VisOpt JobShop
The Solver behind the User Interaction

Roman Barták

InSol Ltd., Haifa, Israel
Web: http://www.insol.co.il/
e-mail: bartak@insol.co.il

Contents

This document describes the scheduling part of the VisOpt system from the end-user point of view. First, we specify the problem areas that the scheduler is suitable for. Then, we describe the technology behind the scheduler and finally we look at the "internals" of the scheduling engine.

Problem Areas

In general, the VisOpt scheduler is appropriate for almost all production scheduling problems. In particular, we concentrate on problems where many different types of resources appear. We call such problems *heterogeneous* because the resources involved in the scheduling task can be of many different types, like producer, mover, store, tool, worker, etc. Actually, we include custom and purchase orders among resources because, in our terminology, resource is everything that is being scheduled[1]. Note that the scheduling engine is (almost) independent of particular structure of resources so it is possible to extend the scheduler with other resource types.

The VisOpt scheduler is especially suitable for *complex environment*s. It is possible to specify various transitions between subsequent activities of the resource including different set-up, change and processing times. It is also possible to express dependencies between resources, the most natural dependency is supplier-consumer relation.

The ability to express many *constraints* (even user defined) between parameters of resources is another advantage of the VisOpt scheduler. Because we use the constraint technology behind the scheduler (see next section) we are capable of capturing many dependencies in a very natural way for the user. Of course, the scheduler follows all these dependencies so the final schedule is consistent according to the user's specifications.

Moreover, the scheduler does not find an arbitrary schedule but a schedule close to the *optimum*. We use the notion of cost (or profit) as a measure of optimality so the scheduler tries to minimise the cost (or maximise the profit). The user can specify various cost functions from the fixed cost to the cost being dependent on other parameters. The scheduler is capable of finding an optimal schedule but this is usually too (time) expensive. Therefore, the scheduler returns any "good" solution first (it is up to the user to define what is good) and, if there is time left, the scheduler can improve the plan even more.

Because of above features, we intend the VisOpt scheduler for very complex areas like plastic petrochemical, chemical or pharmaceutical industries.

Background

There is a sophisticated technology behind the scheduler in VisOpt called *constraint programming* (CP), originating from Artificial Intelligence (AI). It is based on idea of describing the problem as a set of constraints, i.e., dependencies among parameters (unknowns), and solving these constraints.

In general, a constraint is an arbitrary relation among several unknowns. It is possible to describe dependencies between unknowns with different domains (numbers, strings, names,

[1] Purchase and custom orders are scheduled because we have to find the exact time of the order and the structure of the order (number of items, etc.)

etc.), so many real-life relations can be captured in natural way. Here are some examples of constraints (of course, we are able to model all of them in the VisOpt scheduler):

- at time interval <1,10> six workers with the qualification *x* are available
- it takes five minutes to start (or to load) the machine
- before producing item B we need at least eight hours of production of item A
- if tool 1 is used then the production rate is 5, otherwise the production rate is 2.

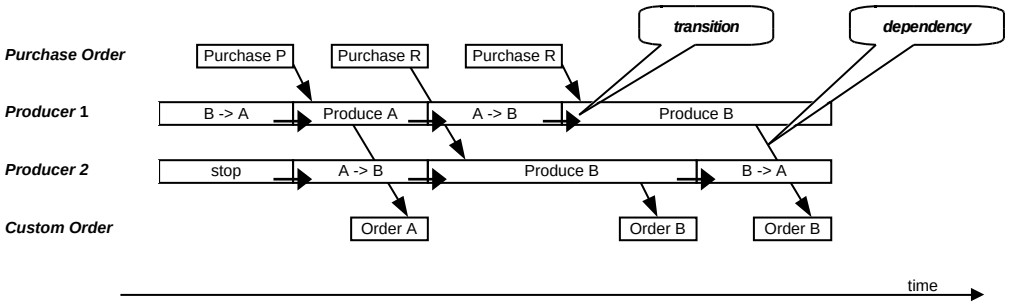
As the above examples show, the problem specification using constraints is very close to user's point of view and, consequently, the end-user understands what is loaded into the system. This is one of the main advantages of constraint programming as opposed to other approaches, such as operational research.

Constraint programming is a close approach to the Holy Grail of computing: a user states a problem and the computer solves it. Of course, there is a powerful technology in the background that is responsible for solving the problems modelled by constraints. This solving technology is based on searching, an ancient AI technology where the constraints are used to reduce the search space. Therefore we do not need to explore all possibilities, e.g., all possible schedules, but only such solutions that satisfy the constraints. This improves the efficiency dramatically. It is also possible to look for best (optimal) solutions using some objective function describing how good the particular solution is. One of the advantages is that we can get sub-optimal solutions in specified time so it is not necessary to wait for optimum if we are satisfied with the current solution. Finally, constraint programming allows using of heuristics, a user experience expressed in some formal fashion that can improve the efficiency even more.

Scheduling Engine

We utilise the constraint technology in the VisOpt scheduler, which allows us to prepare general engine capable of solving various scheduling problems. The scheduling engine is based on the resource-centric model[2], i.e., we schedule the behaviour of individual resources according to the internal limits of these resources (state transitions, etc.) and dependencies between resources. There are certain advantages achieved, for example we can handle by-products, i.e., products that were not ordered but resulted from producing another product, or we can work with various production sequences (alternative processes) to satisfy the order. The important thing is that this model is not dependent on custom orders so it is possible to schedule the production according to some general rules without necessity of putting in the orders (e.g., inventory level policy). However, it is possible to use custom orders to drive the scheduling as well.

Take a look inside the scheduling engine. As we mentioned above, we schedule the behaviour of each resource. This behaviour is specified by the set (usually a sequence) of activities where each activity describes some homogenous part of resource life. The activity can have various parameters, e.g., duration, input specification, etc., and constraints between these parameters. It is also possible to specify relations between activities of the same resource. Each such relation is called a transition because it usually describes the transitional rules between successive activities. Finally, one can specify relations between activities of different resources, like supplier-consumer relation. We call these relations dependencies.



[2] There also exists order-centric model that is based on scheduling production sequences for individual orders, i.e., sequences of actions to satisfy the particular order. However, this model is not able to handle by-products and it cannot be used to schedule non-order driven plants.

The scheduler looks for such schedules that all activity constraints, transitions and dependencies, appearing in the schedule are valid. Moreover, it is possible to specify a cost (or profit) of each activity and the scheduler tries to minimise the cost (or maximise the profit) of the schedule. Because looking for optimal solution is usually too time expensive (and, in fact, it is not necessary to find optimum) we provide the mechanism for finding sub-optimal solutions. The user can specify the cost level that is unacceptable for him/her, so the scheduler does not care about schedules with higher cost. Additionally, the user can specify the cost level that is acceptable, so if the scheduler finds a schedule with this (or better) cost, it returns the schedule to the user immediately. Now, if the user is still not satisfied with the schedule it is possible to ask the scheduler to find an even better schedule (in fact, if there is enough time, the scheduler is able to find the optimal schedule). It is also possible to stop the scheduler when it is in the ”shadow” zone between the acceptable and unacceptable cost levels. In such case, it returns current schedule that is not too bad (the cost is lower than the unacceptable level) but it is still not acceptable as specified by the user (the cost is higher than the acceptable level). Again, the user can decide whether the schedule is appropriate or whether the scheduler should find a better schedule. We call such strategy an almost-anytime scheduling because the scheduler is ready to return some schedule at almost each time.

Conclusions

VisOpt JobShop Scheduler is a generic scheduling engine capable of solving various scheduling problems, including optimisation.