Visopt ShopFloor: On the edge of planning and scheduling

Roman Barták*

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

Abstract. Visopt ShopFloor is a complete system for solving real-life scheduling problems in complex industries. In particular, the system is intended to problem areas where traditional scheduling methods failed. In the paper we describe the heart of the Visopt system, a generic scheduling engine. This engine goes beyond traditional scheduling by offering some planning capabilities. We achieved this integrated behaviour by applying Constraint Logic Programming in a less standard way - the definition of a constraint model is dynamic and introduction of constraints interleaves with search.

1 Introduction

Scheduling is one of the strongest application areas of constraint programming [15]. The reason can be found in a similar static character of both scheduling and constraint programming algorithms. First, the problem structure is fully described: in case of scheduling it means that all the activities and relations among them are known, in case of standard constraint satisfaction problem (CSP): all the variables, their domains, and constraints are known. The problem solving then consists merely of searching the space of alternative combinations of activity positions or variables' values. Search is also a core technology in the similar area of planning. However, the main difference from scheduling is the dynamic character of planning [12]. The structure of the plan is so variable that it can be hardly encoded in variables, domains, and constraints defined before the problem solving starts. Thus, planning typically uses ad-hoc algorithms even if there exist approaches based on general concepts like CSP. These approaches either try to fit planning problem into a static concept of CSP [5,13] or they are based on generalisations of CSP framework like Dynamic CSP [10] or Structured CSP [11].

To solve problems on the edge of planning and scheduling we propose to use existing technology of Constraint Logic Programming (CLP) in the way this framework was originally defined [7]. Note that since CSP has been formalised, the same static approach is applied to solve problems in CLP, i.e., define the variables,

* The research is supported by the Grant Agency of the Czech Republic under the contracts no. 201/99/D057 and 201/01/0942.

domains, and constraints first and then do search [14]. In our opinion this is not a natural way of solving problems in (constraint) logic programming framework. Moreover, it leads to such strange formulations like calling ILOG Solver "a C++ implementation of constraint logic programming" (C++ is no more logic programming). Our idea is to return to the roots, i.e., to use constraints in CLP in the same way as unification is used [7]. It means that constraints are introduced during search and thus in different branches of search tree different sets of constraints are active. The usual argument against this approach is weak constraint propagation. This is not a fully fair argument because different branches in search tree represent disjunction and it is a known wisdom that propagation through disjunction is not good as well. Moreover, we can post the constraints in advance in CLP as well, if we know them. Our proposal is to use CLP capabilities to reduce exponential grow of the number of constraints in the planning-like problems.

We believe that CLP approach is sufficient to model and solve mixed planning and scheduling problems. To demonstrate this attitude we have implemented a generic scheduling engine for Visopt ShopFloor system. The main difference of our solver from other scheduling systems is full integration of a planning component, i.e., the solver does both planning and scheduling task. In fact, there is no strict border between planning and scheduling in the Visopt system

In the paper we describe the motivation behind our system (Section 2), we present a general modelling framework for description of real-life scheduling problems (Section 3) and we show how to represent this model using constraints (Section 4). Finally, we sketch the basic scheduling strategy (Section 5) and we present some results of the system (Section 6). This paper should be read more as a summary of our research in the area of integrated planning and scheduling, the reader is kindly asked to follow the references for more details.

2 Problem area

Visopt ShopFloor is not another academic planner/scheduler but its design has been driven completely by a commercial area, i.e. by existing problems in real-live factories. The emphasis has been put to complex areas where the traditional scheduling techniques are not able to cover whole complexity of the problem. In particular, we concentrate on scheduling problems in food, chemical, and pharmaceutical industrial. Nevertheless, one the original goals was to design a generic scheduling engine applicable to other areas as well. So, from the beginning we accommodate the engine by rich modelling interface for problem description.

Basically, the goal is to generate a plan/schedule of production for a given time period. It means that notions like makespan are not used directly there (we speak about optimisation later in this section). A typical feature of our problem area is using *resources (machines) with complex behaviour*. Currently, we concentrate on batch production but we can model continuous production using batches as well (simply by decomposition of the continuous process into batches). Thus the schedule for the resource consists of a sequence of non-overlapping batches. There are several types of batches per resource; sometimes there is only one batch type per resource, sometimes

the number of types per resource is rather large (tens to hundreds). The user defines a transition scheme between the batch types as well as a minimal and maximal number of batches of the same type in the continuous sequence [4]. There could be also some additional constraints on sequencing, for example after a given number of batches or after a given period of time a cleaning or maintenance batch must be inserted. Thus a transition scheme puts strong restrictions on sequencing of batches in the resource. Modelling complex transition schemes is one of the unique features of Visopt.

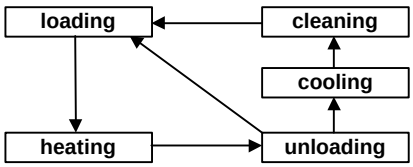

Fig. 1. A simple transition scheme for the resource.

The location of batch can be further restricted by using time windows, i.e., the batch must start and complete in specified time intervals. If the batch is non-interruptible then it must run completely within some time window. We also allow interruptible batches, i.e. the batch may start in one time window and complete in another time window. However, the batch stays in the resource from the beginning to the end so even if the batch is in idle time no other batch can be processed by the same resource. The user may specify a maximal ratio between the process and idle time to prevent the batch to be interrupted for a long time, e.g. weekend.

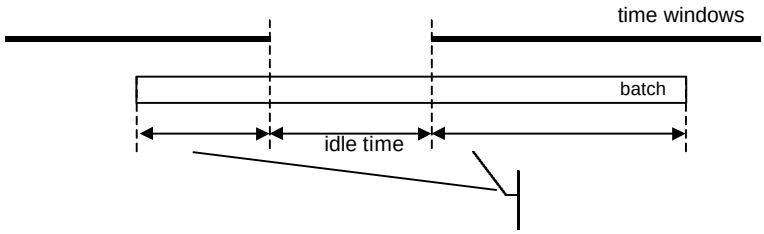

Fig. 2. Interruptible batch occupies the resource out of time windows too.

Connection between resources is described via items. Batches consume and produce some items and using these items the user may describe supplier-consumer relations between the resources. We call these relations dependencies. Again, the *supplier-consumer scheme* is completely general so the user may describe a real structure of the factory including many-to-many relations between the resources or even (re-)cycling. Note that the dependencies describe moving of the items between the resources, so it is possible to assign delay between releasing and consuming the item. Moreover, there could be alternative recipes how to produce the item, so the batch typically have more input items and there could be multiple output items as well, for example by-products that can be used to produce other items. Together, one batch may be connected to several supplying and consuming batches that can be spread over several resources. A general dependency scheme is another unique feature of Visopt.

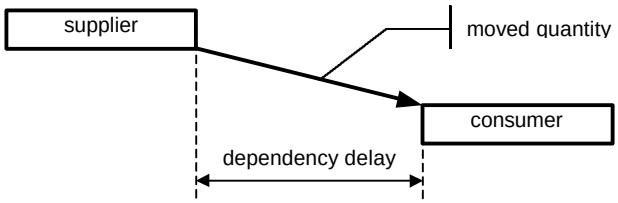

Fig. 3. Batches are connected via dependencies.

Basically, the *production is driven by orders.* Some of the orders are real orders from customers and some of the orders are forecast orders. The orders may have a strict delivery time or they have just recommended delivery time so they can be delayed (typically, the forecast orders can be delayed for free). In the order the user specifies the ordered items and requested quantities, it is also possible to describe alternative items that can be delivered if the ordered item is not available. The ordered quantity can also be relaxed so we can deliver more or less than the user requested (if the user allows it). A penalty mechanism is used to describe how the delivery corresponds to the user requirements - the penalty can be put to time, to alternative items, and to quantities.

As we already mentioned, the basic requirement is to generate a feasible schedule for a fixed period of time. Some users prefer to minimise the number of set-ups, others like to minimise storing time or to maximise satisfaction of customers and quite often *multi-criteria optimisation* is required. Our opinion about optimisation in scheduling is that all the optimisation criteria substitute what the user really needs - to maximise the profit[1]. If the set-ups are expensive then the user asks to minimise the number of set-ups. But what he or she really wants is to minimise the cost of production. The same holds about minimisation of makespan. If the production is faster then the user can satisfy more customers and thus to earn more money. Consequently, the user defines costs and penalties for the objects in the schedule and he or she requires minimising the sum of all costs in the schedule. Finally, the users are typically not looking for an optimal solution; they need a good enough solution. They can describe the good enough solution by putting restrictions about the cost.

3 Problem description

In [1] we analyse the basic scheduling models in detail and in [6] a similar analysis can be found so we put only a summary here. There exists three basic ways how to describe the scheduling problems: resource-centric, task-centric, and timetabling model. In the timetabling model, the time is divided into slots that are filled by activities during scheduling. The number of slots corresponds to the resolution of the schedule so this model is less practical for generic scheduling problems due to large number of slots (see Section 6 for the numbers). Usually, the scheduling systems use

[1] Another criterion asked by the customers is schedule robustness, i.e., small sensitivity to flaws during production. It can be modelled using penalties or more tighten constraints; small flaws are solved at production level, e.g., using a frozen flavour if fresh one is not available.

activities as the basic objects and they allocate the activities to resources over time. In a typical scheduling system, the activities are grouped per task, in our terminology it means that we have a set of batches necessary to satisfy the order. This so-called task centric model has the advantage that the dependencies or precedence relations between the batches can be expressed directly using the constraints. However, there are two major difficulties of this model when it is applied to the problem area that we described above.

First, if there are many alternative production routes how to satisfy the order then it is hard to describe the task as a simple set of batches. Note that the alternative production routes mean not only using alternative resources but also using possibly very different structure of batches including sharing batches by several tasks. There exist approaches based on deactivated (dummy) batches [5,13] but they are less efficient or even inapplicable if the number of dummy batches is large.

The second problem is that the resources in our problem area have typically very complex behaviour with a user defined transition scheme. It is hard to define transition constraints if we do not know to which resource the batch is allocated.

Because of above features of timetabling and task-centric models we decided to use a resource-centric model in the Visopt scheduling engine. In this model, we concentrate on description of resources and connections (dependencies) between the resources. The structure of the task is decided during scheduling, i.e., the planning component is included there. Each *resource* is specified by the list of batch types that the resource can process and by a transition scheme between the batch types. One batch type can be selected as the type of the first batch in the resource (start-up or the batch type that is running at the schedule start). For each batch type, we describe its duration, working time, input and output items with quantities and other parameters. Thus, the user describes the resources in a natural way.

The relations between the resources are described via *dependencies*. In each dependency we specify the item, the supplying resource and the type of the supplying batch as well as the consuming resource and the type of the consuming batch. Of course, the dependency encapsulates other parameters like delay between the end of the supplying batch and the start of the consuming batch. Because the dependency describes the connection between two resources, we can describe arbitrary structure of the factory including re-cycling routes etc. Finally, *orders* are assumed to be a special type of the consumer, so the dependencies also describe which items can be delivered (ordered) and what are the suppliers of these items.

We call a description of resources, dependencies, and orders a *factory model*. Notice that with the exception of the initial batches per resource, there are no batches specified in the factory model. This is the main difference of our system from traditional schedulers. In the Visopt scheduler, the batches are introduced during scheduling according to the actual set of orders which gives us the planning capabilities. On one side, we have the advantage of generality of the factory model, on the other side we have to solve more complicated problems because of the planning component.

Factory model describes fully the problem and it forms the interface between the Visopt ShopFloor user interface and the scheduling engine. The user can build the factory model using the intuitive graphical user interface or the model is generated automatically from the ERP database.

4 Constraint Representation

The factory model describes just the parameters of resources and dependencies, so it is fully declarative. The constraints are hidden in the semantics of the model. In this section we describe how the model is realised in terms of constraint satisfaction, i.e. using domain variables and constraints.

There exist approaches trying to represent dynamic problems in a static way using dummy variables [5,13]. The difficulty here is that the ratio between total number of variables (including dummies) and the variables participating in the solution is very large. So such fully static representation is inefficient and it is even not realisable in large-scale problems. Thus we decided to use a representation where some variables and constraints are introduced dynamically during search [2]. Because we still want to exploit the power of constraint propagation, we are introducing the variables and constraints as soon as possible. In some sense, we follow the idea of active decision postponement [8]. This is realised using the generalised notion of slot from timetabling. In our framework, the slot is a shell to be filled by a batch of a particular type. Slot is assigned to a resource but it can slide in time. In practice, the (partial) schedule for each resource is represented by a list of slots. In each slot, there is a finite domain (FD) variable describing which batch type can be filled into the slot. If this variable is singleton then we know which batch type is filled in the slot. Moreover, there are two more FD variables in each slot describing the start time and the end time of the batch in the slot. These variables are connected to the batch type variable using a constraint describing the time windows. Batch type variables and end time and start time variables from successive slots are connected via a transition constraint describing the transition scheme. To describe the interruptible batches we use a process duration variable in the slot (in case of non-interruptible batches process_duration = end_time - start_time, otherwise process duration describes the time spent in time windows). To describe minimal and maximal number of batches of the same type we use a serial number variable indicating the position of the batch in the sequence of batches of the same type. These variables may further restrict the transition constraint (details can be found in [4]). Basically speaking, the slot variables describe the common variables of all batch types in the resource. Consequently, we can post constraints among them and use constraint propagation.

There are two ways how to introduce slots to the system. First, it is possible to estimate a maximal number of slots using the transition scheme, duration of batches etc. and to generate this number of slots in advance. The disadvantage of this approach is that it may introduce a huge number of dummy slots that will not be used in the final schedule. Therefore we generate the slots dynamically on demand. It means that if we find that some batch could be allocated to the resource then we generate a slot for it. So slots are added due to a transition scheme (restricted transition time) or due to a demand from other resources (asking for supplier/consumer). Note also that even if we introduce the slot it does not mean that it will be filled by the batch that caused this introduction. Perhaps some other batch overhauls it. The ordering of slots is fixed so it is not possible to introduce a new slot in-between two existing slots (because the transition constraints has already been posted). Thus, deciding to which slot the batch is allocated corresponds to the decision about ordering of batches in the resource. There is another difference

between slots and batches. The batch also describes input and output quantities of processed items so for each item there is a FD variable describing its quantity. If the number of items is large, it is not efficient to include such variables into the slot. Thus, these variables (and corresponding constraints, e.g., capacity limit) will be introduced dynamically when the batch type in the slot is known. In our constraint model, such introduction is done automatically using event-driven programming.

Till now, we described the constraint model for a single resource. The resources, or more precisely the batches in different resources, are connected via dependencies that typically describe the supplier-consumer relation. These dependencies are introduced dynamically as the slots are being filled by the batches. In particular, when the slot is filled by a batch then the variables describing produced and consumed quantities are introduced. If these quantities are non-zero then we have to find suppliers and consumers respectively. This is realised by posting dependencies to resources that can supply or consume the item (these resources are defined in the factory model). Such dependency is specified by a FD variable for moved quantity, FD variables for the end time of the supplier and the start time of the consumer, and constraints binding these variables. The resource, which is accepting the dependency, maintains links to the dependency in every slot that can be filled by a batch serving as a supplier or consumer respectively. Maintaining the links means that if the slot is moved out of the dependency time or the slot is filled by a batch type incompatible with the dependency then the link is removed. Naturally, such links asking for a batch in the slot are used when deciding about which batch type will be filled in the slot.

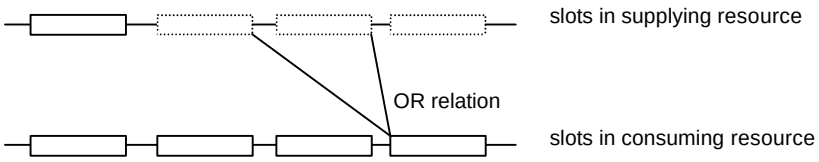

Fig. 4. Slots that can be filled by a supplying batch are linked to a dependency (dashed line indicates the slots where the batch is not filled yet)

5 Labelling strategy

The constraint model is dynamic but autonomous. It means that variables and constraints are introduced automatically when the system finds out that they are necessary. Thus, if the labelling procedure follows some rules about the ordering of variables, then it knows nothing about the dynamic character of the constraint model. For example, the batch type variable in the slot should be labelled before we can label the variables describing item quantities in the batch.

We use standard backtracking-based search driven by heuristic and enhanced by some backjumping features. At the beginning, there are just orders in the system, dependencies going from the orders to some resources, and some (usually empty) slots in the resources. The scheduling strategy works in steps going from left to right

(past to future), the size of the step is defined by the user. The step is represented by a border line called frontier that is moved from left to right. At each step, only the batches (slots) that start before the frontier are scheduled. The consequence of this strategy is that only required production is scheduled.

The labelling within the step goes in the order-to-purchase direction, i.e., the slots in the resources that supply directly to the orders are being closed first, then the slots in the resources supplying these suppliers of orders and so on. In the resource the slots are being filled from left to right. For each slot, we first explore the dependencies going to the slot. Some of these dependencies are selected and thus connected to this slot (earlier dependencies are tried first). This decision further constraints the batch type in the slot so usually the batch type variable becomes singleton. Otherwise, this variable is labelled. As soon as the batch type is known, the labelling proceeds to the variables describing the item quantities in the batch (these variables have been introduced when the batch type variable becomes singleton). At the end of each labelling step, the time variables are labelled in the closed slots (earlier time is preferred).

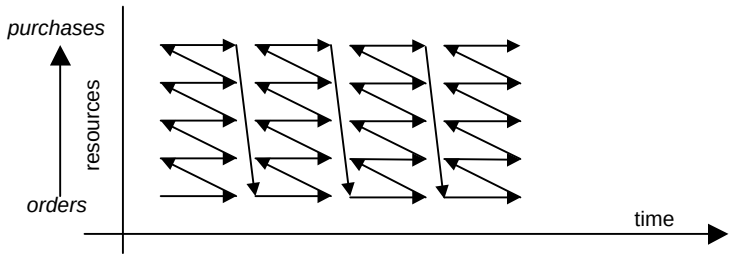

Fig. 5. Ordering used in the labelling strategy.

The key decision in the above labelling procedure is about which dependencies will go to a given slot. In fact this decision is about ordering of batches in the resources. In the standard labelling procedure, the earliest dependencies are preferred. For optimisation, we use a special variant of the labelling procedure where the decisions are done according to the cost. It means that the values are tentatively assigned and the value leading to the smallest cost is selected.

6 Results and conclusions

Visopt ShopFloor system has been tested in several pilot projects in one of the biggest chemical enterprises in Europe, one of the biggest and famous candy producers in The Netherlands, and one of the biggest dairies in Israel among others. The feedback from the companies is very positive - Visopt ShopFloor is the only system, among the systems that they tested, that can fully cover the complexity of production in these enterprises via offering rich modelling capabilities supported by a new solving technology. The unique features of Visopt, that 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.

In the following tables we summarise the numerical results of one of the pilot projects where the goal is to generate a detail schedule for a week production.

Number of resources	34
Number of batch types	991
Size of scheduled period	1 week (10 080 minutes)
Time resolution	1 minute
Number of items	294
Number of orders	45
Total quantity in orders	88.5 tons (88 485 kg)
Quantity resolution	1 kilogram

Table 1. Problem size

The size of the factory model describing fully the problem is almost 1.4 MB. The structure of the problem consists of several groups of alternative resources including secondary resources (workers) and it contains both production and packing of the final items. The resources in the groups of alternative resources are not fully identical so the number of alternative production routes is very large. The item dependent set-up times are included in most production resources and time windows are defined for all the resources. The resolution of scheduling is another interesting point there: we are scheduling one week production with one minute resolution, i.e., over ten thousand time units, and over 88 tons of ordered items with one kilogram resolution.

Number of batches	5938
Number of dependencies	9496
Runtime	65 minutes (Pentium 4/1700 MHz)

Table 2. Solution size

For comparison, the traditional 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 [AIPS02 planning competition]. 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 resources (scheduling). Thus, we are basically solving a (limited) planning problem under time and resource constraints. As Table 2 shows, we can generate plans in the size close to the size of traditional scheduling problems but with a more complex and tied structure.

Design of the scheduling engine for Visopt brings some novelty problems and shows some solutions both to constraint programming and to scheduling. Despite the original disbelief of constraint community we showed that dynamic models are applicable to solving real-life large-scale problems. Our concept of dynamic global constraints [3] is applicable to other areas as well, especially when constraint logic programming is used and when introduction of constraints interleaves with search. For the scheduling area, we showed that resource-centric representation is superior when the number of alternative production routes is huge and the resource description is complex including transition times etc. On the other hand, the memory

consumption and weak constraint propagation through production routes are basic weaknesses of this approach so our next steps include integration of resource-centric and task-centric representations. We also concentrate on design of new filtering algorithms for integrated planning and scheduling. Traditional scheduling constraints like edge-finders does not work there because the domains of time variables are not tighten so edge finding cannot deduce information about batch ordering [9]. In Visopt, we use a new group of constraints based on batch ordering propagated through the resources (details are out of scope of this paper). Finally, we are working on improving the stability of the scheduling engine using advanced search strategies like Limited Discrepancy Search that could escape from search sub-trees where backtracking is trapped.

References

1. Barták, R.: Conceptual Models for Combined Planning and Scheduling. *Electronic Notes in Discrete Mathematics*, Volume 4, Elsevier (1999).
2. Barták, R.: Dynamic Constraint Models for Planning and Scheduling Problems. *Proceedings of the ERCIM/CompulogNet Workshop on Constraint Programming*, LNAI Series, Springer Verlag (2000).
3. Barták, R.: Dynamic Global Constraints in Backtracking Based Environments, in *Annals of Operations Research*, Kluwer (2002), to appear.
4. Barták, R.: Modelling Transition Constraints, submitted to ECAI 2002 Workshop on Modelling and Solving Problems with Constraints (2002).
5. Beck, J.Ch. and Fox, M.S.: Scheduling Alternative Activities. *Proceedings of AAAI-99*, USA (1999) 680-687.
6. 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).
7. Gallaire, H.: Logic Programming: Further Developments, in: IEEE Symposium on Logic Programming, Boston, IEEE (1985).
8. Joslin, D. and Pollack M.E.: Passive and Active Decision Postponement in Plan Generation. *Proceedings of the Third European Conference on Planning* (1995)
9. 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.
10. Mittal, S. and Falkenhainer, B.: Dynamic Constraint Satisfaction Problems. *Proceedings of AAAI-90*, USA (1990), 25-32.
11. Nareyek, A.: Structural Constraint Satisfaction. *Proceedings of AAAI-99 Workshop on Configuration* (1999).
12. Nareyek, A.: AI Planning in a Constraint Programming Framework. *Proceedings of the Third International Workshop on Communication-Based Systems* (2000), to appear.
13. Pegman, M.: Short Term Liquid Metal Scheduling. *Proceedings of PAPPACT98 Conference*, London (1998), 91-99.
14. van Hentenryck, P.: *Constraint Satisfaction in Logic Programming*, The MIT Press, Cambridge, Mass. (1989).
15. Wallace, M.: Applying Constraints for Scheduling, in: *Constraint Programming*, Mayoh B. and Penjaak J. (eds.), NATO ASI Series, Springer Verlag (1994).