Constraint-based reasoning for timetabling

Wojciech Legierski

Institute of Automatic Control, Silesian University of Technology,
Akademicka 16, 44-100 Gliwice
e-mail: wlegierski@ia.polsl.gliwice.pl

Abstract

The paper deals with an effectiveness of Constraint Programming (CP) for scheduling problem particularly for timetabling. The main advantage of the CP is declarativity, a straightforward statement of the constraints serves as part of the program and it is compared with local search procedures. The CP main features, constraint propagation and distribution, are briefly presented. The constraint programming language Mozart-Oz allows to express complex constraints and create complicated, custom-tailored distribution strategy, which is needed for solving timetabling problem. Incorporation of the local search into constraint programming is needed as a method for optimization. Results of implementation for real-world case are presented.

Keywords: constraint programming, scheduling, timetabling, search strategy, optimisation

1. Introduction

Scheduling problems are in the core of many real-world applications. They occur in areas of production planning, timetabling or personnel planning. For certain well-defined problem classes there exist efficient algorithms from Operations Research (OR). But these algorithms are often very specific and slight changes in the problem definition raise difficulties in the adaptation of the special purpose algorithms.

There is possibility of instantly changing already defined algorithms and making it adequate for real-world problems, but it is hard and it often lowers their effectiveness. One of the offspring of AI is Constraint Programming (CP), which offers flexibility by the formulation of constraints in a high-level language. Its main advantage is declarativity: a straightforward statement of the constraints serves as part of the program. This makes the program easy to modify, which is crucial in real-world problems.

Constraint Programming has succeeded in solving standard benchmarks and real-world problems from the area of scheduling [13]. The construction of timetabling falls under the class of scheduling problems. It is large, highly constrained and much more complicated. Problem differs greatly for different schools and educational institutions. Although manual construction of timetables is time-consuming, it is still widespread, because of lack of appropriate programs. This forces from programmer to use a ‘good’ timetable, custom tailored distribution (labeling) strategy is able to introduce ‘soft’ constraints during search, leading quickly to a ‘good’ timetable, incorporation of local search into constraint programming gives the ability to optimize effectively the timetable.

2. Constraint Programming

Constraint Programming techniques have been developed since about 1990. Because they base on backtracking search, at the beginning they have been developed in Prolog, where backtracking and declarativity had been already implemented. In this way Constraint Logic Programming (CLP) was created as a addition to Logic Programming (LP). The languages from this area, which are still popular, are CHIP, Sicstus, Eclips to name a few. Then CP leave a Prolog and comes into two branches – one of them is C/C++ libraries (e.g. ILOG) and the second is multiparadigm languages (e.g. Mozart/OZ). All of these languages have two common features – constraint propagation and distribution (labelling) connected with search.

2.1. Constraint propagation

It is clue of Constraint Programming. Shortly it is automatically removing from the domain of variables all values that do not fulfil constraints. For example, if we have two variables from finite domain x and y, where:

\[ x \in \{1, 5\}, y \in \{1, 6\}\]

And we introduce a constraint saying that \(x>y\), the constraint propagation reduces domains to those values:

\[ x \in \{3, 4, 5\}, y \in \{1, 2, 3\}\]

because values \{1, 2\} from \(x\) domain do not fulfil the constraint \(x>y\) and values \{4, 5, 6\} from \(y\) domain also conflict with the constraint. When we introduce a second constraint \(x+y=6\) none of the values can be removed.

Constraints are not usually so simple as presented in the previous example and often are connected with each other. Therefore constraint propagation does not remove all values that are in conflict with all constraints and its performance is measured as a trade-off between number of removed values and execution time.

2.2. Distribution and search

In the most cases constraint propagation does not lead to the solution (as it is also depicted in above-mentioned example). Therefore there is always added to constraint propagation a distribution connected with search. Distribution is based on incorporation of an additional constraint, often it is a constraint saying
about equality of one variable to one value (one of the task of the distribution is to choose a proper variable and a value). When it is done a consistency is checked and there are three possibilities:
- a solution is found,
- variables domains are narrowed, but there is no unique solution, so distribution is made with another variable,
- the additional constraint is inconsistent with other constraints, so the backtrack is made and from chosen variable domain a chosen value is removed.

This process is made in iterative way and is called search. Search is responsible for stopping after finding: first solution or some number of solution or all solution. Search forms a search tree, where each node is a state of variables. In Fig.1 is depicted a search tree for example mentioned in section 2.1.

Fig. 1. An example of search tree.

2.3. CP for standard benchmarks

Efficiency of the CP was shown on the standard benchmarks for ‘job-shop’ problems. It describes a schedule of the \( n \) jobs on the \( m \) resources. Each job consists of \( m \) tasks that each must be schedule on different resource and they must not overlap. In Table 2. are presented time of finding solution and time of proving optimality for two problems, where 10 jobs are schedule on 10 resources. The computation was made on station Pentium III 850MHz/ 256 MB RAM, using Mozart/OZ. These problems were presented in publications: ABZ6 [2], MT10 [8]. It is worth to notice, that proving of optimality for famous MT10 problem takes more than 25 years and was found in 1989r [3].

Table 1. Computation time for standard benchmarks.

<table>
<thead>
<tr>
<th></th>
<th>Best solution search</th>
<th>Optimality proving</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABZ6</td>
<td>9[s]</td>
<td>1[s]</td>
</tr>
<tr>
<td>MT10</td>
<td>35[s]</td>
<td>6[s]</td>
</tr>
</tbody>
</table>

Not only CP techniques lead to quick solving of the ‘job-shop’ problems, but first of all described by Würtz [13][7] a special propagator Schedule.serialize and distributor Schedule.firstsLastsDist which were implemented in Mozart/OZ. Analysis of the solution method for ‘job-shop’ problems allows to develop a proper method for timetabling problem. First of all an adequate search strategy, because introducing a complicated propagators does not lead to a good results [6].

3. Comparison with local search

During the last years two main approaches seem to be successful for solving timetabling problems. The first approach is based on local search procedures such as simulated annealing [4], tabu search [12], genetic algorithms [9]. These methods express constraints as some cost functions, which are minimised by heuristic search of better solutions in a neighbourhood of some initial feasible solution. They were implemented with success particularly for big timetable problems. Their greatest disadvantages are:
- the difficulty of taking into account hard constraints,
- the determination of their parameters through experimentation.

Although they are good for optimising the initial feasible solution, they have problems with finding it. The second approach that is presented in this paper is based on CP paradigm [5] [6]. Its main advantage is declarativity. The constraints are handled through a system of constraint propagation, which reduces domains of variables, coupled with backtracking search. In modern CP languages, both features don’t need to be programmed explicitly. The main disadvantages of this approach are:
- difficulties with expressing ‘soft’ constraints,
- problems with improving the initial feasible solution.

An attempt to overcome the drawbacks with ‘soft’ constraints was discussed by Rudova [10]. Both local search procedures and CP have advantages and disadvantages therefore it is worth to connect these two approaches to overcome drawbacks. White and Zhang [12] made a successful approach to combine local search with constraint satisfaction. They determined an initial solution using constraint logic programming and then optimised it using tabu search, but it was rather switching between approaches. An idea of this paper is to incorporate local search into constraint programming.

4. Constraints for timetabling

Timetable scheduling is a very constrained problem, where constraints are divided into a ‘hard’ constraints that must be fulfil and ‘soft’ constraints that should be fulfilled as much as possible. ‘Soft’ constraints are implemented as a cost function that is minimised. The Constraint Programming paradigm gives the ability to straightforward express ‘hard’ constraints that are handled through constraint propagation mechanism implemented in the system. The formulation of ‘hard’ constraints is difficult for two reasons:
- it is not unique and always obvious,
- it influences strongly the propagation efficiency.

Constraints should be introduced in a way to maximize propagation efficiency (how many values are removed from the domains of variables) and minimize time of checking them.

4.1. Reified constraints

Sometimes constraints express connections of variables in a complicated and indirect way. To cope with them, reified constraints are needed, which are provided by the Mozart-Oz language. Generally, reified constraints are used to reflect the fact that some constraint is fulfilled by attaching to it a proper value of some 0/1-value variable. Through this variable the fulfilment of the constraint can be checked, if it is enumerated, it imposes either the validity or the negation of the constraint. It can be also used to build nested constraints, which connect not only variables, but also other constraints. Some timetable problem constraints need to be expressed using reified constraints (e.g. some courses must be held in different days).

4.2. Constraint “one course in a room”

The analysis described in [6] shows the advantage of refraining from complex non-overlapping constraints. The performance
increase was achieved through proper introducing constraints connected with room allocation (“one course in a room”), which was performed simultaneously with search for start times of courses. The hardness of timetabling is often due to a rather small number of rooms available; making room allocation after courses have been scheduled would lead to many unfeasible solutions. Therefore the constraint “one course in a room” seems to be the most difficult constraint. The essence of this difficulty is that it depends on two undetermined variables (start time and room) of any course and corresponding values of all the other courses. It can be written as a rather complicated reified constraints, which checks every room, in every timeslot, for every course. But it creates a lot of propagators that have to be solved during the entire search. It is computationally very expensive.

This problem was solved by changing of the standard CP program structure. Generally all constraints should be introduced before searching solutions by variables instantiation. This rule was omitted only for the constraint “one course in a room”, which is introduced after instantiation of room and start time for a specific course and refers to all unlabelled courses. The program does not need to search at each step every possible room and start time to find inconsistencies. This approach (called constraining while labelling) gave good results [6].

5. Search strategy

Mozart’s search efficiency is due to two major factors:
- the distribution strategy, which determines what variables are selected for instantiation and which value they are instantiated with,
- the search method, which determines how the search tree is explored.

The distribution strategy and search method is referred to as search strategy. The timetabling problem is not a standard scheduling problem, for which the standard search strategies can be used effectively. The complexity of the problem suggests a special search strategy.

5.1. Distribution strategy

For timetabling the distribution strategy tries to answers the question “Which courses should be scheduled first and in which timeslots?”. The main assumptions for a good distribution strategy are:
- reduction of backtracks (decrease the frequency of constraint violation),
- searching for a ‘good’ timetable right away (fulfil as many ‘soft’ constraints as possible)

Assessment for values in domain was introduced in solving timetable by Abdennadher and Marte [1]. They represent a domain as a list of value-assessment pairs. For example, X :: [3, 4, 5] can be [(3, 0), (4, 1), (5, 8)]. Next they use Constraint Handling Rules (CHR) for solving Weighted CSP. For variable selection they use first-fail strategy and value was selected with minimal assessment.

The idea of value assessment for variables related to start times was applied to the presented timetable problem as well as the weighted CSP framework. The assessment of the value corresponds to the fulfillment of the ‘soft’ constraint and the hardness of scheduling the course. The assessment of values was made in two stages:
1. One time only, at the beginning of the search, each value depends on:
   - position in a day (morning hours have lower assessment),
   - hardness coefficient of teacher giving courses. It is defined as follows:
   \[
   \text{hardness}\_\text{coefficient} = \frac{\text{duration}\_\text{of}\_\text{teacher}'s\_\text{courses}}{\text{availability}\_\text{times}\_\text{of}\_\text{teacher}}
   \]
   - kinds of course (common courses for several group have lower assessment),
   - how many teachers gives the course,
   - in how many rooms course is placed.
2. Many times, during the search, each value assessment is recomputed if the domain of variable is changed and depends on:
   - the size of the domain, where values for variables with small domain have smaller assessment,
   - the possibility of schedule pairs for courses running in an odd-even weakly cycle,
   - the number of courses of the same group already scheduled in a day corresponding with value,
   - whether the value corresponding to the start time does not make gaps between already scheduled courses,
   - the number of values in the domain, which do not make gaps,
   - the duration of a gap if it occurs.

The value assessment is used for variable and value selection. The variable with the smallest assessment of one of the value in its domain is chosen and to this value is variable instantiated. The diagram of the distribution strategy is presented in Figure 2. The described distribution strategy was crucial for finding timetables in reasonable time.

![Fig. 2. Diagram of the distribution strategy](image)

5.2. Search method and optimisation

Search method determines how search tree is explored, how many solutions should be found and if more than one, it implements method for finding next solution. There are several search methods for a single feasible solution, but for timetabling problems depth-first search (DFS) seems to be best. This obvious way to explore a search tree from left to right side is justified by the distribution strategy, which attempts to make the start with leftmost leaves most preferable.

Finding feasible solution of timetabling problem is often not sufficient. The timetable should be ‘good’ due to requirements specific for an institution. Finding right away a ‘good’ timetable
was described in previous subsection, but there is still a need for better solutions. The most popular method for finding better solution, branch-and-bound (BAB) search, was checked. The chosen criterion was the overall number of gaps for all groups, which is the most important soft constraint. This method did not give any better solution. There are three reasons. Distribution strategy as described in previous subsection already tries to minimize the gaps; therefore branch-and-bound is given little chance for improving the already good situation. The second reason is that main advantage of BAB - pruning the branches with worse cost function - is not effective, because it can be fully computed after instantiation of all courses. Performing a complete branch-and-bound search is impossible because of the problem size and virtual memory restrictions. At last BAB is a systematic method. It maintains the ordering of variables from distribution strategy and tries to change their values beginning from the bottom of a search tree. It makes redundant the work of checking different configuration of the last variables. For this reasons another optimisation method was developed.

5.3. Incorporation of local search into CP

Local search techniques are based on searching for a better solution in the neighbourhood of the already found one. For timetabling problem this neighbourhood can be e.g. a timetable with one course placed differently or with two courses exchanged. It seems easy to make this kind of operation and compute a cost function, but when the problem is described using constraint programming some restriction are necessary. The constraint propagation needs to know explicitly the state of variables (either their values or domains), the search must be done by instantiating variables one by one. From a timetable point of view it is possible to replace during the search only the latest scheduled course. To overcome this restriction a new search method was proposed.

Mozart-Oz enables programming of a customized search method described in details by Shulte [11]. It provides ability of building new search from already found solutions. The search engine, after finding the first solution, makes the following steps:
1. Finds a course which makes gaps between lecture (takes course schedule with the highest assessment).
2. Finds a second course to exchange with first.
3. Creates a new search from the original problem.
4. Instantiates to the values from solution all courses besides these two chosen (it can be made in one step because they surely do not violate constraints).
5. Schedules first course at the place of the second.
6. Finds the best start time for the second course.
7. Computes the cost function. If it is better than the solution found so far, the new solution is memorized else another exchange is made.

Next solution is not based on the previous search tree, but it is based only on the previous solution. New search tree is generated, where almost all nodes are unchanged, besides nodes corresponding to two courses, which are exchanged (Fig.3).

6. Results and conclusion

The approach presented in the paper allows to generate feasible solutions for real university department timetabling data. The university institute problem (223 courses) was tested on a Pentium III/850 MHz, 256 MB station. The first solution was found after 45s, and 20% improvement of number of gaps was achieved after next 20s using the presented idea of the local search, whereas BAB did not give any improvements.

The features of the multiparadigm Constraint Programming Mozart/OZ language allow formulation of complicated timetabling constraints. The strategy is specialized for timetabling problems and based on value assessment. Using standard distribution strategy and search method seems to be insufficient to deal with it effectively. That is why the custom-tailored search strategy for timetabling problem allows solving the problems rather quickly. The new idea of incorporating local search into constraint programming was presented which is much better than commonly used BAB. Developing new methods of finding better solution seems to be crucial and they should be adapted to the nature of real problems.

References