Bulletin of the *Transilvania* University of Braşov
Series I: Engineering Sciences • Vol. 4 (53) No. 2 - 2011

CONSTRAINT BASED APPROACH FOR OPTIMIZED PLANNING-SCHEDULING PROBLEMS

A. GÎRBEA[1] C. SUCIU[1,2] F. ŞIŞAK[1]

Abstract: *This paper outlines the way the planning/scheduling applications can be solved using the optimization potential of the Choco CSP solver. Consequently two specific problems which can be applied in various fields were described in this paper. The first one is a pure planning problem and supposes that a factory should manufacture products ordered by a client in the shortest time possible. The second one is a scheduling problem regarding simultaneous file transmission-visualization, where the task is to determine the start moments of the copy operations. The results show that both problems, otherwise difficult to solve in a classical way, can be solved readily using the CSP approach.*

Key words: *CSP, scheduling, planning, optimization.*

1. Introduction

Constraint programming is the study of computational systems based on constraints. A constraint is a logical relation between several variables, where each variable has a predefined domain. Thus a constraint restricts the possible values that variables can take and it represents some partial information about the variables of interest [1]. The problems solved through this approach are called *Constraint Satisfaction Problems* (CSP) and consist of:

- a set of variables: $x_1, x_2, \ldots, x_n$;
- a set of possible values for each variable: $D_1, D_2, \ldots, D_n$;
- a set of constraints which restrict either the values of a single variable (unary constraint) or the values which a set of variables can simultaneously take (binary constraints, ternary constraints etc.).

The solution of a CSP problem consists of a tuple $v = \{v_1, v_2, \ldots, v_n\}$ specifying a value for each variable, values which satisfy all constraints [8].

Constraint programming has been successfully applied in various domains: operations research problems [2] (scheduling and routing), database systems (consistency of data), business applications (option trading) etc. The two main goals of the CSP domain are the formulation and the resolution of the combinatorial problems [3]. This is a very effective way of solving several industrial problems such as scheduling, planning or design of timetables. The user has to only build the model of the problem, he is not interested in the way the problem is solved. Several frameworks exist to implement Constraint

[1] Dept. of Automatics, *Transilvania* University of Braşov.
[2] Corporate Technology, PSE Siemens, Romania.

Programming [9]: ECLiPSE, CHOCO, KOALOG, ILOG SOLVER, ILOG SCHEDULER, ILOG OPL, We have sought to use an open-source solver so that our research activity should not be limited by any license agreement and it should be accessible to every one. Secondly, we have tried to use a Java-based solver because we want to integrate the solver into an application also containing an OPC UA server [10], which has already been developed by us. Through the combined use of the solver and the OPC UA server we are able to immediately and automatically use the solutions of the problems without any human tasks [7].

The two most important open-source Java solvers are Choco [12] and JaCoP [13]. After carefully analyzing the two APIs we have chosen to use the Choco solver. The Choco solver has a better documentation and the code is easier to understand [4]. Also its API contains more constraints when compared to JaCoP, which has a rather minimalistic approach regarding the supported constraints, and it allows the use of TaskVariables which is very useful for scheduling problems. The only disadvantage of Choco is that it requires more system resources and it has longer solving times. A very important feature of Choco is the possibility to define an objective variable which is used to determine the best solution of all possible solutions of the problem. The solver will seek to find the values which either maximize or minimize the chosen objective variable, as specified by the user.

The goal of this article is to asses the optimization potential of the Choco CSP solver for planning-scheduling applications, based on two specific problems, which we have developed. The first one is a pure planning problem [6] and refers to a factory which manufactures different types of screws on various work stations which work in parallel. The goal is to plan the manufacturing of the screws on the work stations so as to obtain a minimum execution time.

The second problem is a scheduling problem which can be applied in various fields. In order to provide a specific framework the following situation is considered: four files, each representing a part of a movie, have to be sent over a network and viewed afterwards. Each file has a specific size and duration. The files can be sent simultaneously but the transmission speed decreases sequentially with increased number of simultaneous transfers.

The task is to determine the timing of the transmissions, namely the moments in time when to start to copy the files so as to finish the visualization of these four files as soon as possible.

During the next section we will focus on the first problem and its particularities. Section three presents a detailed description of the second problem. Then section four presents the results and finally we will draw some conclusions on our work in section five.

2. Optimized Planning for Part Manufacturing

The first problem refers to a factory which has to manufacture products requested by a client. There are four work stations which work in parallel and each of them can be used to manufacture four types of screws (*R*1, *R*2, *R*3, *R*4). When a client sends an order, the goal is to schedule the manufacturing of the screws on the three work stations so as to obtain a minimum execution time.

The constraint satisfaction problem, which we have implemented, is composed of a model (in which all the constraints and consequently all the variables are added) and a solver (which reads the model and returns one, all or the optimum solution).

After the creation of the model twelve main variables, whose values will be determined by the solver, are created. Four *arrays* of four elements each are defined, every array represents a screw type and every element of an array representing how many screws of that type will be manufactured on the corresponding work station. The upper bound of every variable represents the number of screws requested by the client.

After that we have added a set of constraints stating that the sum of the number of screws of a certain type manufactured on the work stations has to be equal to the number requested by the client.

We have also specified the execution times (expressed in minutes) for each type of screw on each machine (Table 1 - as one can see, the manufacturing times of the screws depend on the work stations on which they are manufactured). These values are specified in this case as integers (constants). Afterwards another three variables corresponding to the manufacturing times of the three stations have been created.

Manufacturing times of the screws on the various work stations Table 1

Screw type	Work station 1	Work station 2	Work station 3	Work station 4
*R*1	3	2	4	9
*R*2	2	4	5	6
*R*3	2	3	7	5
*R*4	5	6	4	7

Listing 1. *Minimization of the greatest manufacturing time*

```
IntegerExpressionVariable[]manuf_times=new IntegerExpressionVariable[] { manuf_timeWS1,
manuf_timeWS2, manuf_timeWS3, manuf_timeWS4};
IntegerExpressionVariable total_time_max=Choco.max(manuf_times);
IntegerVariable time_max = Choco.makeIntVar("time_max", 0, 45);
m.addConstraint(Choco.eq(total_time_max, time_max));
Solver s=new CPSolver();
s.read(m);
s.minimize(s.getVar(time_max), true);
```

Then a fourth variable representing the maximum of the three manufacturing times on the work stations has been defined.

Finally, after creating the solver object, the goal is specified, namely the variable representing the maximum of the four manufacturing times has to be minimum (Listing 1 displays the code corresponding to these actions).

3. Optimized Scheduling for File Transmissions

The second problem represents a scheduling problem. A scheduling problem represents the allocation of resources to activities over time so that input demands are met in a timely and cost-effective manner [5].

Most typically, this involves determining a set of start and end times of activities, together with resource assignments, which satisfy all temporal constraints on activity execution (following from process considerations), satisfy resource capacity constraints, and optimize some set of performance objectives [11].

This problem consists in sending four files over a network. Each has a size of 1 GB and represents a part of a movie with duration of 30 minutes. The transmission speed depends on the number of files that are sent simultaneously. If only one file is sent the transmission speed is of 200 kB/s, for two files the speed is of 170 kB/s, for three files the speed is of 140 kB/s and for four files the speed is of 110 kB/s.

The goal is to determine the timing of the transmissions, namely the moments in time when to start to copy the files so as to finish the visualization of the four files as soon as possible. Two natural constraints are that a person can start visualizing a file only when he/she has received it and that only one file can be visualized at a time.

Even if at first sight the problem seems easy it is very difficult to solve because several scenarios have to be taken into consideration. Figure 1 shows the various overlapping possibilities regarding the transmission of the four files.

In the model of the problem the transmission speeds are defined as *IntegerConstantVariables*. Afterwards we have defined four *TaskVariables* for the four files and other four *TaskVariables* for the visualization of the files.

Each *TaskVariable* consists of three *IntegerVariables*: the duration, the start and the end moments. Setting fair values for these IntegerVariables is very important

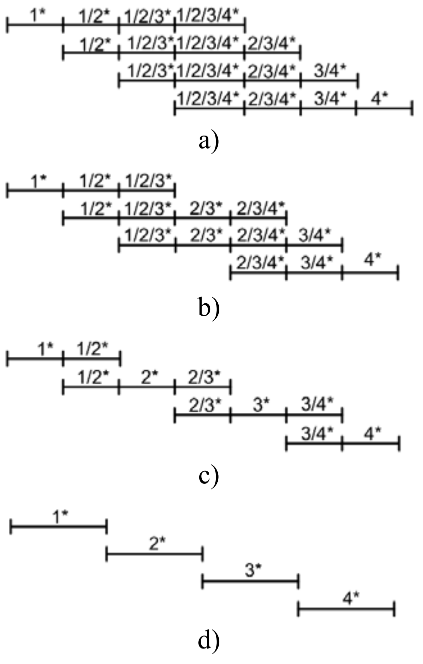

Fig. 1. *Overlapping scenarios*

not only in order to obtain a correct solution but also to obtain it in a timely manner. Further, to obtain the solution faster we have decided to express files sizes in MB (megabytes) and time intervals as multiples of five minutes. Hence the transmission time of a file can vary between the values of 17 (obtained when sending only one file at a time) and 30 (obtained when sending four files simultaneously). The start moment of the first file is zero of course. Consequently the end moment of the transmission of the first file lies between 17 and 30. For the second file the start moment may vary between 0 and 17. And the final moment for the second file may vary between 19 (in case two files are transmitted simultaneously) and 33 (in case the files are sent like in Figure 1d). In a similar way the start and end times of the other two files are set.

The duration of visualization tasks is 6 (30 mins/5 = 6) and of course only one file can be visualized at a time. The start moment for the visualization of the first file is equal to the final moment of the transmission of the same file. And the final moment is equal to the start moment plus the visualization time. The same rule applies to other files. Listing 2 displays the creation of the tasks corresponding to the first file.

In order to model the problem adequately two sets of variables (IntegerVariables) have been defined: with and without star. Each set contains both the durations of the stand-alone tasks and of the overlapping areas (for two, three and four simultaneous transmissions): 1, 2, 3, 4, 1/2, 1/2/3, 1/2/3/4, 2/3, 2/3/4, 3/4. The variables without a star, which represent overlapping areas, are defined by differences between the end and the start times of the tasks which define them. Next the variables with stars and the constraints for these variables have been defined. A variable with star represents the duration of a stand-alone task or overlapping

Listing 2. *Operations corresponding to the first file*

```
IntegerVariable duration_t1=Choco.makeIntVar("duration_trans_1", 17,30);
IntegerVariable start_t1=Choco.makeIntVar("start_mom_trans_1", 0,0);
IntegerVariable final_t1=Choco.makeIntVar("final_mom_trans_1", 17,30);
TaskVariable task_file1=new TaskVariable("file_trans_1",start_t1,final_t1,duration_t1);

//The task defined for the visualization of the first file
IntegerVariable duration_v1=Choco.makeIntVar("duration_vis_1", 6,6);
IntegerVariable start_v1=Choco.makeIntVar("start_vis_1", 17,30);
IntegerVariable final_v1=Choco.makeIntVar("final_vis_1", 17+6,30+6);
TaskVariable task_view1=new TaskVariable("file_vis_1",start_v1,final_v1,duration_v1);
```

area without interference of other stand-alone tasks or overlapping tasks. The durations displayed in Figure 1 are all with stars; for example, in Figure 1a, the variable 1/2 without star is composed of the variables $1/2^*$, $1/2/3^*$ and $1/2/3/4^*$ because this is the actual overlapping area between tasks one and two. The variable $1/2^*$ on the other hand represents only that portion of 1/2 where no other task interferes.

In order to determine the values of the variables with stars "*if-then-else*" constraints have to be used. Hence the values of these variables are defined as differences between the same variables without a star and their higher ordered neighbors. The "*if-then-else*" has to be used to avoid negative values for these variables. Listing 3 displays the definition of the variable $2/3^*$.

If the variable with star has only one neighbor then it is not necessary to use the "*if-then-else*" constraint and its value is equal with the same variable without a star minus the higher-order neighbor.

Then we have defined constraints for the four files:

$$File\ size = \sum_{i=1}^{n} t_i^* \cdot v_i \ . \qquad (1)$$

As duration variables we have used the previously defined variables with stars. Eq. (1) has been implemented using constraints like "*mult*" and "*sum*". Finally we have used the "*geq*" constraint to specify that the result of the sum is greater or equal with size of a file (1000MB). We have used the *greater or equal* constraint instead of the *equal* constraint because the file sizes are expressed in MB and the time intervals as multiples of five minutes and hence the multiplication between different speeds and durations will not be exactly 1GB. Listing 4 displays the operations for one task.

Listing 3. *Definition of variable $2/3^*$*

```
IntegerVariable duration_2_3_star=Choco.makeIntVar("2/3*", 0,15);
m.addConstraint(Choco.ifThenElse(Choco.gt(Choco.sum(duration_1_2_3,duration_2_3_4),duration_2_3),
Choco.eq(duration_2_3_star,0),Choco.eq(duration_2_3_star,Choco.sum(duration_2_3,
Choco.neg(duration_1_2_3),Choco.neg(duration_2_3_4)))));
```

Listing 4. *File size constraint*

```
IntegerExpressionVariable file1 =Choco.sum(Choco.mult(duration_1_star, transmission_1_file),
Choco.mult(duration_1_2_star, transmission_2_files),Choco.mult(duration_1_2_3_star,
transmission_3_files),Choco.mult(duration_1_2_3_4_star, transmission_4_files));
IntegerConstantVariable size_file1 = Choco.constant(1000);
m.addConstraint(Choco.geq(file1,size_file1));
```

Listing 5. *Precedence constraints*

```
//The transmission of file two has to start no later than the end of transmission for file one
m.addConstraint(Choco.startsBeforeEnd(task_file2, task_file1));
//File two can be visualized only after file one was visualized
m.addConstraint(Choco.startsAfterEnd(task_view2, task_view1));
//The visualization of file one can be started only after its transmission has finished
m.addConstraint(Choco.startsAfterEnd(task_view1, task_file1));
```

Of course we have to define some precedence constraints between the various tasks (Listing 5). We have added constraints "*startsBeforeEnd*" to establish the transmission order for the four transmission tasks. For example the second file starts before the end time of the first file, the third file starts before the end time of the second file and so on.

After that we have defined constraints "*startsAfterEnd*" in order to specify that the visualization tasks can be started only after the end of the transmission tasks. For example the first file can be viewed only after the finalization of its transmission.

Afterwards we have used the same constraint "*startsAfterEnd*" to establish the order of the visualization tasks (for example the second file can be visualized only after finishing the visualization of the first file). The same rule applies to the other files.

And finally we have minimized the final time of the latest visualization task because the goal is to finish viewing all four files as soon as possible.

4. Results

The problems described in the previous chapters have been implemented with the help of the Choco solver v.2.1.1 and the programming environment Eclipse Galileo.

For a constraint problem the most important aspect is the approach. First of all the problem should be understood very well and then it should be implemented using the constraints which describe the problem in a clear way so that the constraints do not overwrite/overlap or reduce the effect of each other. The search strategy should not be overlooked! A suited search strategy can reduce: the execution time, the number of expanded nodes, the number of backtracks.

One of the major challenges has been the total solving time for the scheduler. To show the power of the CSP approach, for the first problem the following example has been considered: a client requests 20 screws of type $R1$, 18 screws of type $R2$, 22 screws of type $R3$ and 16 screws of type $R4$.

Figure 2 displays the solving times corresponding to the manufacturing of the screws ordered by the client. We have displayed the solving times for the five best search strategies.

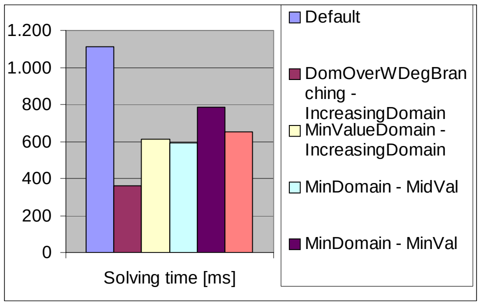

Fig. 2. *Solving times using various search strategies*

The best search (branching) strategy for this problem has been *DomOverWDegBranching* in combination with *IncreasingDomain* (a value iterator which selects the variable with the smallest value). It is known that *DomOverWDegBranching* is a n-ary branching assigning distinct values to an integer variable. Thus the best solving time was of 359 milliseconds. The legend displayed on the right hand side of Figure 2 always contains pairs of options. The first expression is the variable selector and the second one is either a value selector or a value iterator. The second best solving time has been obtained for the variable selector *MinDomain* and the value selector *MidVal* (594 seconds). Table 2 displays the planning results for manufacturing the order of the client. They show that the solver distributes optimally the work onto the three work stations, keeping them constantly occupied.

Results for the planning problem corresponding to an order of 76 screws Table 2

Screw type	Work station 1	Work station 2	Work station 3	Work station 4	Total number of screws
***R*1**	0	19	1	0	20
***R*2**	10	0	8	0	18
***R*3**	12	2	0	8	22
***R*4**	0	0	0	16	16
Execution time	**44**	**44**	**44**	**40**	**Total execution time (max): 44**

The results for the second problem may vary depending on the values set for file sizes, transmission speeds and visualization times. According to these values the data transmissions will follow one of the four patterns presented in Figure 1. The results presented in Table 3 correspond to the first case from Figure 1 where the transmissions of all four files are overlapped. We have seen that the visualization of a task is started as soon the transmission is finished. We have modified the visualization times of each file to an hour (60 minutes/5 = 12) and the obtained results are presented in Table 4. The results correspond to the third case presented in Figure 1 where at most the transmissions of two files are overlapped at a time.

Scheduling results Table 3

File	Transmission interval	Visualization interval
File 1	[0, 21]	[21, 27]
File 2	[4, 27]	[27, 33]
File 3	[10, 33]	[33, 39]
File 4	[19, 39]	[39, 45]

Table 4
Scheduling results for modified visualization durations

File	Transmission interval	Visualization interval
File 1	[0, 18]	[18, 30]
File 2	[12, 30]	[30, 42]
File 3	[23, 42]	[42, 54]
File 4	[36, 54]	[54, 66]

5. Conclusions

Constraint-based methods have proven successful when problems are hard (solutions not obvious, many hard constraints, strong constraint interactions), when domain-specific and redundant constraints are available, and when problems change often. The two different problems presented in the paper clearly show the difference between planning and scheduling problems. Both of them use timing information, but scheduling problems provide the exact moments in time when to start certain operations, while planning problems only provide information regarding the resources on which the operations should be executed. This is the main reason why planning problems are usually solved much faster than scheduling problems.

Section four has showed that a crucial part of the constraint-based approach, besides the correct definition of the model, and especially when the optimum solution is sought, is to choose the right search strategy. Considering the planning problem as an example, we have seen that the total execution time has been reduced by 67.23%.

Another advantage of the CSP approach has been proven through the scheduling problem. We have seen that a minor change in the input data has lead to a totally different schedule. This shows that once the model is correctly built, the solver finds a solution (the best solution for optimization problems) for any input values, which have been provided by the user.

An important aspect, which we have kept in mind throughout the development, is to build a CSP model which is scalable, i.e. to choose a very general approach. Both of the models described in the paper fulfill this requirement, e.g. one could easily model a planning problem with *n* workstations or with *n* files to be sent.

As a final conclusion we would say that complex planning/scheduling scenarios can be implemented readily through the open source Choco library. The solving times, especially of scheduling problems can become a problem for complex scenarios, but they can be greatly reduced through adequate search strategies.

Acknowledgment

This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD), financed from the European Social Fund and by the Romanian Government under the contract number POSDRU/88/1.5/S/59321.

References

1. Baker, K.R.: *Introduction to Sequencing and Scheduling*. New York. Wiley & Sons, 1974.
2. Baptiste, P., Le Pape, C., Nuijten, W.: *Incorporating Efficient Operations Research Algorithms in Constraint-based Scheduling*. In: Workshop on Artificial Intelligence and Operations Research, Timberline Lodge, Oregon, June 6-10, 1995, p. 13-19.
3. Baptiste, P., Le Pape, C., Nuijten, W.: *Constraint-Based Optimization and Approximation for Job-Shop Scheduling*. In: SIGMAN Workshop on Intelligent Manufacturing Systems, Montreal, Canada, August 20-25, 1995, p. 21-27.
4. Benavides, D., Segura, S., Trinidad, P., Ruiz-Cortes, A.: *Using Java CSP Solvers in the Automated Analyses of Feature Models*. In: Generative and Transformational Techniques in Software Engineering, Braga, Portugal, July 4-8, 2005, p. 399-408.
5. Fromherz, M.P.J., Carlson, B.: *Optimal Incremental and Anytime Scheduling*. In: Workshop on Constraint Languages/ Systems and their Use in Problem Modeling, Syracuse, N.Y., USA, Feb., 1994, p. 45-59.
6. Fromherz, M.P.J., Saraswat, V.A., Bobrow, D.G.: *Model-based Computing: Developing Flexible Machine Control Software*. In: AI Journal **114** (1999) No. 1-2, p. 157-202.
7. Girbea, A., Suciu, C., Sisak, F.: *Design and Implementation of a Fully Automated Planner-Scheduler Constraint Satisfaction Problem.* In: Proceedings of the 6th IEEE International Symposium on Applied Computational Intelligence and Informatics, Timişoara, Romania, May 19-21, 2011, p. 54-59.
8. Hentenryck, P.: *Constraint Satisfaction in Logic Programming*. Cambridge. MIT Press, 1989.
9. Hentenryck, P.: *The OPL Optimization Programming Language*. Cambridge. MIT Press, 1999.
10. Mahnke, W., Leitner, S.H., Damm, M.: *OPC Unified Architecture*. Berlin. Springer Press, 2009.
11. Zweben, M., Fox, M.: *Intelligent Scheduling*. Burlington. Morgan Kaufman, 1994.
12. http://choco.emn.fr. Accessed: 06-12-2010.
13. http://jacop.osolpro.com/. Accessed: 12-11-2010.