Constraint-Based Scheduling: An Introduction for Newcomers

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. Constraint-based scheduling is an approach for solving real-life scheduling problems by stating constraints over the problem variables. By providing generic constraint satisfaction techniques on one side and specialised constraints on the other side, constraint programming achieves a very good generality and efficiency and thus it becomes very popular in solving real-life combinatorial (optimisation) problems. In this paper we present some constraint satisfaction techniques used in constraint-based scheduling. Our goal is to introduce the technology to newcomers rather than to provide a deep survey of the area or to describe some new results there.

Introduction

Automated scheduling is a long-time studied subject in computer science, especially in operations research, and many fast scheduling algorithms for various problem classes have been proposed there [5]. The difficulty of this academic view of scheduling is that the problems like job-shop scheduling do not exist in reality. In real-life scheduling problems, neither the structure of the resources nor the structure of the tasks is homogenous and many side constraints must be assumed to model the problem [4]. Constraint programming provides technology to model and solve such real-life problems.

Scheduling problems arise in situations where a set of activities has to be processed by a limited number of resources in a limited amount of time. In general, the scheduling problem consists of resource allocation, i.e., assigning resources to activities, and resource scheduling, i.e. ordering of activities at each resource. Sometimes, a planning component is necessary to decide what activities should be scheduled [4].

Activity is a core object of every scheduling problem. It requires some resource(s) for processing and some time of processing. The position of activity in time can be restricted by specifying the earliest start time (release time) and/or the latest end time (deadline). In general, it is possible to describe time windows for processing the

[*] Supported by the Grant Agency of the Czech Republic under the contracts 201/99/D057 and 201/01/0942.

activity. Activities can also depend each on another, e.g., a given activity must be processed before another activity. Other relations among the activities are imposed by the resources to which the activities are allocated. Some resources can process just one activity at given time - they are called unary resources. In other resources, the number of activities processed at given time is restricted by the capacity of the resource - we are speaking about general discrete resources or cumulative resources. Sometimes the activities must form batches in the resource, i.e., the parallel activities start and complete at the same times. The ordering of activities in the resource may be restricted by a special transition scheme with sequence dependent set-up times inserted between the activities [12]. Other resources, called reservoirs, can be both consumed and produced by the activities [8].

The scheduling task is to allocate activities to available resources and to time respecting all the constraints. Usually some objective function defines quality of the schedule so the goal is to minimise makespan (the end time of the latest activity), or to minimise tardiness (the lateness of the activity according to specified time) etc.

Opposite to “academic” scheduling problems [5], the real-life problems consist of resources of several types with connections between the resources defined by the factory structure [4,13]. The resources are quite often unique so even alternative resources provide different capabilities for processing the activities and there are many side constraints. Also the objective function is usually more complex, typically the best profit is required. Such problems can be naturally described in terms of constraint satisfaction.

In the paper we first describe the constraint satisfaction technology in general. Then we show how constraints can be applied to model scheduling problems. Finally, we present some special filtering algorithms and search strategies designed for scheduling problems.

Constraint satisfaction at glance

Constraint programming (CP) is a framework for solving combinatorial (optimisation) problems. The basic idea is to model the problem as a set variables with domains (the values for the variables) and a set of constraints restricting the possible combinations of the variables' values (Figure 1). Usually, the domains are finite and we are speaking about constraint satisfaction problems (CSP). The task is to find a valuation of the variables satisfying all the constraints, i.e., a feasible valuation. Sometimes, there is also an objective function defined over the problem variables. Then the task is to find a feasible valuation minimising or maximising the objective function. Such problems are called constraint satisfaction optimisation problems (CSOP).

Note that modelling problems using CS(O)P is very natural because the constraints can capture arbitrary relations. Opposite to frameworks like linear and integer programming, the constraints are not restricted to linear equalities and inequalities. The constraint can express arbitrary mathematical or logical formula, like ($x^2<y$ *or* $x=y$). It could even be an arbitrary relation that can hardly be expressed in an intentional form and a table is used to described feasible tuples [3]. Moreover the

constraints can bind variables with different even non-numerical domains, e.g. to restrict the length of the string by a natural number.

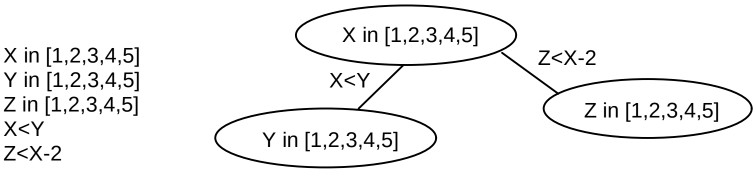

Fig. 1. CSP consists of variables, their domains, and constraints. It can be represented as a constraint (hyper) graph.

Constraint satisfaction technology must take in account the above described generality of problem specification. Usually, a combination of search (enumeration) with constraint propagation is used; some other techniques, e.g., local search, can also be applied to solve problems with constraints. Even if many researchers outside CP put equality between constraint satisfaction and simple enumeration, the reality is that the core technology of CP is hidden in constraint propagation combined with sophisticated search techniques. Constraint propagation is based on idea of using constraints actively to prune the search space. Each constraint has assigned a filtering algorithm that can reduce domains of variables involved in the constraint by removing the values that cannot take part in any feasible solution. This algorithm is evoked every time a domain of some variable in the constraint is changed and this change is propagated to domains of the other variables and so no (Figure 2). Hence the technique is called constraint propagation.

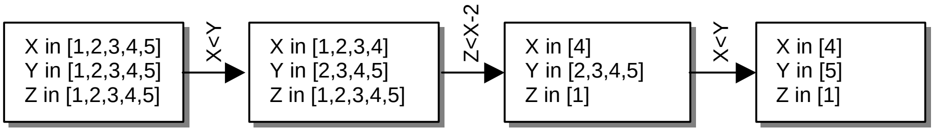

Fig. 2. Constraint propagation does domain reduction by repeated evoking of filtering algorithms until a fix-point is reached.

Notice that each constraint may have its own filtering algorithm so there is no difficulty to solve problems with very different constraints. The generic constraint propagation algorithm known under the notion of *arc consistency* takes care about the correct combination of the local filtering algorithms. On the other hand, this local view of the problem has the disadvantage of incomplete domain reduction. It means that some infeasible values may still sit in the domains of the variables and thus search (with backtracking) is necessary to find a complete feasible valuation of the variables. To reduce deficiency of local propagation, it is possible to group several constraints and see this group as a special constraint called global constraint. Instead of using local propagation over the set of constraints, it is possible to design a special filtering algorithm for the global constraint and thus to achieve more efficient domain filtering (Figure 3).

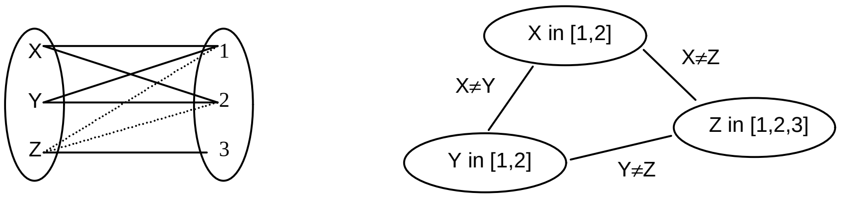

Fig. 3. Global constraint all-different (left) uses the technique of matching in bipartite graphs which achieves better domain pruning than local propagation through the set of binary inequalities (right). Values 1,2 are removed from the domain of variable Z by all-different.

The standard constraint satisfaction technique looking for feasible solutions can be extended to find out optimal solution. Usually a technique of branch-and-bound is used there. First, some feasible solution is found and then, a next solution that is better than the previous solution is looked for etc. This could be done by posting a new constraint restricting the value of the objective function by the value of the objective function for the so-far best solution.

A deep general view of constraint programming can be found in [2,7,10,11]. We will describe now how to apply CP to scheduling problems [13].

Constraints in scheduling

Scheduling problems belong to the area of combinatorial optimisation problems so they can be naturally described as constraint satisfaction problems as well. To model the problem as CSP one needs to decide how to map the problem objects into variables and constraints. One of the traditional modelling approaches uses variables to describe the activities. In particular, there are three variables identifying the position of the activity in time, namely, the start time, the end time, and the processing time (duration). Let A be an activity, we denote these variables *start(A)*, *end(A)*, and *p(A)*. We expect the domains for these variables to be discrete (e.g., natural numbers) where the release time and the deadline of the activity make natural bounds for them (and the time windows make the domains even more restricted). Note that if the processing time of the activity is constant then one variable is enough to locate the activity in time. We still prefer to use all three variables to simplify description of the constraints.

The first constraint binds the time variables of each activity: *start(A)+p(A)=end(A)*. Time dependencies between the activities can also be naturally described by constraints between the time variables. Assume that A must be processed before B, denoted A<<B, then we post the constraint *end(A)£start(B)*. In general, time dependencies between the activities can be described in the form: *min_delay(A,B) £ start(B)-end(A) £ max_delay(A,B)*. Notice that we put no restriction about the structure of the activities so arbitrary time dependencies can be modelled.

If resource allocation is included in the problem then there is one more variable for the activity. This variable describes the resource to which the activity is allocated, we denote it *source(A)*. Assume that each resource has assigned a unique number. Then the domain of *source(A)* consists of identifications for the resources to which the activity A can be allocated. This variable participates in constraints that involve the resource, e.g., there could be a tabular constraint binding *source(A)* and *p(A)* describing different processing time of the activity A in different resources.

When the activity is allocated to the resource, additional resource constraints are posted[1]. Assume that activities A and B are allocated to the same unary resource. Because no activity overlaps are allowed in unary resources we can post a disjunctive constraint: A<<B ∨ B<<A, i.e., *end(B)£start(A) Ú end(A)£start(B).* The propagation through this constraint works as follows: as soon as we know that *start(A)<end(B)* then we can deduce *end(A)£start(B)* and vice versa. If there are *n* activities in the resource then we need $O(n^2)$ binary constraints of the above form. It is a known wisdom that propagation through disjunctive constraints is rather weak. Therefore special global constraints describing the resources are used (see next section).

In the above paragraphs we give some examples of scheduling constraints. Recall that in the CSP framework one can combine arbitrary constraints so the user is allowed to use additional constraints specifying the properties of the resources and activities [4,8].

Domain filtering for scheduling

In this section we present some filtering techniques for global constraints used in scheduling applications. Recall that the filtering algorithm reduces domains of the variables and it is evoked every time a domain of any involved variable is changed.

One of the most popular scheduling global constraints is *edge finding*. We describe the version for unary resources but there exist variants for discrete resources [1] and batch resources as well [12]. The basic idea of edge finding is to identify an "edge" between the activity and the group of activities, in particular to find out if the activity must be processed before the set of activities (or after it). Assume that A is an activity and Ω is a set of activities that does not contain A. In unary resource the processing time for the set of activities equals to the sum of processing times of these activities:

$$\boldsymbol{p}(\Omega) = \sum_{\boldsymbol{X} \in \Omega} \boldsymbol{p}(\boldsymbol{X}).$$

Assume that processing of activities from Ω∪{A} does not start with A. Then processing must start with some activity from Ω so the minimal start time is:

$$\mathbf{min}(\boldsymbol{start}(\Omega)) = \min_{\boldsymbol{X} \in \Omega}\{\boldsymbol{start}(\boldsymbol{X})\}.$$

If we add the processing time of Ω∪{A} to the minimal start time of Ω and we get the time after the maximal end time of Ω∪{A} then we know that the activity A can be processed neither inside nor after Ω (Figure 4). Thus, the activity A must start before Ω. Formally:

$$\min(\mathit{start}(\Omega)) + p(\Omega) + p(\mathrm{A}) > \max(\mathit{end}(\Omega \cup \{\mathrm{A}\})) \Rightarrow \mathrm{A} \ll \Omega.$$

[1] The resource constraints are usually posted earlier over the copies of the time variables. Violation of this constraint means that the activity cannot be allocated to given resource.

$A \ll \Omega$ means that A must be processed before every activity from Ω so it must be processed before any $\Omega' \subseteq \Omega$. We can use this information to decrease the upper bound for the end time of the activity A using the formula:

$$end(\mathrm{A}) \leq \min\{ \max(end(\Omega')) - p(\Omega') \mid \Omega' \subseteq \Omega\}.$$

A similar rule can be constructed to deduce that A must be processed after Ω:

$$\min(start(\Omega \cup \{\mathrm{A}\})) + p(\Omega) + p(\mathrm{A}) > \max(end(\Omega)) \Rightarrow \Omega \ll \mathrm{A}.$$

The above edge finding rule forms the core of the filtering algorithm reducing the bounds of the time variables. It may seem that this algorithm must explore all subsets Ω of the set of activities allocated to a given resource. Fortunately we can explore only the sets defined by pairs of activities called tasks intervals [1] so the time complexity of the edge finding filtering algorithm is $O(n^3)$ where n is a number of activities allocated to the resource.

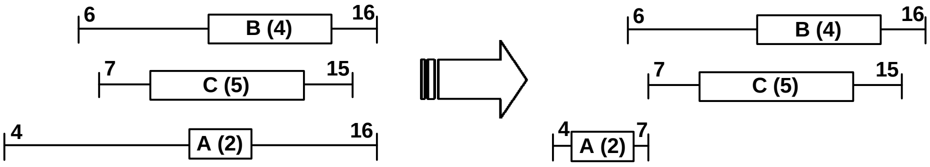

Fig. 4. Edge finding can deduce that the activity A must be processed before the activities B and C (processing time is in parentheses). Notice that binary disjunctive constraints deduce nothing there.

For discrete resources with capacity greater than one we can use a curve of necessary aggregated demand to deduce some domain filtering. This curve is computed from the activities A such that max(*start*(A))<min(*end*(A)), i.e., the activity A will consume the resource in the interval max(*start*(A))..min(*end*(A)). By aggregating demands of such activities we get necessary demand for each time point. Now for each activity we can find out time intervals where there is not enough capacity for processing this activity. Using these intervals we can reduce the time bounds for the activity (see Figure 5). Time complexity of this algorithm is $O(n)$, where n is a number of activities processed by the resource.

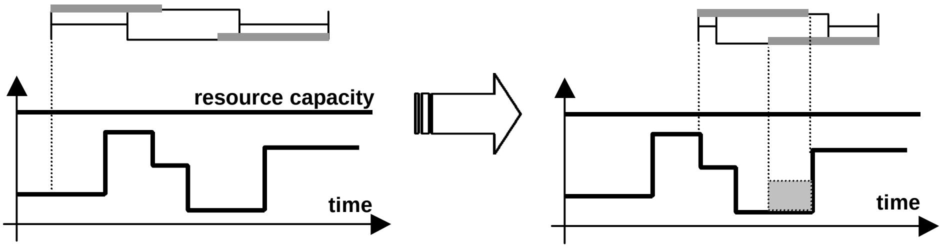

Fig. 5. Necessary aggregated demand is used for reduction of time bounds using the intervals where there is not enough capacity for processing the activity. Every activity contributes to necessary demand in times when it must be processed (a shadow rectangle on right).

Scheduling strategies

When the scheduling problem is formulated as a constraint satisfaction problem, we can use the standard enumeration techniques. They are based on trying a value for the variable, i.e., posting a constraint in the form *X=h*. If the new problem has no solution, this constraint is substituted by the constraint *X¹h* and the enumeration continues. Such technique is useful for resource allocation, i.e., assigning values to resource variables. Moreover because the real meaning of the variable is known, we can use some variable and value selection heuristics derived from the original problem. For example the activity with the minimal number of alternative resources should be allocated first (the first-fail principle) and it should be allocated to the least used resource (succeed-first principle).

For resource scheduling, i.e., deciding the time of the activity, it is more useful to use a different branching scheme, namely *X<h* and *X³h*. In particular, we can decide about ordering of two non-yet ordered activities. First, we can post the constraint A<<B, i.e., *end(A)£start(B).* If scheduling fails then we post a negation of that constraint. It could be B<<A, if both activities cannot run in parallel, or ¬ A<<B otherwise. The question is what activities should be ordered first. Again, we can use experience from solving scheduling problems saying that the bottleneck resources should be scheduled first. Such resources can be identified by the user or they can be identified automatically. First let us define the slack of the set of activities Ω using the following formula:

$$max(end(W)) - min(start(W)) - p(W).$$

Then, the resource with the minimal slack for any subset of the activities processed by that resource is scheduled first. We can use the same idea to select the pair of activities to be ordered first. The slack for the pair of activities A and B is:

$$max\{ max(end(A)) - min(start(B)), max(end(B)) - min(start(A)) \} - p(\{A,B\}).$$

Now, the pair of activities with the minimal slack is selected for ordering. Notice that the slack for two non-yet ordered activities consists of slacks for both orderings B<<A and A<<B. Ordering leading to a bigger slack is tried first.

In the above paragraphs we presented some heuristics that guide scheduling. These heuristics are part of a general search framework that could be a simple depth-first search with backtracking. Nevertheless, there exist more advanced search frameworks like limited discrepancy search [6] that proved to be very efficient especially in scheduling problems.

Conclusions

Constraint-based scheduling is a glass-box framework for solving scheduling problems. It has two major advantages over the existing scheduling approaches: clarity (thus glass-box) and generality of the models. Moreover, it provides generic solution techniques of constraint satisfaction that can be further tuned for scheduling problems by using special filtering algorithms and scheduling strategies. Despite its "young age", constraint-based scheduling proved itself to be an efficient tool for solving real-life scheduling problems. In fact, one of the leading companies in the optimisation industry, ILOG, is using constraint satisfaction as a core technology in their products.

References

[1] Baptiste, P., 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.: On-line Guide to Constraint Programming, Prague, http://kti.mff.cuni.cz/~bartak/constraints/ (1998).

[3] Barták, R.: Filtering Algorithms for Tabular Constraints, in *Proceedings of CP2001 Workshop CICLOPS*, Paphos, Cyprus (2001), 168-182.

[4] Barták, R.: Visopt ShopFloor: On the Edge of Planning and Scheduling. In *Proceedings of ERCIM and CologNet Workshop on Constraint Solving and Constraint Logic Programming*, Cork, Ireland (2002), to appear.

[5] Brucker P.: *Scheduling Algorithms*. Springer Verlag (2001).

[6] Harvey W.D., Ginsberg, M.L.: Limited Discrepancy Search. in Proceedings of *International Joint Conference on Artificial Intelligence*, (1995) 607-613.

[7] Kumar, V.: Algorithms for Constraint Satisfaction Problems: A Survey, *AI Magazine* 13(1): 32-44 (1992).

[8] 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.

[9] Régin J.-Ch..: A filtering algorithm for constraints of difference in CSPs. In *Proc. 12th National Conference on Artificial Intelligence* (1994).

[10] Tsang, E.: *Foundations of Constraint Satisfaction.* Academic Press, London (1995).

[11] van Hentenryck, P.: *Constraint Satisfaction in Logic Programming.* The MIT Press, Cambridge, Mass. (1989).

[12] Vilím P., Barták, R.: Filtering Algorithms for Batch Processing with Sequence Dependent Setup Times. In *Proceedings of The Sixth International Conference on Artificial Intelligence Planning and Scheduling*, Toulouse, France (2002), 173-181.

[13] Wallace, M.: Applying Constraints for Scheduling. In *Constraint Programming*, Mayoh B. and Penjaak J. (eds.), NATO ASI Series, Springer Verlag (1994).