10.1515/acsc-2015-0024

Archives of Control Sciences
Volume 25(LXI), 2015
No. 3, *pages 367–375*

A Two-stage approach for an optimum solution of the car assembly scheduling problem. Part 2. CLP solution and real-world example

MICHAŁ MAZUR and ANTONI NIEDERLIŃSKI

A *Constraint Logic Programming* (CLP) tool for solving the problem discussed in Part 1 of the paper has been designed. It is outlined and discussed in the paper. The program has been used for solving a real-world car assembly scheduling problem.

Key words: car sequencing, car assembly scheduling, workstation capacity constraints, constraint logic programming.

1. Why use CLP?

The problem presented in the first part of the paper (M. Mazur, A. Niederliński, "A Two-Stage Approach for an Optimum Solution of the Car Assembly Scheduling Problem, Part 1. Problem statement, solution outline and tutorial example") has been solved using Constraint Logic Programming (CLP). It is a declarative programming tool, based on ideas first presented in Prolog, and used for for solving constraint satisfaction problems (CSP). For the important combinatorial case CSP are characterized by following features (see [2], [3], [8]):

- a finite set S of integer variables $X_1,...,X_n$, with values from finite *domains* $D_1,...,D_n$;
- a set of constraints between variables. The *i-th constraint* $C_i(X_{i_1},...,X_{i_k})$ between k variables from S is given by a *relation* defined as subset of the Cartesian product $D_{i_1}\times,...,\times D_{i_k}$ that determines variable values *corresponding* to each other in a sense defined by the problem considered. Quite often the constraints may not be stated as relations, but by equations, inequalities, subroutines etc.;
- a CSP *solution* is given by any assignment of domain values to variables that satisfies all constraints. It may be *non-unique* or *unique*;

M. Mazur is with General Motors Manufacturing Poland, Gliwice. A. Niederliński, the corresponding author, is with University of Economics in Katowice, Poland. e-mail: antoni.niederlinski@ae.katowice.pl

Received 28.06.2015.

- a CSP *solution* may additionally minimize or maximize an *objective function*. Then it is usually referred to as *constraint optimization problem* (COP), and its solution as *optimum* solution.

A basic notion of CLP is that of a *predicate*. A predicate is a relation between ordered variables referred to as arguments, declared by naming it, naming the variables, arranging their order and defining them either by other predicates or by declaring their domains. Predicates are used to express constraints. They may be either *built-in*, i.e. be part of the CLP language, or be *user-defined*.

A salient feature of combinatorial CSP and COP is that all variables take values from finite domains. Hence they are referred to as *domain variables*. It follows that in theory any CSP and COP can either be shown to have no solution or be solved using an algorithmically simple exhaustive search or direct enumeration approach. Therefore the wisdom of developing special tools for such problems may be questioned. Why are present-day tools for solving combinatorial CSP and COP better than exhaustive search? The answer to this question is as follows:

- Because of the numerical effectiveness of determining CSP and COP solutions, which for exhaustive search and large numbers of variables is very bad indeed: the number of enumerations needed to get those solutions may be exorbitant. CLP copes (to some extent, not entirely) with such problems by early and judicious use of problem constraints, and by using implicit feasible problem-specific heuristics, in order to substantially decrease the search space.

- Because of the *declarativity* of CLP programs, which means that a properly formalized description of the solved problem is tantamount to the program solving the problem. It is contrasted with imperativity (procedurality) based on designing algorithms needed to solve problems. Declarativity means further that while using CLP languages no algorithms for problem solving need to be designed. The algorithms, which are of course necessary for any computer-based problem solving, have been embedded into CLP compilers. To simplify a bit, it may be stated that the art of CLP consists in designing such problem descriptions that are understood by CLP language compilers, and that ensure an effcient determination of the solution.

2. Formulating a car assembly ontology

The first step in designing a CLP solution for a complicated combinatorial problem like scheduling car assembly lines consists of formulating a car assembly *ontology*, see [7]. The ontology is a naming of all entities, ideas, and events, and precise defining of their properties and relations, as needed to describe car assembly lines. The proposed ontology covers over `170` entities, ideas, and events, only some of them will be used below.

The need for the ontology arises from the obvious circumstance, that humans performing car assembly scheduling use a lot of tacit and often vague background knowledge, which - for the sake of computerizing the scheduling process - must be put on a precise footing.

3. Choosing global constraints

The next step consists of choosing a set of *global constraints* suitable for modelling the car assembly line. Global constraints (see [1], [5] and [8]) are built-in predicates, defining advanced relations capturing sometimes tens of thousands of code lines in imperative programming languages. The following global predicates are particularly useful:

- The *among()* predicate which ensures that a desired number of particular cars will appear in chosen subsequences of the car sequence. This predicate is responsible for enforcing in the schedule the workstation capacity constraints. Two versions of this predicate are used in the program:

  1. `among (N,[X1,..,Xs],[C1,..,Cs],[V1,..,Vm],ground)`
  where `N` is a natural number or domain variable, `[X1,..,Xs]` is a list of domain variables, `[C1,..,Cs]` is a list of natural numbers, `[V1,..,Vm]` is a list of natural numbers in ascending order, `Vi<V(i+1)`. The constraint is fulfilled if `[X1,...,Xs]` contains `N` such values `Xi`, for which `Xi+Ci` belong to list `[V1,...,Vm]`. E.g. the constraint:

  ```
  among(2,[S1,S2,S3,S4,S5,S6,S7,S8,S9,S10],
            [0,0,0,0,0,0,0,0,0,0],[5],ground)
  ```

  is fulfilled if car model `5` appears `2` times in the list of produced car models `[S1,S2,S3,S4,S5,S6,S7,S8,S9,S10]`.

  2. `among([Low,Up,Seq,Least, Most], [X1,..,Xs],`
  `[C1,..,Cs], [V1,..,Vm], ground)`
  where `Low` is a nonnegative integer, `Up` is a positive integer, `Seq`$\leqslant$ `s`. The constraint is fulfilled if at least `Low` integers and at most `Up` integers of each `Seq` consecutive integers from the list `[X1+C1, .. , Xs+Cs]` belong to list `[V1,..,Vm]`, and that at least `Least` and at most `Most` of the integers in the list `[X1+C1, .. , Xs+Cs]` are equal to integers from the list `[V1,..,Vm]`. E.g. the constraint:

  ```
  among([0,1,5,2,2],[S1,S2,S3,S4,S5,S6,S7,S8,S9,S10],
                [0,0,0,0,0,0,0,0,0,0],[3],ground)
  ```

is fulfilled if at least none and at most one car of each `5` consecutive cars should be car model `3`, and that at least `2` and at most `2` of the car models in the list `[S1,S2,S3,S4,S5,S6,S7,S8,S9,S10]` should be car model `3`.

- The `disjunctive()` predicate is used to enforce non-overlapping of time segments of assembly operations at the same workstation for different car models:

```
disjunctive(Time_segments, End),
```

where

```
Time_segments =
    [[Segment_1_Start_Time, Segment_1_Duration],...
       ...[Segment_n_Start_Time, Segment_n_Duration]]
```

is a list of lists and `End` is an integer representing a coordinate not to be occupied by time segments. The disjunctive constraint is fulfilled if there is no time overlap of operations with start times `Segment_i_Start_Time` (i=1,..,n) and corresponding durations `Segment_i_Duration` (i=1,..,n). Any workstation must be represented in the CLP program by its own `disjunctive()` predicate.

Its usage is presented in Fig. 1 for three time segments `A`, `B` and `C`.

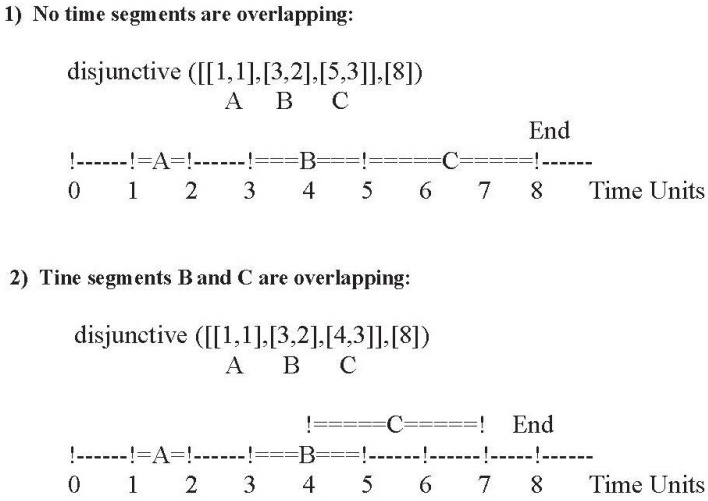

Figure 1: Examples of disjunctive() usage

- The minimizing predicate `minimize(labeling(ENDS),MAKESPAN)` is used to search for a set of end times for all car bodies (in the list ENDS) that

minimizes the MAKESPAN. The end times (being the proper decision variables of the mimimization) are searched by a labeling predicate that implements the *first fail* search strategy. This strategy always starts with instantiating the variable with the smallest domain.

The predicate is generating a list of end times for all cars (which is determinig backwards the entire assembly schedule) together with the minimum makespan.

The bulk of the program consists of inequality constraints defining the assembly technbology: the assembly of the next car at any station may start no sooner than the assembly of the present car at that station is done with.

4. A real-world example

To illustrate the power of the proposed approach, the CLP program developed has been applied to a following real-world example.

4.1. Determining admissible car body sequences

The investigated production plan consists of `30` cars to be produced in `6` groups of options (`Group 1, 2,.., 6`), with `5` different options (Option O_1,...,Option O_5). A production plan for 30 cars (if all cars had different options) results in 30! = $2.65 * 10^{32}$ sequences. Taking into account the groups of options results in a "smaller" number of sequences equal to $4.23 * 10^{19}$. It is assumed that capacity constraints for needed options are given by Tab. 8.

Table 8: Workstation capacity constraints for real-world example

Group	1	2	3	4	5	6	$WCCO_i$
Option O_1	0	0	0	0	0	1	2/27
Option O_2	1	0	0	4	0	0	3/9
Option O_3	0	1	0	0	1	0	4/10
Option O_4	1	0	1	0	1	0	3/5
Option O_5	0	0	1	0	0	1	3/11
Number of cars in production plan	5	5	5	5	7	3	

The meaning of this table is as follows: e.g. any of the 7 cars belonging to group 5 should be outfitted with option O_3 with $WCCO_3 = 4/10$, and option O_4 with $WCCO_4 = 3/5$.

4.2. Sequencing results

Taking into account the capacity constraints, 239 admissible sequences are generated using the `among()` predicates. The reduction of the search space form $4.23 * 10^{19}$ points to 239 points is thus quite dramatic. The two first and two last admissible sequences named by their `S Number` are listed in Tab. 9.

Table 9: Two first and two last admissible sequences

$S001$	$[6,1,1,2,5,2,5,3,4,6,1,3,1,2,2,5,5,4,3,4,3,4,3,5,2,5,4,5,1,6]$
$S002$	$[6,1,1,5,2,2,5,3,4,6,1,3,1,2,2,5,5,4,3,4,3,4,3,5,2,5,4,5,1,6]$
.....	.....
$S238$	$[6,1,5,4,5,2,5,3,4,3,4,3,1,2,2,5,5,4,3,4,6,1,3,5,2,2,5,1,1,6]$
$S239$	$[6,1,5,4,5,2,5,3,4,3,4,3,1,2,5,2,5,4,3,4,6,1,3,5,2,2,5,1,1,6]$

The meaning of this table is as follows: e.g. for the admissible sequence $S001$ the first car body fed into the line is from group 6, the next - from group 1, the next - from group 5, and so on, from the right end of the sequence to the left end.

4.3. Scheduling constraints

It is assumed that there are 50 workstations on the assembly line. To each group of car bodies a value of duration time of operations in each workstation has been attributed as shown in Tab. 10. It is assumed that for the entire production plan the duration time of all operations is in the range [3..7] minutes. The time is measured in Time units. Practically a Time Unit is different for different automakers and usually may correspond to values from 70 to 120 seconds.

The meaning of this table is as follows: e.g. for workstation 3 cars from group 1 are handled 6 minutes, from group 2 are handled 7 minutes, from group 3 are handled 6 minutes, and so on.

The tact time is assumed to be `TT=6`, i.e. it is smaller than the maximum duration time for some operation (equal 7). In order to perform all assembly operations it must be thus assumed, that operators manning any workstation `i` can start assembly operations in advance on the car body in the previous workstation `i-1`. To make this assumption precise it is further assumed that for the `Advance Time` holds `AT <= 5` minutes. This is a difficult constraint to handle, but precisely this constraint results in different admissible sequences having different makespans.

Table 10: Duration time of assembly operation

Workstation number	Group 1 cars 1–5	Group 2 cars 6–10	Group 3 cars 11–15	Group 4 cars 16–20	Group 5 cars 21–27	Group 6 cars 28–30
1	5	5	7	6	5	5
2	5	4	6	7	5	7
3	6	7	6	5	7	5
4	4	5	4	6	7	5
5	4	4	7	5	5	5
6	5	5	4	6	4	5
7	4	6	6	4	4	4
8	7	5	4	5	7	6
9	4	6	6	6	5	4
10	4	5	7	5	4	5
11	4	6	4	7	7	4
12	4	5	4	5	7	4
13	7	4	5	4	4	4
14	6	5	4	6	6	6
15	5	4	4	6	6	4
16	6	5	5	4	5	7
17	5	6	4	6	5	7
18	6	4	5	4	7	6
19	4	5	5	6	5	5
20	4	7	7	5	7	5
21	5	4	7	4	5	4
22	5	4	7	6	7	6
23	4	6	5	4	5	5
24	4	7	4	7	7	6
25	6	7	4	5	6	4
26	5	4	7	6	5	6
27	4	7	7	5	6	6
28	6	5	5	4	5	7
29	7	5	6	4	4	5
30	6	4	7	5	4	7
31	5	7	5	6	5	6
32	5	7	5	6	7	4
33	7	7	4	6	6	7
34	4	5	5	7	6	5
35	4	6	4	5	7	7
36	4	7	5	5	6	7
37	5	7	5	5	6	6
38	7	5	5	6	7	6
39	4	4	7	6	6	6
40	4	5	4	6	6	7
41	5	7	6	4	6	4
42	6	7	4	5	5	7
43	5	6	4	5	7	6
44	7	4	6	5	4	4
45	4	4	5	5	4	4
46	6	6	7	5	5	5
47	4	4	6	6	7	4
48	6	6	5	5	7	5
49	5	7	6	7	7	5
50	7	5	6	4	4	7

4.4. Scheduling results

The assembling for all 239 admissible sequences was simulated and the corresponding makespans determined. The results are presented in Fig. 2. It can be seen that

makespans for different admissible sequences may vary between 510 and 521 TU, with two sequences having the minimum makespan 510.

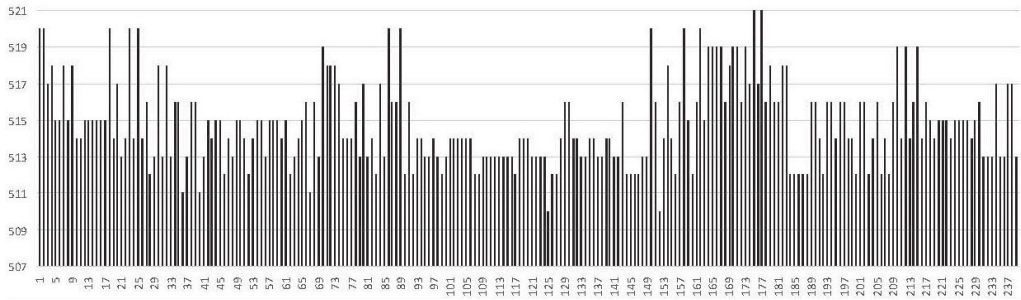

Figure 2: Makespans for admissible sequences

Fig. 3 presents the distribution of makespans for the entire set of admissible sequences.

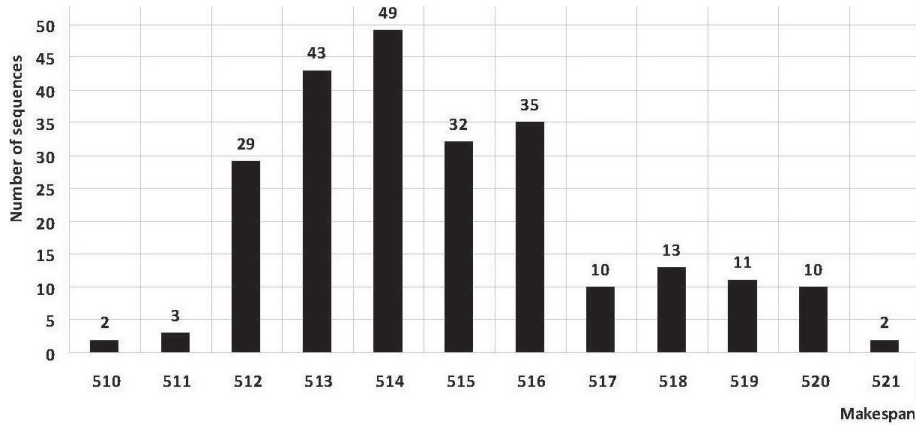

Figure 3: Distribution of makespans for admissible sequences

5. Summary

A two stage decomposition for determining minimum makespan schedules for car assembly lines using a CLP approach has been presented and applied to a nontrivial real-world. The most important global constraints used in the CLP program have been discussed.

References

[1] A. AGGOUN and N. BALDICEANU: Extending CHIP in Order to Solve Complex Scheduling and Placement Problems. *Mathematical and Computer Modelling*, **17**(7), (1993), 57-73.

[2] K.R. APT: Principles of Constraint Programming. Cambridge University Press, Cambridge, 2003.

[3] K.R. APT and M.G. WALLACE: Constraint Logic Programming using ECL^iPS^e. Cambridge University Press, Cambridge, 2007.

[4] N. BALDICEANU and E. CONTEJEAN: Introducing Global Constraints in CHIP. *Mathematical and Computer Modelling*, **20**(12), (1994), 97-123.

[5] N. BALDICEANU: Global Constraints Catalog. http://www.emn.fr/x-info/sdemasse/gccat/, 2010.

[6] M. DINCBAS, H. SIMONIS and P. VAN HENTENRYCK: Solving the car-sequencing problem in constraint logic programming. *Proc. of the ECAI*, (1988), 290-295.

[7] M. MAZUR: Ontologia procesu wyznaczania harmonogramu optymalnego dla montażu samochodów (Car Assembly Ontology). In K. Malinowski, J. Józefczyk and J. Świątek: Aktualne problemy automatyki, EXIT, Warszawa, 2014, 716-726, (in Polish).

[8] A. NIEDERLIŃSKI: A Gentle Guide to Constraint Logic Programming via ECL^iPS^e. PKJS, Gliwice, 3-rd edition, revised and expanded, 2014.