Cell formation in group technology using constraint programming and Boolean satisfiability

Ricardo Soto, Hakan Kjellerstrand, Orlando Durán, Broderick Crawford, Eric Monfroy, Fernando Paredes

1. Introduction

Group technology (GT) is a common methodology for increasing productivity in manufacturing systems. Cell formation is a key step in GT, it consists in dividing a plant into a set of cells, each of them containing machines that process similar types or families of parts. The idea is to minimize the part flow among cells in order to reduce costs and increase productivity. The literature presents different approaches devoted to solve this problem, which are mainly based on mathematical programming and on evolutionary computing. Mathematical programming can guarantee a global optimal solution, however at a higher computational cost than an evolutionary algorithm, which can assure a good enough optimum in a fixed amount of time. In this paper, we model and solve this problem by using state-of-the-art constraint programming (CP) techniques and Boolean satisfiability (SAT) technology. We present different experimental results that demonstrate the efficiency of the proposed optimization models. Indeed, CP and SAT implementations are able to reach the global optima in all tested instances and in competitive runtime.

2. Related work

This paper focuses on the efficient solving of manufacturing cell formation problems. We introduce constraint programming (CP) and Boolean satisfiability (SAT) as powerful techniques to tackle this problem. We implement five different models by using five different global optimizers: two CP-based solvers, two SAT-based solvers, and a hybrid CP+SAT solver. Those implementations demonstrate the efficiency and feasibility of using this technology for solving cell formation problems. Indeed, all implementations are able to reach the global optimum in all instances and most of them in excellent runtime. This paper is organized as follows: Sections 3 and 4 give an overview of constraint programming and SAT, respectively. The cell formation problem is described and modeled in Section 5. Experimental results are presented and discussed in Section 6. Finally, we conclude and we give some directions for future work.
search space to guarantee a global optimum, as a consequence the computational cost in terms of memory and time consumed is commonly higher.

Within the first group, different metaheuristics have been used for cell formation, for instance Aljaber, Baek, and Chen (1997) and Lozano, Díaz, Eguía, and Onieva (1999) use Tabu Search, Wu, Chang, and Chung (2008) presents a simulated annealing (SA) approach, Durán, Rodríguez, and Consalter (2010) combines particle swarm optimization with a data mining technique, and Venugopal and Narendran (1992) propose the use of genetic algorithms (GA). In Gupta, Gupta, Kumar, and Sundaram (1996), the same genetic representation of solutions as Venugopal and Narendran (1992) is used, but a different multi-objective optimization approach is followed for the simultaneous minimization of the total number of intercell-intracell moves and within-cell load variation. Hybridization techniques can also be found in the literature. For instance, Wu et al. (2008) combined SA and GA, James, Brown, and Keeling (2007) presented a hybrid solution including local search and GA, and Nsakanda, Diaby, and Price (2006) proposed a solution methodology based on a combination of a GA and large-scale optimization techniques.

On the second group, preliminary experiments for cell formation were performed by using linear programming, some examples are the works of Purcheck (1975) and Olivia-Lopez and Purcheck (1979). Linear quadratic models have also been proposed, for instance by Kusiak and Chow (1987) and Docter (1991). Goal programming (GP) is another paradigm for global optimization. It can be seen as a generalization of linear programming, for handling multiple objective functions. Some GP-models have been proposed by Sankaran (1990) and Shafer and Rogers (1991). Hybridizations of global optimization techniques and metaheuristics can also be found in this group, for instance Boulif and Atif (2006) combines GP and SA, Wu et al. (2008) combined SA and GA, James, Brown, and Keeling (2007) presented a hybrid solution including local search and GA, and Nsakanda, Diaby, and Price (2006) proposed a solution methodology based on a combination of a GA and large-scale optimization techniques.

3. Constraint programming

Constraint programming is a powerful programming paradigm devoted to the efficient resolution of constraint-based problems. It draws on methods from operational research, numerical analysis, artificial intelligence, and programming languages. Currently, CP is widely used in different application areas, for instance, in computer graphics to express geometric coherence, in engineering design for the conception of complex mechanical structures, in database systems to ensure and/or restore data consistency, in electrical engineering to locate faults, and even for sequencing the DNA in molecular biology (Rossi, 2006). In CP, a problem is formulated as a constraint satisfaction problem (CSP). This representation mainly consists in a sequence of variables lying in a domain, and a set of relations over such variables, namely the constraints. Formally, a CSP $\mathcal{P}$ is defined by a 4-tuple $\mathcal{P} = (X, D, C, f)$, where $f$ is the objective function to be maximized or minimized.

3.1. CSP solving

The basic CP idea for solving CSPs is to build a tree-data structure holding the potential solutions by interleaving two main phases: enumeration and propagation. In the enumeration phase, a variable and a value from its domain are chosen to create a tree branch. In the propagation phase, a consistency property is enforced to prune the tree, i.e., the values that do not lead to any solution are temporarily deleted from domains (see Bessière (2006, chap. Constraint propagation) for a detailed description about constraint propagation). In this way, the exploration does not inspect infeasible instantiations accelerating the whole process.

Algorithm 1 (see Fig. 1) illustrates a general procedure for solving CSPs. The goal is to iteratively generate partial solutions, backtracking when an inconsistency is detected, until a result is reached. The algorithm has as input the set of constraints and domains. Then, a while loop encloses a set of actions to be performed until success (i.e. a solution is reached) or a failure is detected (i.e. no solution is found). The first two enclosed actions correspond to the variable and value selection. The third action is a call to a propagation procedure, which is responsible for attempting to prune the tree. Finally two conditions are included to perform backtracks. A shallow backtrack corresponds to try the next value available from the domain of the current variable, and the backtracking returns to the most recently instantiated variable that has still values to reach a solution.

Algorithm 2 (see Fig. 1) is a CP-based branch and bound algorithm for handling COPs, in particular minimization problems (maximization problems are handled similarly). It is a slight modification of algorithm 1. It includes a cost function as an additional input and maintains an upper bound on the global minimum in the variable $m$, which is initialized to $+\infty$ at the beginning of the search. This upper bound is used to discard parts of the search space whose cost is larger than it by adding $f(x) \leq m$ to the set of constraints. The idea is to propagate involving the cost function (line 5). Finally, the upper bound is updated whether a better solution has been found (line 6).

4. SAT solving

SAT solvers are also propagation-based solvers, but devoted to Boolean variables and clause constraints. Most modern SAT solvers are based on the Davis–Putnam–Logemann–Loveland (DPLL) algorithm (Davis, Logemann, & Loveland, 1962), which also interleaves two phases: enumeration and propagation (so called unit propagation). They incorporate sophisticated engineering to (1) achieve a fast constraint propagation, to (2) record as nogoods part of the search that lead to failure, and (3) to automate the search by tracking how often a variable is part of the reason for causing failure (activity) and concentrating search on variables with high activity. Some SAT solvers frequently restart the search from scratch relying on nogoods recording to prevent repeated search, and activity to drive the search into more profitable areas (see Eén & Sörensson (2003) for a gentle presentation of modern SAT solving).

Fig. 2 depicts a rough architecture of a modern SAT solver. The search starts the unit propagation process which interacts with the clause database. When a failure is detected, conflict analysis is initiated. Conflict analysis uses the graph of explanations to construct a nogood which is a resolvent of clauses causing the failure.

This is stored in the clause database and causes search to backtrack. It prevents the search revisiting the same set of decisions. Activity counters (not detailed in figure) record which variables
are most responsible for failure, these are the variables chosen for enumeration by the search.

5. Problem statement

In this work, we model the cell formation problem by using an array-based clustering approach. The idea is to represent the processing requirements of parts on machines through an incidence matrix named machine-part. This matrix holds binary domains, and is denoted as $A = [a_{ij}]$, where

$$a_{ij} = \begin{cases} 1 & \text{if } j \text{th part visits } i \text{th machine;} \\ 0 & \text{otherwise.} \end{cases}$$

Let us note that when a machine-part incidence matrix is constructed, no machine groups or part of families are easily visible. The main objective is machine grouping is the formation of set of machines and workpieces in groups so that the number of intercell transportation of pieces is minimized. Therefore, the initial (left matrix in Fig. 3) has to be transformed into a matrix that has a block diagonal structure (right matrix in Fig. 3). This rearrangement aims at minimization of total intercell moves and of within-cell load variation. A rigorous mathematical formulation of machine-component grouping problem with these objectives is given by Boctor (1991).

The optimization model is stated as follows. Let:

- $M$, the number of machines,
- $P$, the number of parts,
- $C$, the number of cells,
- $i$, the index of machines ($i = 1, \ldots, M$),
- $j$, the index of parts ($j = 1, \ldots, P$),
- $k$, the index of cells ($k = 1, \ldots, C$),
- $A = [a_{ij}]$ the $M \times P$ binary machine-part incidence matrix,
- $M_{\text{max}}$, the maximum number of machines per cell.

We selected as the objective function to be minimized the number of times that a given part must be processed by a machine that does not belong to the cell that the part has been assigned to. Let:

$$y_{ik} = \begin{cases} 1 & \text{if machine } i \in \text{cell } k; \\ 0 & \text{otherwise;} \end{cases}$$

$$z_{jk} = \begin{cases} 1 & \text{if part } j \in \text{family } k; \\ 0 & \text{otherwise;} \end{cases}$$

The problem is represented by the following mathematical model:

$$\text{minimize } \sum_{k=1}^C \sum_{i=1}^M \sum_{j=1}^P a_{ij}z_{jk}(1 - y_{ik}).$$

Subject to

$$\sum_{k=1}^C y_{ik} = 1 \quad \forall i, \quad \sum_{k=1}^C z_{jk} = 1 \quad \forall j, \quad \sum_{i=1}^M y_{ik} \leq M_{\text{max}} \quad \forall k.$$
Table 1
S solver description.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FznTini</td>
<td>It is a general purpose constraint solver based on SAT (Boolean satisfiability problem) (Gomes, Kautz, Sabharwal, &amp; Selma, 2008, chap. Satisfiability solvers). Models are written in MiniZinc/FlatZinc (Mariott et al., 2008) and then launched in Tinsat via Booleanization. Details about FznTini and the transformation process into Tinsat can be seen in Huang (2008).</td>
</tr>
<tr>
<td>Fzn2smt</td>
<td>It (Bofill, Suy, &amp; Villaret, 2010) is a general purpose constraint solver based on SMT (Satisfiability Modulo Theories) (Nieuwenhuis &amp; Oliveras, 2006) working under a similar principle that FznTini. Fzn2smt compiles models from the FlatZinc language to the standard SMT-LIB language. Thus, models can be solved in any SMT solver, working by default with Yices 2 (Dutertre &amp; de Moura, 2006).</td>
</tr>
<tr>
<td>Gecode</td>
<td>It is a well-known constraint programming solver. It has been implemented as a C++ library, but currently can be interfaced to several languages, among others, MiniZinc, Alice, Ruby, and Lisp. Implementation details of Gecode can be seen in Schulte and Tack (2006).</td>
</tr>
<tr>
<td>Eclips</td>
<td>It is another well-known constraint (logic) programming solver. It has been implemented on top of Prolog, and extended along the years with several specific CP primitives and features. Several academic publications involve Eclips (Puget, 1994).</td>
</tr>
<tr>
<td>Chuffed</td>
<td>It is a lazy clause solver, which can be seen as a hybrid that combines some of the advantages of CP (high level modeling and programmable search) with some of the advantages of SAT solvers (reduced search by nogood creation, and effective autonomous search). Details of its implementation can be seen in Feydy and Stuckey (2009).</td>
</tr>
</tbody>
</table>

Table 2
Variable and value ordering heuristics used.

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anti-first-fail</td>
<td>Selects the variable with the biggest domain size</td>
</tr>
<tr>
<td>occurrence</td>
<td>Selects the variable with the largest number of attached constraints</td>
</tr>
<tr>
<td>min max</td>
<td>Selects the minimum/maximum value of the domain</td>
</tr>
</tbody>
</table>

our CP/SAT implementation, column 3 (Sa) the optimum value using simulated annealing, and column 4 (Pso) the optimum value using particle swarm optimization. Here, it is possible to see that the proposed models are unique in reaching the global optimum for all instances.

Table 3
Different models implemented.

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCDP1</td>
<td>It exactly corresponds to the original mathematical programming model presented in Section 5.</td>
</tr>
<tr>
<td>MCDP2</td>
<td>It is an extension of MCDP1 that includes a dual representation of the problem. Two additional arrays are added in order to identify the machine assignments (i.e. which machines have been assigned to the ( y_{ik} ) matrix) as well as the part assignments (i.e. which parts have been assigned to the ( z_{ik} ) matrix).</td>
</tr>
<tr>
<td>MCDP3</td>
<td>This model skips the initial matrix representation of MCDP1 and MCDP2. Decision variables are now completely handled via lists including the dual representation of MCDP2. This allows one to introduce an extra heuristic for the constraints that ensure that a part should be in the same cell as one of its machines, and vice versa.</td>
</tr>
<tr>
<td>MCDP4</td>
<td>This model is an extension of MCDP3 that includes symmetry breaking (Gent &amp; Smith, 2000) in order to reduce the search space. The idea is to force using cells in order e.g., if there is a machine assigned to cell ( k ), then there must be a machine assigned to cell ( k - 1 ).</td>
</tr>
<tr>
<td>MCDP5</td>
<td>This model implements sets instead of single decision variables. It maintains the dual representation of MCDP2 and the extra heuristic of MCDP3. Additionally, it involves the use of the ( \text{partition}^a ) and ( \text{all_different}^b ) global constraints for speeding up the solving process.</td>
</tr>
</tbody>
</table>

\(^a\) The \( \text{partition}^a \) global constraint forces to partition a given universe into disjoint sets.

\(^b\) The \( \text{all_different}(X_1, \ldots, X_n) \) constraint specifies that the values assigned to the variables \( X_1, \ldots, X_n \) must be pairwise distinct (Régis, 1994).
use of the dual representation and the extra heuristic (included in MCDP3, MCDP4, and MCDP5) allow a better propagation process, which positively impact the solving time. In the case of Fzn2Smt, Gecode/fz and Eclips2, the propagation process is boosted with the use of symmetry breaking (included in MCDP4 and MCDP5).

7. Conclusions

In this paper, we have modeled and solved the cell formation problem by using state-of-the-art CP and SAT technology. We have implemented five different models and we have tested them in five solvers: two CP solvers, two SAT solvers, and a hybrid SAT+CP solver. The results demonstrated the feasibility of using this approach, as well as the good performance of the proposed models. Indeed, the global optimum was reached in all instances and in excellent runtime.

The results illustrated here can clearly be extended by enhancing the solving engines with autonomous search. This will allow an automatic configuration of enumeration strategies with the aim of enhancing even more the propagation and as a consequence to obtain better solving processes.

References


