A Global Constraint for Bin-Packing with Precedences: Application to the Assembly Line Balancing Problem.

Pierre Schaus and **Yves Deville**
Department of Computing Science and Engineering,
University of Louvain,
Place Sainte Barbe 2,
B-1348 Louvain-la-Neuve, Belgium
{pierre.schaus,yves.deville}@uclouvain.be

Abstract

Assembly line balancing problems (ALBP) are of capital importance for the industry since the first assembly line for the Ford T by Henry Ford. Their objective is to optimize the design of production lines while satisfying the various constraints. Precedence constraints among the tasks are always present in ALBP. The objective is then to place the tasks among various workstations such that the production rate is maximized. This problem can be modeled as a bin packing problem with precedence constraints (BPPC) where the bins are the workstations and the items are the tasks. Paul Shaw introduced a global constraint for bin-packing (without precedence). Unfortunately this constraint does not capture the precedence constraints of BPPC. In this paper, we first introduce redundant constraints for BPPC combining the precedences and the bin-packing, allowing to solve instances which are otherwise intractable in constraint programming. We also design a global constraint for BPPC, introducing even more pruning in the search tree. We finally used our CP model for BPPC to solve ALBP. We propose two search heuristics, and show the efficiency of our approach on standard ALBP benchmarks. Compared to standard non CP approaches, our method is more flexible as it can handle new constraints that might appear in real applications.

Introduction

In a variant of the bin packing problem, items of different volume must be packed into a finite number of bins with a fixed capacity in a way that balances the load of the different bins (e.g. minimizes the maximum load of the bins). We are interested in bin packing problems with precedence constraints between items (BPPC). Here the bins are ordered. A precedence constraint between items a_1 and a_2 is satisfied if item a_1 is placed in a bin B_1, and item a_2 in a bin B_2, with $B_1 \leq B_2$.

BPPC occurs frequently in the design of assembly lines in the industry[1]. We are given a set of tasks of various lengths, subject to precedence constraints, and a time constant called cycle time. The problem is to distribute the tasks over workstations along a production (assembly) line, so that no workstation takes longer than the cycle time to complete all the tasks assigned to it (station time), and the precedence constraints are satisfied. The decision problem of optimally partitioning (balancing) the tasks among the stations with respect to some objective is called the assembly line balancing problem (ALBP) (Boysen, Fliedner, & Scholl 2007; Scholl & Becker 2006).

In particular, when the number of stations is fixed, the problem is to distribute the tasks to stations such that the station time is balanced, that is to minimize the cycle time. This type of problems is usually called Simple ALBP-2 (SALBP-2) (Boysen, Fliedner, & Scholl 2007). It is clear that SALBP-2 and BPPC are equivalent.

BPPC and SALBP-2 can be solved by exact or heuristic methods. In this paper, we are interested in *exact* methods. Existing exact methods are usually dedicated branch and bound algorithms such as Salome 2 (Klein & Scholl 1996). These algorithms are very efficient and have been improved since about 50 years. Unfortunately these algorithms are not flexible to new constraints that might appear in real applications such as minimal distance between two tasks or restriction on the cumulated value of a particular task attribute (see the problem classifier available on *www.assembly-line-balancing.de* for more details). When such constraints are added, we obtain so-called Generalized Assembly Line Balancing Problems (GALBP) (Becker & Scholl 2006). The existing exact methods are not flexible enough to handle GALBP efficiently. The Constraint Programming (CP) paradigm is a good candidate to tackle such problems since constraints can be added very easily to the model.

The Constraint Programming (CP) framework has already been used for the bin packing problem (e.g. (Shaw 2004)). The Balanced Academic Curriculum Problem (BACP) is equivalent to BPPC. The objective is to schedule courses into a given number of periods such that the prerequisites relations between the courses are satisfied and such that the workloads among the periods are well balanced. Basic CP models have been proposed in (Castro & Manzano 2001; Hnich, Kiziltan, & Walsh 2002).

In this paper, we propose a CP model for BPPC and SALBP-2, allowing a flexible expression of new constraints, such as in GALBP. More specifically, our contributions are :

- An efficient and simple CP model for the bin packing with precedence constraints (BPPC).

[1]See for example the two commercial softwares Proplanner® *www.proplanner.com* and OptiLine® *www.optimaldesign.com*

- A new global constraint for BPPC. This constraint is based on set variables, and exploit the transitive closure of the precedence graph. An $O(n^2)$ filtering algorithm is presented, where n is the number of items (tasks). This constraint allows to solve instances which are otherwise intractable in CP.
- An experimental validation on standard SALBP-2 benchmarks, showing the feasibility, the efficiency, and the flexibility of this approach.

The paper is structured as follows. A background on constraint programming is first given, followed by a CP model for BPPC. A new global constraint and its associated filtering algorithm is then described. Before concluding this paper, an experimental section analyzes the performance of our approach on standard SABLP benchmarks.

CP Background

Constraint Programming is a powerful paradigm for solving Combinatorial Search Problems (CSP). A CSP is composed of a set of variables; each variable having a finite domain of possible values, and a set of constraints on the variables. The objective is to find an assignment of the variables that satisfies all the constraints. An objective function may also be added. CP interleaves a search process (backtracking or branch-and-bound) with inference (also called propagation or filtering), aiming at reducing the search space by removing values that cannot belong to any solution.

Given a finite domain integer variable X with domain $Dom(X)$, we denote by $X^{\min}$ and $X^{\max}$ the minimum and maximum value of its domain. Set variables in CP (Gervet 1993) allow to represent a set rather than a single value. The domain of a set variable $\mathcal{S}$ is represented by two sets $\underline{\mathcal{S}}$ and $\overline{\mathcal{S}}$ with $\underline{\mathcal{S}} \subseteq \overline{\mathcal{S}}$. The lower bound $\underline{\mathcal{S}}$ represents the values that must be in the set, and $\overline{\mathcal{S}}$ represents the values that may figure in the set; *i.e.* $\underline{\mathcal{S}} \subseteq \mathcal{S} \subseteq \overline{\mathcal{S}}$.

Redundant constraints (also called implied or surrogate constraints) are constraints which are implied by the constraints defining the problem. They do not change the set of solutions, and hence are logically redundant. Adding redundant constraints to the model may however further reduce the search space by allowing more pruning. Redundant constraints have been successfully applied to various problems such as car sequencing.

A global constraint can be seen as a constraint on a set of variables, modeling a well-defined part of the problem, and with a dedicated filtering algorithm (van Hoeve & Katriel 2006).

CP Models for the Bin Packing with Precedence Constraints

The bin packing with precedence constraint (BPPC) implies the following parameters and variables :

- n positive values $[s_1, ..., s_n]$ representing the size of each item.
- a number of available bins m.
- m variables $[L_1, ..., L_m]$ representing the load of each bin ($Dom(L_i) = \{0, \ldots, \sum_i s_i\}$)

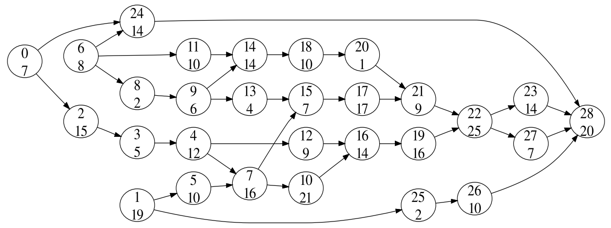

Figure 1: Precedence graph of the Buxey instance from Scholl benchmark data set (Scholl 93).

- n variables $[B_1, ..., B_n]$ representing for each item the bin where it is placed ($Dom(B_i) = \{1, \ldots, m\}$)
- a precedence (directed and acyclic) graph $G(\{1, .., n\}, E)$

The constraints should model the precedence, and the load of the bins. The objective is to minimize the maximum of the loads, *i.e.* $minimize \quad \max\{L_i\}$.

The precedence constraints are easily modeled as a set of constraints

$$B_i \leq B_j \ \ (\text{with } (i, j) \in E) \tag{1}$$

The simplest way to model the load of each bin is to add binary variables $X_{ij} \in \{0, 1\}$ telling whether item j is placed into bin i. These variables are linked to the variables of the problems through reified constraints:

$$X_{ij} = 1 \leftrightarrow B_j = i. \tag{2}$$

The loads of each bin is then expressed as a scalar product:

$$L_i = \sum_j X_{ij} \cdot s_j. \tag{3}$$

We improve this model with the following redundant constraint (Shaw 2004):

$$\sum_i L_i = \sum_j s_j. \tag{4}$$

The global constraint for bin packing, described in (Shaw 2004) and called `IloPack` in Ilog Solver (ILOG-S.A.), can also be added.

The speedup obtained with the redundant constraints (4) and `IloPack` is illustrated on a small instance of SALBP-2 from the Scholl benchmark data set (Scholl 93). The problem is to place the items (tasks) into 12 bins (workstations) while satisfying the precedence constraints, given in Figure 1, and minimizing the maximum height of the bins.

Table 1 shows (columns A,B and C) that adding the simple redundant constraint (4) reduces significantly the number of backtracks and time, and that the bin packing constraint from (Shaw 2004) reduces it even more.

New Redundant Constraints for the BPPC

We introduce new redundant constraints improving the filtering for BBPC by linking the precedences with the bin packing problem thanks to set variables.

We denote by $\mathcal{P}_i$ the set variable representing the predecessors of item i in the BPPC. An item j is predecessor of

A		B		C		D	
#bks	time	#bks	time	#bks	time	#bks	time
6894	10.62	4850	7.34	2694	4.3	445	1.07

Table 1: Comparison in terms of backtracks and time (seconds) of the bin packing models on the Buxey instance with 12 workstations. Column A is obtained with the basic model, column B by adding the redundant constraint (4), column C by also adding the `IloPack` constraint described in (Shaw 2004) and column D by adding also the redundant constraints (5) and (7).

an item i if and only if item j is placed in a bin preceding or equal to the bin of item i:

$$j \in \mathcal{P}_i \leftrightarrow B_j \leq B_i. \quad (5)$$

The lower bound of $\underline{\mathcal{P}}_i$ is initialized to i plus the set of items having an arc pointing to i in the transitive closure of the precedence graph. The transitive closure is computed with the $O(n^3)$ Floyd Warshall's algorithm (see (Cormen *et al.* 2001)). The preprocessing time to compute the transitive closure for the instantiation of the lower bounds $\underline{\mathcal{P}}_i$ is negligible for the instances considered in the experimental section (less than 150 tasks).

The upper bound $\overline{\mathcal{P}}_i$ is initially all the tasks, that is $\{1, ..., n\}$. A similar reasoning holds for the successor set variable $\mathcal{S}_i$ of item i.

The set of predecessors of item i is used to filter the lower bound of B_i since we know that items in $\mathcal{P}_i$ must be placed before item i. For a set U with elements taken from $\{1, ..., n\}$, we denote by $\text{sum}(U)$ the total size of items in U, that is $\text{sum}(U) = \sum_{j \in U} s_j$.

Theorem 1 *A redundant constraint for the* `BPPC` *is:*

$$\text{sum}(\mathcal{P}_i) = \sum_{k \leq B_i} L_k. \quad (6)$$

Proof 1 *The left and right members are simply two different ways of counting the cumulated size of the B_i first bins. The right member is the natural way and the left member counts it by summing the sizes of the items lying in a bin smaller or equal to B_i.* ■

Computation of $\text{sum}(\mathcal{P}_i)$**:** The computation of $\text{sum}(\mathcal{P}_i)$ can be easily modeled with binary variables representing the set of predecessors but a better formulation in terms of filtering and efficiency is possible using the power of the set variables. Indeed $S(\mathcal{P}_i) = \sum_{j \in \mathcal{P}_j} s_j$ is expressed as such in Ilog Solver using a global constraint called `IloEqSum` making a summation over a set variable. A function must be defined to make the mapping between the indices of items and the size of the items: $f : \{1, ..., n\} \mapsto \{s_1, ..., s_n\} : f(j) = s_j$. The global constraint takes three arguments: a set variable, a variable and a function:

$$\texttt{IloEqSum}(\mathcal{P}_i, \text{sum}(\mathcal{P}_i), f) \equiv \sum_{j \in \mathcal{P}_i} f(j) = \text{sum}(\mathcal{P}_i).$$

Computation of $\sum_{k \leq B_i} L_k$**:** The formulation of $\sum_{k \leq B_i} L_k$ could be achieved with m binary variables for each item i. A better formulation is possible introducing an array of m variables $\mathbf{CL} = [CL_1, ..., CL_m]$: $CL_i = \sum_{k=1}^{i} L_k$ for $i \in [1, ..., m]$ (CL for Cumulated Load). With this array, $\sum_{k \leq B_i} L_k$ is written with an element constraint (Van Hentenryck P. 1988) as CL_{B_i}.

The Ilog model of the redundant constraints (6) for the predecessors of item i is:

$$\texttt{IloEqSum}(\mathcal{P}_i, \text{sum}(\mathcal{P}_i), f) \wedge \text{sum}(\mathcal{P}_i) = CL_{B_i}. \quad (7)$$

Constraint (7) filters the domains of $\mathcal{P}_i$, B_i and the L_i's. We define similar constraints for the successor variables $\mathcal{S}_i$. The results obtained with the improved formulation are given in column D of Table 1. The redundant constraints really pays off for the time (1.07 against 4.3) and the number of backtracks (445 against 2694).

A Global Constraint for the BPPC

The definition of the global constraint BPPC is the following. $\texttt{BBPC}([B_1, ..., B_n], [L_1, ..., L_m], E)$ holds iff

(i) $B_i \leq B_j, \forall (i, j) \in E$ and

(ii) $L_i = \sum_{\{j \in [1..n] \mid B_j = i\}} s_j$, $\forall i \in [1..m]$.

Constraints (i) and (ii) are modeled with constraints (1-4), `IloPack` and the new redundant constraints (5) and (7).

For this global constraint, we also propose a new $O(n^2)$ algorithm to filter further the domains of $[B_1, ..., B_n]$ and $[L_1, ..., L_m]$. This filtering does not subsume the filtering obtained with the redundant constraints. Hence it must be added to the filtering obtained with the redundant constraints.

The redundant constraints (5) and (7) mainly prune the lower bound of the variable B_i. Considering the array of the upper bounds of the bin loads $[L_1^{\max}, ..., L_m^{\max}]$, the redundant constraints enforce that

$$B_i^{\min} \leftarrow \min\{j : \sum_{k=1}^{j} L_k^{\max} \geq \sum_{j \in \underline{\mathcal{P}}_i} s_j\} \quad (8)$$

The filtering rule (8) is one of the pruning achieved by (7).

Example 1 *An item has a size of 4 and has three predecessors of size 4,3,5. The maximum height of all the bins is 5. The item can certainly not be placed before the bin 4 because for the bin 4 we have $\sum_{k=1}^{4} L_k^{\max} = 20 \geq 16$ while for the bin 3 we have $\sum_{k=1}^{3} L_k^{\max} = 15 < 16$.*

Rule (8) is a relaxation of the largest lower bound that could be found for B_i:

- it assumes a preemption of the items over the bins, and
- it assumes that all the predecessors can potentially start from the first bin.

We propose an algorithm to compute a better lower bound by conserving the preemption relaxation but disallowing a predecessor j to start before its earliest possible bin $B_j^{\min}$.

Our algorithm requires the predecessors j to be sorted increasingly with respect to their earliest possible bin $B_j^{\min}$.

This is achieved in $\Theta(|\underline{\mathcal{P}}_i| + m)$ with a counting sort algorithm (Cormen *et al.* 2001) since the domains of the B_j's range over $[1, ..., m]$. This complexity can be simplified to $O(n)$ since $|\underline{\mathcal{P}}_i| < n$ and typically $m \sim O(n)$ (less bins than items).

Algorithm 1 computes the minimum possible bin for item i by considering that :

- each predecessor j cannot start before its earliest possible bin $B_j^{\min}$ but can end in every other larger bin and
- an item can be splitted among several bins (the preemption relaxation).

Algorithm 1 first places the predecessors of i that is elements of $\underline{\mathcal{P}}_i$. This is done in the **forall** external loop. Then the item i in placed in the earliest possible bin without preemption for it. We assume that there are $m + 1$ bins of capacity $[L_1^{\max}, ..., L_m^{\max}, \sum_{i=1}^n s_i]$. The additional fictive $(m + 1)^{th}$ bin has a capacity large enough such that every items can be put inside it. This guarantees the termination of the **while** loops. The complexity of the algorithm is $O(|\underline{\mathcal{P}}_i| + m)$ where m is the number of bins. For n items, the complexity becomes $O(n^2)$.

Algorithm 1 returns two values bin and $idle$. The value bin is used to prune the lower bound of B_i:

$$B_i^{\min} \leftarrow \max(B_i^{\min}, bin).$$

The value $idle$ is used to prune $L_{bin}^{\min}$. Indeed if B_i is assigned then $bin = B_i$. It means that $L_{bin}^{\min}$ must be at least larger that $L_{bin}^{\max} - idle$:

$$L_{bin}^{\min} \leftarrow \max(L_{bin}^{\min}, L_{bin}^{\max} - idle).$$

Of course, we use a similar filtering using the set variable $\mathcal{S}_i$ to filter the upper bound of B_i.

Algorithm 1: Considering bins of maximum loads $[L_1^{\max}, ..., L_m^{\max}, \sum_{i=1}^n s_i]$, bin is the smallest bin index such that every item from the set $\underline{\mathcal{P}}_i$ have been placed in a preemptive way in a bin smaller or equal to bin without starting before their earliest possible bin. The value $idle$ is the remaining space in this bin.

```
$bin \leftarrow 0$
$idle \leftarrow L_{bin}^{\max}$
forall $j \in \underline{\mathcal{P}}_i \setminus \{i\}$ do
    /* invariant: $bin$ is the smallest bin index such that
       items $\{1, ..., j-1\} \setminus \{i\}$ have all been placed in a
       preemptive way in a bin smaller or equal to $bin$
       without starting before their earliest possible bin and
       $idle$ is the remaining place in this bin.                */
    if $B_j^{\min} > bin$ then
        $bin \leftarrow B_j^{\min}$
        $idle \leftarrow L_{bin}^{\max}$
    $s \leftarrow s_j$
    while $s > 0$ do
        if $idle > s$ then
            $idle \leftarrow idle - s$
            $s \leftarrow 0$
        else
            $s \leftarrow s - idle$
            $bin \leftarrow bin + 1$
            $idle \leftarrow L_{bin}^{\max}$
/* place item $i$ without preemption                             */
if $B_i^{\min} > bin$ then
    $bin \leftarrow B_i^{\min}$
    $idle \leftarrow L_{bin}^{\max}$
while $idle < s_i$ do
    $bin \leftarrow bin + 1$
    $idle \leftarrow L_{bin}^{\max}$
$idle \leftarrow idle - s_i$
return $bin$, $idle$
```

Experimental results

We propose two different heuristics for the SALBP-2. The first one chooses the next variable to instantiate on bases of the domain sizes (first fail) while the second one is based on the topology of the precedence graph and try to build heuristically a good solution satisfying the precedences. For both heuristic, the decision variables to instantiate are $[B_1, ..., B_n]$ that is for each item, we decide the bin where it is placed:

- Heuristic 1: The next variable to instantiate is the one with the smallest domain (classical first fail heuristic). As tie breaking rule, the variable corresponding to the item with largest size is chosen first. As value heuristic, the chosen item is placed in the less loaded bin.
- Heuristic 2: The order of instantiation of the variables is static: variables are instantiated in an arbitrary topological order of the precedence graph (see for example the upper numbers inside the nodes on Figure 1). Then the chosen item is placed in the first possible bin less loaded than $\sum_{i=1}^n s_i/m$ (average load of the bins) or in the less loaded bin if there are no such possible bin.

We selected some instances of SALBP-2 from the benchmark of (Scholl 93) with a number of tasks ranging from 29 to 148. The name of the instances are given in Table 2 with the number of tasks indicated between parentheses. For each of the precedence graph (instance) we generate three problems with 6, 10 and 14 workstations (bins).

All experiments where conducted with Ilog solver 6.3 with a CPU Intel® Xeon(TM) 2.80GHz with a timeout of 300 seconds.

As first experiment, we propose the solve the problem with a Branch and Bound DFS using Heuristic 1 for three different models:

- C: state of the art CP model (1-4) and `IloPack`.
- D: C + redundant constraints (5) and (7).
- E: D + new filtering algorithm. The filtering algorithm of the global constraint is triggered whenever one of the bounds of a bin load variable L_i changes.

The results obtained (time and best objective) for the settings C, D and E with the first and the second heuristic are

Heuristic 1							Heuristic 2						
m	Time (s)			Objective			m	Time (s)			Objective		
	C	D	E	C	D	E		C	D	E	C	D	E
Buxey(29)							Buxey(29)						
6	1	**0**	**0**	55	55	55	6	3	**0**	**0**	55	55	55
10	**0**	1	**0**	34	34	34	10	300	**0**	**0**	34	34	34
14	7	**1**	2	25	25	25	14	0	0	0	25	25	25
Kilbrid(45)							Kilbrid(45)						
6	**1**	34	33	92	92	92	6	6	**0**	**0**	92	92	92
10	**6**	300	8	**56**	74	**56**	10	1	1	1	56	56	56
14	5	32	**2**	55	55	55	14	0	0	0	55	55	55
Hahn(53)							Hahn(53)						
6	**2**	**2**	**2**	2400	2400	2400	6	98	**1**	**1**	2400	2400	2400
10	3	**1**	**1**	1775	1775	1775	10	300	**1**	**1**	1827	**1775**	**1775**
14	25	**1**	**1**	1775	1775	1775	14	0	0	0	1775	1775	1775
Warnecke(58)							Warnecke(58)						
6	300	**1**	**1**	260	**258**	**258**	6	47	**1**	**1**	258	258	258
10	300	300	**71**	158	166	**155**	10	300	300	300	160	158	**156**$^{(155)}$
14	300	42	**19**	112	**111**	**111**	14	300	300	300	113	**112**	**112**$^{(111)}$
Tonge(70)							Tonge(70)						
6	**19**	68	68	585	585	585	6	126	**3**	**3**	585	585	585
10	300	300	300	355	**352**	**352**$^{(352)}$	10	300	**15**	**15**	369	**352**	**352**
14	300	300	300	326	**267**	**267**$^{(251)}$	14	300	161	**144**	260	**251**	**251**
Wee-mag(75)							Wee-mag(75)						
6	300	300	300	254	254	**252**$^{(250)}$	6	4	**2**	**2**	250	250	250
10	300	300	300	235	**201**	**201**$^{(150)}$	10	300	300	300	152	**151**	**151**$^{(150)}$
14	300	300	300	235	**201**	**201**$^{(108)}$	14	300	300	300	111	111	111$^{(108)}$
Lutz2(89)							Lutz2(89)						
6	300	**6**	**6**	142	**81**	**81**	6	300	**1**	**1**	83	**81**	**81**
10	300	300	**93**	142	62	**49**	10	300	**2**	**2**	51	**49**	**49**
14	300	300	300	348	**41**	**41**$^{(35)}$	14	300	**2**	**2**	36	**35**	**35**
Mukherje(94)							Mukherje(94)						
6	300	12	**10**	972	**704**	**704**	6	300	**5**	**5**	706	**704**	**704**
10	300	**43**	47	972	**424**	**424**	10	300	**5**	**5**	424	424	424
14	300	111	**93**	972	**311**	**311**	14	300	300	300	318	**313**	**313**$^{(311)}$
Barthold(148)							Barthold(148)						
6	**4**	5	5	939	939	**939**	6	300	300	300	940	940	940$^{(939)}$
10	300	**28**	**28**	574	**564**	**564**	10	300	300	300	566	566	566$^{(564)}$
14	300	300	300	569	**407**	**407**$^{(403)}$	14	300	**31**	34	404	**403**	**403**

Table 2: Results with Heuristics 1 and 2 for settings C, D and E.

given in Table 2. The optimum value is given in exponent between parentheses whenever the optimality could not be proved.

Analysis of results with heuristic 1: The positive effect of the redundant constraints and the global constraint is quite clear. Indeed, settings C, D and E allow to solve and prove the optimality of respectively 11, 17 and 20 instances on a total of 27 within a time limit of 5 minutes (300 seconds). See for example for example Lutz2 instance with 10 workstations that could be solved only with the global constraint.

Analysis of results with heuristic 2: Settings C, D and E allow to solve and prove the optimality of respectively 10, 20 and 20 instances on a total of 27 within a time limit of 5 minutes (300 seconds). The positive effect of the redundant constraints is still impressive but the global constraint does not allow to solve additional instances. For the instance Warnecke with 10 workstations, the objective is better with the global constraint.

Heuristic 2 allows to solve instances intractable with heuristic 1 (e.g. Barthold 14). The contrary is also true since the Warnecke instance with 10 and 14 workstations cannot be solved with heuristic 2 and can be easily solved with heuristic 1. For the most difficult instances unsolved with both heuristics (Wee-mag 10 and Wee-mag 14), heuristic 2 obtains a better objective value ($151<201$ and $111<201$) very close to the optimal values 150 and 108.

Comparison with state of the art dedicated algorithm: The state of the art algorithm for this problem is Salome 2 (Klein & Scholl 1996; Scholl & Becker 2006). A binary file of the implementation of the algorithm is available on *www.assembly-line-balancing.de*. Salome 2 finds the optimal solutions of almost all the instances within less than one second. As with our solution, it is not able to find and prove the optimum for the instances Wee-mag 10 and Wee-mag 14. Salome 2 uses a lot of dominance and reduction rules specific to this problem and objective function.

Generalized Assembly Line Balancing Problem: SALBP-2 is an academic problem. In real life assembly line problems, additional requirements are possible and the dominance rules used in Salome 2 are not valid anymore. Some possible additional constraints are (Becker & Scholl 2006):

- some tasks must be assigned in the same station,
- some tasks can not be assigned in the same station,
- there is a restriction on the cumulated value of particular task attributes,
- some tasks need to be assigned to particular stations,
- some tasks can not be assigned to particular stations,
- some tasks need a special station,
- some tasks need a minimum distance to other tasks,
- some tasks need a maximum distance to other tasks.

All these additional constraints can be added very easily in our CP model without changing anything else while dedicated algorithms such as Salome 2 cannot. To show the flexibility of our approach we have added some constraints to the Barthold instance with 10 workstations to form a Generalized Assembly Line Balancing Problem (GALBP): $|B_{138} - B_{16}| \leq 2$, $|B_{104} - B_{41}| \geq 2$, $|B_{12} - B_{35}| \leq 2$, $|B_{65} - B_{76}| \leq 2$, $|B_{101} - B_{102}| \geq 2$, $|B_{83} - B_{113}| \geq 2$, $|B_{19} - B_{28}| \geq 3$ and $B_{16} = 4$.

The time needed to reach and prove optimality (662) for settings C, D and E with Heuristic 1 are respectively 458, 171 and 143 seconds. Heuristic 2 gives bad results on the GALBP because we designed it specifically for problems with precedence constraints only. Here again, the new redundant and global constraints really help to solve the problem faster. This problem is unsolvable with state of the art dedicated algorithms such as Salome 2.

Another advantage of the constraint programming approach is that the objective function can be easily changed. For example, it is often desirable to smooth the workload among a given number of stations (Becker & Scholl 2006; Scholl & Becker 2006; Rekiek *et al.* 1999). This problem is called Vertical Line Balancing. This can be efficiently achieved in CP with the global constraints `spread` and `deviation` for the variance (Pesant & Regin 2005; Schaus *et al.* 2006) and the mean absolute deviation (Schaus *et al.* 2007).

Conclusion and Perspectives

We proposed a CP model for BPPC and SALBP-2, allowing a flexible expression of new constraints, such as in GALBP. We also designed a global BPPC constraint based on set variables representing the predecessors of each items, which exploits the transitive closure of the precedence graph. We have conducted an experimental validation on standard SALBP-2 benchmarks, showing the feasibility, the efficiency, and the flexibility of this approach. As future work, we plan to make an hybridization of CP and Local Search through Large Neighborhood Search. We also plan to use `spread` and `deviation` to solve the Vertical Line Balancing Problem. Currently there exists no exact method for this problem.

Acknowledgements

This research is supported by the Walloon Region, project Transmaze (516207) and partially by Interuniversity Attraction Poles Programme (Belgian State, Belgian Science Policy).

References

Becker, C., and Scholl, A. 2006. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research* 168(3):694–715.

Boysen, N.; Fliedner, M.; and Scholl, A. 2007. A classification of assembly line balancing problems. *European Journal of Operational Research* 674–693.

Castro, C., and Manzano, S. 2001. Variable and value ordering when solving balanced academic curriculum problem. *Proc. of the ERCIM WG on constraints*.

Cormen, T. H.; Stein, C.; Rivest, R. L.; and Leiserson, C. E. 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education.

Gervet, C. 1993. New structures of symbolic constraint objects: sets and graphs.

Hnich, B.; Kiziltan, Z.; and Walsh, T. 2002. Modelling a balanced academic curriculum problem. *Proceedings of CP-AI-OR-2002*.

ILOG-S.A. Ilog solver 6.3. user manual.

Klein, R., and Scholl, A. 1996. Maximizing the production rate in simple assembly line balancing: A branch and bound procedure. *European Journal of Operational Research* 91(2):367–385.

Pesant, G., and Regin, J.-C. 2005. Spread: A balancing constraint based on statistics. *Lecture Notes in Computer Science* 3709:460–474.

Rekiek, B.; De Lit, P.; Pellichero, F.; Falkenauer, E.; and Delchambre, A. 1999. Applying the equal piles problem to balance assembly lines. *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning* 399–404.

Schaus, P.; Deville, Y.; Dupont, P.; and Régin, J. 2006. Simplication and extension of the spread constraint. *Third International Workshop on Constraint Propagation And Implementation*.

Schaus, P.; Deville, Y.; Dupont, P.; and Régin, J. 2007. The deviation constraint. *LNCS CP-AI-OR* 4510:269–284.

Scholl, A., and Becker, C. 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research* 168(3):666–693.

Scholl, A. 93. Data of assembly line balancing problems. *Technische Universitt Darmstadt*.

Shaw, P. 2004. A constraint for bin packing. In Wallace, M., ed., *CP*, volume 3258 of *LNCS*, 648–662. Springer.

Van Hentenryck P., C. J.-P. 1988. Generality versus specificity: an experience with ai and or techniques. *In Proceedings of AAAI-88*.

van Hoeve, W.-J., and Katriel, I. 2006. Global constraints. In Rossi, F.; Beek, P. V.; and Walsh, T., eds., *Handbook of constraint programming*. Elsevier. chapter 6, 169–208.