run in parallel and a worker is required (left) or via a serial production when the item is pre-processed in the first machine and then finished in the second machine (right).

Another difficulty of the conventional scheduling is modelling many-to-many relations between the batches, i.e., the batch has more suppliers and/or more consumers, and modelling recycling. In recycling, the set-up batch produces a by-product that can be used to satisfy some demands. Because the set-ups are not known until the production batches are allocated, it is not possible to plan recycling in the foregoing planning stage.

To address the above issues, we propose to describe supplier-consumer dependencies between the resources rather than to specify precedence relations between particular activities. Each supplier-consumer dependency is specified by the supplying and the consuming resource (and their states) and by the delay between the end of the supplying batch and the start of the consuming batch. When the dependency is established between two batches, the dependency describes also the quantity moved between the batches. Therefore a single supplying batch can be connected to more consuming batches and vice versa.

The supplier-consumer dependencies model naturally the item flow in the factory so they provide a declarative description of the processes in the factory. We can see them as a specimen for the precedence constraints that are posted when the batches of given type are introduced dynamically during scheduling (see Section 4). Figure 5 shows an example how the user describes the processes, i.e. the supplier-consumer dependencies using the graphical user interface of Visopt ShopFloor.

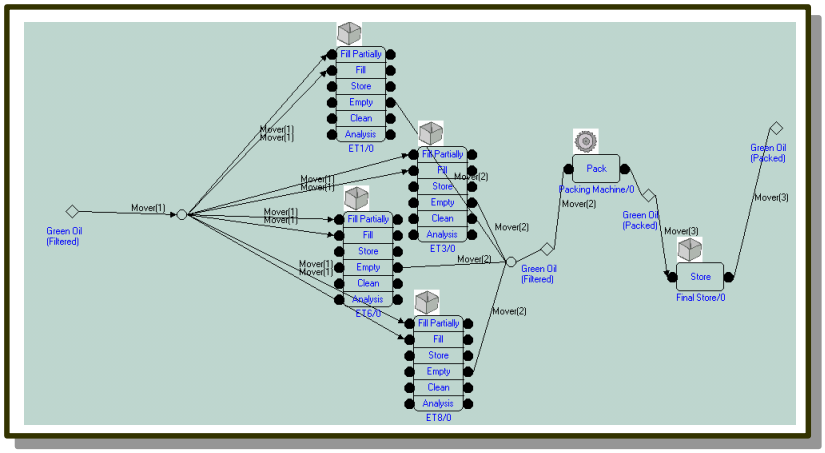

Fig. 5. Visopt ShopFloor graphical user interface describing an item flow.

2.3 The task at a glance

The Visopt ShopFloor system concentrates primarily on the problems going beyond traditional scheduling. Let us now summarise the task solved by the Visopt ShopFloor by giving a particular benchmark example.

Let us consider a small factory with two machines, r1 (Figure 3) and r2 (Figure 1), producing a single final item. These machines run either in a parallel mode or in a serial model (Figure 4). In the parallel mode, the batches of both machines run in parallel and a worker is required. One final item is outputted from the batch and duration of this batch depends on the experience of the worker (see below). In the serial mode, the machine r1 pre-processes the item (3 time units) that is finished in the machine r2 (3 time units). There is no delay for moving the item from r1 to r2.

During the parallel production, a by-product is produced. This by-product can be recycled only on the machine r2 and we need three by-products to get a single final item. Recycling takes 2 time units and it must be done immediately after the three batches of the parallel processing (Figure 1).

Both machines require cleaning after eight production batches or sooner (Figure 3) and the cleaning must be done at the same time on both machines. At the beginning, both machines are clean.

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

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

As we can see from the above example the goal of the system is to find out the batches that are necessary to satisfy the demands (planning) and to allocate these batches to available resources (scheduling). A plan/schedule for a given time period is returned to the user.

In this paper we discuss only feasibility issues, but the Visopt ShopFloor does optimisation based on cost as well. The conventional schedulers use some objective function like makespan, tardiness, or earliness to define quality of the schedule. However, in real-life environment the schedule quality is usually subjective, evaluated by the plant persons. To model these subjective criteria we use the cost parameters attached to batches, transitions, dependencies etc. The total cost is then used to guide scheduling, for details see [6].

3 The technology

Traditional scheduling technology is either based on special scheduling algorithms [7] or some general schema like constraint-based scheduling [20] is applied. If the activities are known in advance then it is quite natural to model the scheduling problem as a constraint satisfaction problem (CSP). However if the planning component is present then the static approach is hardly applicable due to variability of

possible plans [16]. Some approaches try to fit the planning problem into the static concept of CSP via dummy activities [8,17] but it works only when the planning branching does not lead to many different structures of the plan. Other researchers propose to use some generalised concept of CSP that provide more dynamic features like Dynamic CSP [14] or Structural CSP [15].

In the Visopt ShopFloor system, we solve the scheduling problems where the appearance of the activity depends on allocation of other activities. In terms of CSP it means that the existence of some variables and constraints depends on assigning a value to another variable. Moreover, the variable/constraint disappears from the system only when the original assignment is withdrawn, i.e., during backtracking. Having this in mind we decided to use the existing technology of Constraint Logic Programming (CLP) in the way this framework was originally defined [10], i.e., the constraints are used to reduce the search space of the logic program.

Opposite to the standard CSP technique (i.e., define the variables and the constraints first and then do labelling) we propose to interleave the labelling stage with the introduction of new variables and constraints. Basically, it means that we model the planning decisions (branching) using the disjunctive constraints (constructive disjunction). When some element of the disjunction is selected then the system automatically introduces other variables and constraints corresponding to the selected planning branch. This gives us the freedom to define different sets of variables and constraints in different branches of the search tree, i.e., to explore different plans. Thus planning decisions are resolved during scheduling.

4 The solver

The Visopt ShopFloor system consists of two independent parts: the ShopFloor graphical modelling environment and the scheduling engine (see Figure 6).

In the *ShopFloor*, the user specifies completely the problem to be solved. In particular he or she describes the available resources, i.e., their states and transitions, the item flow, i.e., the supplier-consumer dependencies (see Figure 5), and the customer orders (demands). Data can be entered and modified manually or they can be extracted automatically from the databases of ERP systems. The ShopFloor module generates the problem description in the form of a text file called a factory model that is passed to the solver.

The *factory model* contains a complete description of the problem (resources, dependencies, and orders) in a human readable form. It means that the factory model can be explored, prepared, and modified in an arbitrary text editor. This file is the only input to the scheduling engine.

The *scheduling engine* (the solver) first generates a constraint model from data (from the factory model) and then it searches for the solution. The solver has a modular architecture so it is possible to add a new module describing a new type of resource. Also, the search strategy is a separate module so it can be exchanged by a new strategy. The scheduling engine returns the plan/schedule into the ShopFloor that displays it in the form of a Gantt chart.

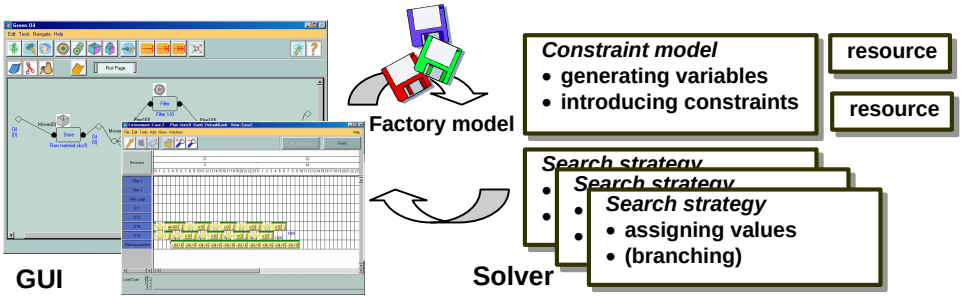

Fig. 6. The Visopt ShopFloor system architecture consists of two independent modules: the graphical modelling environment (left) and the scheduling engine (right).

4.1 The constraint model

The traditional static constraint models are defined by the set of variables, their domains, and by the set of constraints restricting possible combinations of values. As we discussed in Section 3, we need a more dynamic approach to CSP, namely, the variables and the constraints are introduced as search progresses. There exist some static approaches to overcome difficulties with the unknown set of variables/constraints based on dummy variables and deactivated constraints [8,17]. Unfortunately, such approaches lead to huge models so they cannot be used to model the problem completely statically. Nevertheless, we use the dummy variables partially to do look-ahead for planning decisions (via constructive disjunction, see Section 3) and to realise the idea of active decision postponement [11].

The constraint model in the Visopt solver is a piece of code responsible for introduction of variables and constraints. The basic idea is as follows: at the beginning we introduce only the objects that are known, i.e., the customer orders. As these customer orders should be satisfied, we also start dependencies to the resources that can produce the ordered items. When the actual supplier is found (this is usually decided during labelling), we need to find suppliers for this supplier etc. To summarise it: if there is a planning decision, i.e. the decision about what objects should be part of the plan/schedule, we introduce all of them (via dependencies). Together with these objects, the relevant constraints are posted so we can exploit the power of constraint propagation. Let us now describe some details about what variables and what constraints are used.

The slot representation

Opposite to most scheduling systems that use the task-centric model of the problem, we decided to apply the resource-centric model because it simplifies modelling of the complex transition schemes [2]. It means that the batches are grouped per resource rather than per task. Of course, we do not know the batches in the resource at the beginning so we use a chain of empty slots to represent the schedule for each resource. Opposite to the slots used in the timetabling applications, the slots in our

system may slide in time and they may have variable duration. The only restriction is that the ordering of slots must be preserved (due to the transition constraints).

Each slot has some attributes like the start time, the end time, and the duration represented as finite domain variables. Also, there is a special variable describing the type of the batch (the state) that can be filled in the slot. When this state variable becomes a singleton we know the batch in the slot - we say that the slot is filled by the batch. This may introduce other variables that are specific for the particular batch, e.g. quantities of consumed and produced items. Naturally, all the slot variables are connected via constraints describing the time windows etc. and these constraints can be posted even if the batch in the slot is not known yet. Moreover, there can be also constraints between the neighbouring slots to describe the transition scheme.

To model the minimum and the maximum batches per state we introduce a special variable called a serial number that "counts" the batches of the same state. This variable participates in the transition constraints so it may force the state change when the maximum number of batches is reached or it may forbid the state change when the minimum number of batches is not reached. Figure 7 illustrates this mechanism, for technical details see [5]. The same mechanism is used to model the batch counters.

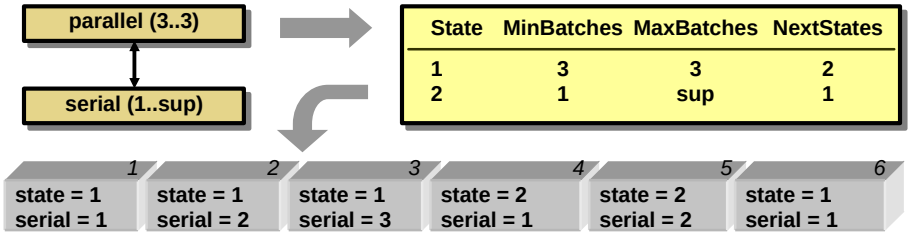

Fig. 7. The transition scheme (top left) is modelled using the serial numbers and the special transition constraints defined over the transition table (top right).

As we mentioned above the slots are also introduced dynamically which saves some memory. In fact, a new slot is attached to the end of the slot list when there is a demand to the resource but there is no free slot to satisfy this demand. Note however, that it does not mean that the new slot will be filled by the coming batch that caused its introduction. Perhaps some waiting (not yet allocated) batch or a future batch overhauls it or the slot will stay empty if we find later that the batch is not necessary. Still, the ordering of slots is fixed so it is not possible to introduce a new slot in-between two existing slots. Thus, deciding to which slot the batch is allocated corresponds to the decision about the absolute ordering of batches in the resource. This view is similar to the idea of permutation based scheduling presented in [21]. The main difference of our approach is that we can solve problems where the appearance of the batch depends on allocation of other batches. In particular, the structure of the batches in the resource depends on the demands from other resources as well as on the transition scheme for the resource.

Notice that although we do not know the batches in the resource, thanks to the slot representation we can post many constraints in advance and thus to use the power of constraint propagation. The main reason for using the slot representation is modelling complex transition schemes.

Dependencies

The slots of different resources are connected via dependencies modelling the supplier-consumer relations. Because the dependency is closely related to the item we cannot introduce the dependency until we know the item and its quantity. As described in the previous section, the variable specifying the item quantity is generated as soon as we know the batch - the state - in the slot. At the same time we can start the dependencies from the given slot.

Assume that we have an input item defined for the batch in the slot. Dependencies should connect this batch with all the supplying batches. It is possible to post the dependency to every slot that can be filled by the supplying batch. However, this eager method has huge memory consumption when applied to large-scale problems with hundreds or thousands of slots. Thus, we use a more lazy method that posts a minimal number of dependencies covering the required quantity. Typically, these dependencies go to the first "free" slot of every possible supplying resource. If we find later that the slot cannot be filled by the supplying batch then we move the dependency to the next slot and so on (see Figure 8). This is realised by setting the quantity in the dependency to zero (so the constraint connecting the slot times "evaporates") and by introducing a new dependency going to the next slot. If no supplying batch is found in the resource then the dependency to the resource is finally made empty and the supplying batch must be found in another resource. Other dependencies can be introduced as soon as we find that the dependencies generated so far are not enough to cover the requested quantity (e.g. because some of them have been made empty).

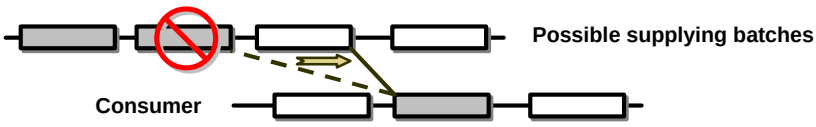

Fig. 8. The dependency generator introduces the dependencies to the first possible slot (from left) of each candidate supplier. If the slot is not dependent (⃠) then the dependency is moved to the next free slot etc.

For each slot, the system maintains the links to all non-empty dependencies going to this slot. These links are used during scheduling for decision about which batch will be filled in the slot (see Section 4.2). Of course, we also know all the dependencies going to a particular resource so we can post special ordering constraints between the dependencies [12]. Because, every dependency defines a demand for one batch, these constraints decide about the ordering of the batches in the resource. Note finally that these global constraints must be open to accept incoming batches [4].

4.2 The search strategy

The Visopt constraint model is responsible for introduction of all variables and all constraints. The scheduling strategy then decides about the batches in the slots and about the connections between the slots. Recall that the slots are introduced from left

to right and the dependencies are started first from the orders. Thus, the labelling procedure must be aware of this ordering.

In the Visopt scheduling engine, we are closing the slots (decide about the batch types in the slots) in slices going from left (past) to right (future). We call processing the slice a *scheduling step*. The decision whether the slot belongs to the slice or not is done using the time variables in the slot. In the slice, the slots are closed in the order-to-purchase ordering.

When the slot is selected, the second problem is which batch should be filled in the slot. This decision is naturally based on the dependencies going to the slot. The labelling strategy first selects the "best" dependencies and then it connects them to the slot. This is done via setting the quantity variable in the dependency to be greater than zero (the maximum value is tried first). The relation "better" between the dependencies is defined using the time of the dependency (the earliest time is preferred), using the cost information (the smallest cost is preferred), and using other heuristic criteria. Recall that when some dependencies are fixed to the slot, the incompatible dependencies are moved automatically to the next possible slot (see Figure 9). Thus, the decision about the ordering of dependencies is equivalent to the decision about the ordering of batches in the slots. After selecting the dependencies in the slot, the labelling strategy assigns a value to the state variable. In the end of each scheduling step, the time variables in the closed slots are labelled - the earlier times are preferred.

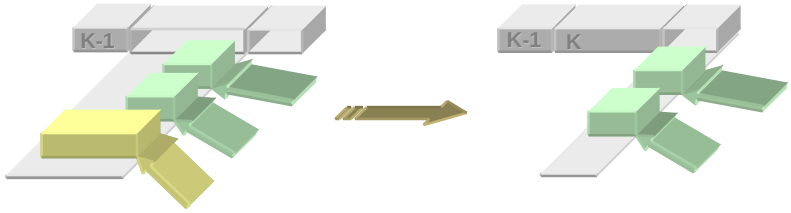

Fig. 9. The basic decision of the scheduling strategy is which dependencies will go to a particular slot, i.e., which batch will be filled in the slot (left). Then the incompatible dependencies are moved automatically to the next slot (right).

As described in the above paragraphs, the scheduling strategy is based on depth-first search (backtracking). Because the variable ordering is selected carefully, the scheduling strategy knows nothing about the dynamic character of the constraint model. Every time the labelling procedure attempts to assign a value to the variable, the variable is present in the system. Still, notice that the structure of the variables is different in different branches of the search tree (because different dependencies are introduced). This dynamic character complicates the usage of more advanced search techniques like the limited discrepancy search.

So far we have enhanced the base backtracking mechanism by *user defined backjumping*. We have developed this new search algorithm using the following idea. If we cannot satisfy the demand in a given scheduling step then we leave this demand open (no batches for the demand are closed) and we try to satisfy it in the next step. This is realised by jumping to a pre-selected variable after failure, unassigning this variable and continuing with labelling of the other variables. This unassigned variable will be tried in the next scheduling step again. This algorithm is similar to graph-

directed backjumping, the main difference is that the back jump is realised only for some pre-selected variables.

5 The results

The Visopt ShopFloor scheduling engine is completely implemented in SICStus Prolog (currently we use the version 3.8.7). It has been tested in several pilot projects in one of the biggest chemical enterprises in Europe, in one of the biggest and famous candy producers in The Netherlands, and in one of the biggest dairies in Israel among others. We are not aware of any other scheduling system that can model and solve the problems described in this paper, so we cannot compare our solver to existing schedulers. In this section we first show the plans produced by our solver for the problem from Section 2.3 and then we summarise the results of some real-life models.

The problem from Section 2.3 requires both planning, i.e., deciding which batches are necessary to satisfy the demands, and scheduling, i.e., allocating the batches to available resources. Recall, that there are three different ways of producing the final item, namely parallel production, serial production, and recycling. Moreover, there is a complex transition scheme describing the resource including insertion of a cleaning batch after a specified number of the production batches. Last but not least, there is a worker that influences the timing of the parallel production.

Figure 10 shows a Gantt chart of the plan produced by our solver (3 seconds on 1.7 GHz Mobile Pentium 4). We can see that this plan satisfies all the production rules, in particular using the recycling and the cleaning batches. Also the duration of the parallel batches decreases when the worker became experienced (roughly at time 35).

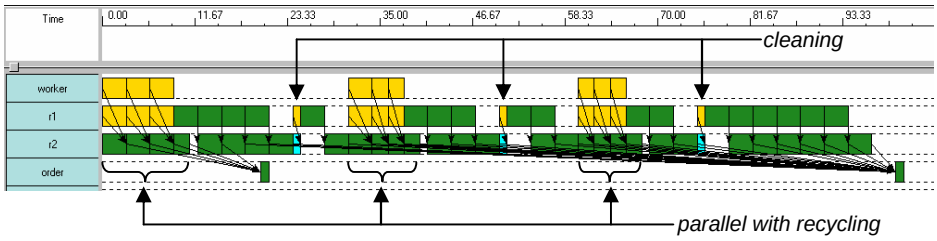

Fig. 10. The Gantt chart of the plan for the problem from Section 2.3

We have relaxed the restriction about parallel cleaning for both machines to see if the production can be more efficient. Figure 11 shows that the resulting plan is shorter because the cleaning batches can be scheduled asynchronously (planning took 3 seconds on 1.7 GHz Mobile Pentium 4).

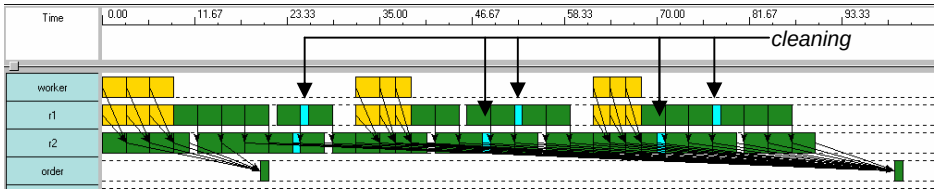

Fig. 11. The Gantt chart of the plan for the problem from Section 2.3 where the cleaning is asynchronous.

To show the size and the complexity of the real-life problems we have prepared a summary of some pilot problems solved by the Visopt ShopFloor system. These test problems are based on the real-life production lines so the actual models and the plans are confidential. Thus we can present only some global parameters of the models. In particular, Table 1 shows the model size and the runtime results. For each model we include the number of resources, the total number of states in the resources, the number of orders together with the ordered quantity, the number of items, and duration of the scheduled period. The ordered quantity corresponds roughly to the size of domains of the quantity variables - we track every quantity unit in the production. The schedule duration corresponds to the size of the domains for the time variables - it shows the resolution of scheduling (e.g. 10.080 time units means a one week production with a minute resolution). The solution is characterised by the runtime and by the solution size measured in the number of batches and in the number of dependencies.

Table 1. Model and solution size for some test problems. Runtime is measured in seconds on a Mobile Pentium 4 1.7 GHz.

	model					solution		
	res.	states	orders # / quantity	items	duration time units	runtime sec.	batches	dependencies
1	19	334	1 / 144000	47	10080	105	1000	1441
2	28	115	1 / 50	34	8640	79	256	310
3	22	677	9 / 7600	56	3168	40	651	898
4	57	704	256 /196748	45	840	77	990	1428
5	34	574	45 / 88485	294	11520	2339	5807	10175

For comparison, the state of the art schedulers handle about 20.000 batches [personal communication to Wim Nuijten from ILOG] but all these batches are known in advance. In planning, the size of plans is measured in tens of actions [13]. As Table 2 shows we can handle problems with hundreds to thousands batches. However, when the number of batches increases, the large memory consumption becomes a limitation. Thus in the model 5 there is a trade off between the memory consumption and the runtime. This observation confirms our claim that such problems cannot be handled in a fully static way using the dummy activities because then the memory consumption becomes critical.

To summarise the results, we can handle problems much larger than the traditional planning problems and close to the size of the problems in conventional static

scheduling. Recall that the input to the Visopt engine consists of the model of the factory and the list of demands. All the batches are introduced (planned) during the problem solving and allocated to the resources (scheduling). Thus, we are basically solving a (limited) planning problem under time and resource constraints. Moreover, our system can handle more complex resource constraints (a transition scheme) and resource dependencies (recycling, many-to-many relations etc.) than the conventional schedulers can.

6 Conclusion

In this paper we described the heart of the Visopt ShopFloor system - the scheduling engine. The integrated planning component is the main difference of our system from the conventional schedulers. We are not aware of any other system doing such deep integration so it is hard to compare Visopt to existing systems. As we showed in Section 2, the planning component provides flexibility that cannot be reached by conventional scheduling software. The unique features of Visopt, which the other scheduling systems cannot cover, include modelling of complex transition schemes for resources, modelling of an 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 [22] provides intuitive graphical modelling environment independent of the solver, generality covering many scheduling problems, and extendibility via adding new types of resources.

Acknowledgements

The research is supported by the Grant Agency of the Czech Republic under the contract 201/01/0942 and by Visopt B.V. I would like to thank the reviewers of the paper for useful comments and to Petr Štepánek and Ondrej Cepek for proofreading.

References

1. Baptiste, P. and Le Pape, C.: Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling, in *Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group* (1996).
2. Barták, R.: Conceptual Models for Combined Planning and Scheduling. *Electronic Notes in Discrete Mathematics*, Volume 4, Elsevier (1999).

3. Barták, R.: Dynamic Constraint Models for Planning and Scheduling Problems. *Proceedings of the ERCIM/CompulogNet Workshop on Constraint Programming*, LNAI Series, Springer Verlag (2000).
4. Barták, R.: Dynamic Global Constraints in Backtracking Based Environments, in *Annals of Operations Research 118*, Kluwer (2003) 101-119. Forthcoming.
5. Barták, R.: 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 (2002) 186-194.
6. Barták, R.: 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, (2002) 587-602.
7. Brucker P. *Scheduling Algorithms*. Springer Verlag (2001).
8. Beck, J.Ch. and Fox, M.S.: Scheduling Alternative Activities. *Proceedings of AAAI-99*, USA (1999) 680-687.
9. Brusoni, V., Console, L., Lamma. E., Mello, P., Milano, M., Terenziani, P.: Resource-based vs. Task-based Approaches for Scheduling Problems. *Proceedings of the 9th ISMIS96*, LNCS Series, Springer Verlag (1996).
10. Gallaire, H.: Logic Programming: Further Developments, in: IEEE Symposium on Logic Programming, Boston, IEEE (1985).
11. Joslin, D. and Pollack M.E.: Passive and Active Decision Postponement in Plan Generation. *Proceedings of the Third European Conference on Planning* (1995).
12. Laborie P.: Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results. In *Proceedings of 6th European Conference on Planning*, Toledo, Spain (2001), 205-216.
13. Long D. and Fox. M. International Planning Competition 2002. Toulouse, France (2002). http://www.dur.ac.uk/d.p.long/competition.html
14. Mittal, S. and Falkenhainer, B.: Dynamic Constraint Satisfaction Problems. *Proceedings of AAAI-90*, USA (1990), 25-32.
15. Nareyek, A.: Structural Constraint Satisfaction. *Proceedings of AAAI-99 Workshop on Configuration* (1999).
16. Nareyek, A.: AI Planning in a Constraint Programming Framework. *Proceedings of the Third International Workshop on Communication-Based Systems* (2000).
17. Pegman, M.: Short Term Liquid Metal Scheduling. *Proceedings of PAPPACT98 Conference*, London (1998), 91-99.
18. Srivastava B. and Kambhampati S.: Scaling up Planning by teasing out Resource Scheduling. Technical Report ASU CSE TR 99-005, Arizona State University (1999).
19. Van Hentenryck, P.: *Constraint Satisfaction in Logic Programming*, The MIT Press, Cambridge, Mass. (1989).
20. Wallace, M.: Applying Constraints for Scheduling, in: *Constraint Programming*, Mayoh B. and Penjaak J. (eds.), NATO ASI Series, Springer Verlag (1994).
21. Zhou, J.: A Permutation-Based Approach for Solving the Job-Shop Problem. *Constraints*, vol. 2 no. 2 (1997), 185-213.
22. Visopt B.V. http://www.visopt.com