Constraint Programming for Solving Various Assembly Line Balancing Problems

Yossi Bukchin[1] and Tal Raviv

Dept. of Industrial Engineering, Tel Aviv University,

Tel Aviv 69978, ISRAEL

July 2017

Abstract

In this paper, the constraint programming (CP) approach is applied for the simple assembly line balancing problem (SALBP) as well as some of its generalizations. CP is a rich modelling language that enables the formulation of general combinatorial problems and is coupled with a strong set of solution methods that are available through general purpose solvers. The proposed formulations are conversions of well-known mixed integer programming (MILP) formulations to CP, along with a new set of constraints that helps the CP solver to converge faster. As a generic solution method, we compare its performance with the best known generic MILP formulations and show that it consistently outperforms MILP for medium to large problem instances. A comparison with SALOME, a well-known custom-made algorithm for solving the SALBP-1, shows that both approaches are capable of efficiently solving problems with up to 100 tasks. When 1000-task problems are concerned, SALOME provides better performance; however, CP can provide relatively good close to optimal solutions for multiple combinations of problem parameters. Finally, the generality of the CP approach is demonstrated by some adaptations of the proposed formulation to other variants of the assembly line balancing problem including the U-shaped assembly line balancing problem and the task assignment and equipment selection problem.

Keywords: Constraint programming (CP), assembly line balancing, mixed-integer linear programming (MILP), branch & bound (B&B).

[1] Corresponding author: bukchin@tau.ac.il

1 Introduction and literature review

Assembly lines are often used in the last step of production, in which the final assembly of the product from previously manufactured parts is performed. An assembly line typically consists of several workstations in a sequential order, where each workstation is responsible for performing a specific set of tasks. The items move through the line from one workstation to the next according to their order and ending as finished products.

To maximize the efficiency of the line, the total assembly time has to be divided as equal as possible, among the workstations. This way, the workers' idle time, caused by the load differences among stations, is minimized. Since the assembly of each product consists of indivisible elements (tasks), the problem of allocating the tasks to the stations becomes a combinatorial problem, which is called the *assembly line balancing problem* (Scholl, 1999).

The basic and most general structure of this problem, the simple assembly line balancing problem (SALBP), was first introduced by Salveson (1955). In this problem, a set of tasks, characterized by deterministic processing times and precedence constraints among them, has to be assigned to stations. Although the basic objective is to maximize the efficiency of the line (or minimize the total workers' idle time), two common versions of the problem are known as the SALBP-1 and the SALBP-2 (Baybars 1986). The former minimizes the number of stations for a given (maximal) cycle time, and the latter minimizes the cycle time (or maximizes the throughput rate) for a given number of stations. Although many extensions of the SALBP have been investigated in the last few decades (see, for example, the reviews of Rekiek et al. 2002 and Becker and Scholl 2006), there are still attempts to develop efficient algorithms for the SALBP, which were shown to be NP-Hard (Baybars 1986) as a reduction of the partition problem (Karp, 1972).

Due to the nature of these problems, both heuristic and exact methods have been developed over the years (Scholl and Becker 2006). In the exact methods, we can distinguish between generic methods, such as mixed-integer linear programming (MILP) and custom-made methods. The first mathematical formulation for the SALBP was introduces by Bowman (1960) and was later on improved by White (1961). An improved MILP formulation, which also presented additional upper- and lower-bound constraints on the number of stations, was presented in Patterson and Albracht (1975) and later on in Talbot and Patterson (1984). A comparison between several formulations was performed by Pastor et al. (2004). Amen (2006) was the first

to claim that the effectiveness of the designed formulation depends on the solution technique (e.g., branch-and-bound techniques with LP-relaxation or general implicit enumeration techniques). A recent improvement of the MILP formulation was presented in Pastor and Ferrer (2009), in which additional constraints on the station index were added for both SALBP-1 and SALBP-2. Note that although their formulation was found to be superior compared to previous MILP formulations, it was still outperformed by custom-made algorithms for large-scale instances of the problem (Otto et al. 2013).

The main exact custom-made algorithms for the SALBP (Baybars 1986, Scholl and Becker 2006) can be divided to dynamic programming (Held et al. 1963, Jackson 1956, Schrage and Baker 1978) and branch and bound (B&B), while the latter significantly outperforms the former (Scholl and Klein 1997). Among others, one can mention the FABLE algorithm (Johnson 1988), Eureka (Hoffmann 1992), SALOME (Scholl and Klein 1997), and BB&R (Sewell and Jacobson 2012, Morrison et al. 2014). The reason for the superiority of custom-made approaches over generic formulations may be explained by the fact that generic formulations cannot exploit the characteristics of a specific problem as a custom-made procedure does. However, the generic approach has an inherent advantage because it can be used as a building block for other variations of the line balancing problem by a minor modification of the objective functions and/or the constraints. Hence, we believe that there is need for further research in both directions.

One of the generic approaches for solving combinatorial optimization problems is constraint programming (CP). Several authors introduced solutions methods based on the CP paradigm to solve some variants of the assembly line balancing problems. Bockmayr and Pisaruk (2001) presented a hybrid algorithm, which combines CP and MILP, for solving a generalization of SALP-1. Pastor et al. (2007) compared several integer-programming formulations with CP for SALBP-1 and SALBP-2 using a commercial solver. They concluded that one of the integer programming models delivered the best solutions for each of the two problems. Schaus and Deville (2008) and Schaus (2009) compared several CP formulations and branching heuristics for the SALBP-2. Topaloglu et al. (2012) presented a rule-based model for an extension of the SALBP with alternative precedence constraints. The model was formulated as an integer program and as a CP. These models were solved using commercial solvers and the performances of CP were found superior.

In this paper, we suggest new CP based formulations for several variations of assembly line balancing problems. More specifically, we present formulations for SALBP-1, SALBP-2, the U-shape assembly line balancing problem- type 1 and the equipment selection and task assignment problem (Bukchin and Tzur 2000). We use the SALBP-1 to demonstrate the effectiveness of the proposed method through an extensive experiment based on the diverse dataset introduced by Otto et al. (2011, 2013). The performance evaluation is carried out by comparison with the MILP formulation of Pastor and Ferrer (2009) and SALOME (Scholl and Klein 1997 SALOME). The reason is twofold: since the proposed approach is generic in nature, it is only natural to compare it to recent generic solution approaches. However, at the same time, it is important to compare CP with SALOME, which is considered as one of the most efficient custom-made solution procedures known to date for the SALBP. The effectiveness of the CP approach for the other variations was demonstrated by comparison with a state of the art MILP formulations solved by a commercial solver.

The main contribution of this study is first in demonstrating that, by exploiting certain special properties of the problem, CP is a general solution method that can be competitive with a state-of-the-art customized algorithm for the simple assembly line balancing problems. Next, we show that CP also performs well in solving various generalized assembly line balancing problems, when compared with mixed integer approach.

The rest of the paper is organized as follows. In the next section, an overview on constraint programming is provided, along with the formulation for SALBP-1. In Section 3, an extensive experiment is presented with a comparison between the proposed CP approach (solved with IBM ILOG CP Solver), SALOME and the MILP formulation of Pastor and Ferrer (2009) (solved with IBM CPLEX). In Section 4, we develop several formulations of CP for common generalized assembly line balancing problems and report on an extensive numerical experiment that demonstrates their merits. Concluding remarks are provided in Section 5.

2 Constraint programming (CP) formulation

2.1 An overview

Constraint programming (CP) is a solution approach that is based on formulating combinatorial problems as constraint satisfaction models and solves them by using domain-specific or general

methods (see, for example, Apt 2003). In this study, we focus on the modeling aspect and use a state-of-the-art general commercial solver to demonstrate the effectiveness of our formulation. Using a general solver (either commercial or open source) is a practical and attractive option because it enables very rapid implementation of solution methods for a diverse set of optimization problems. It also makes it easy to apply minor modification and side constraints that can appear in practical settings. Using a rich modeling language, one can enlist a lot of the coding effort already conducted by the developers of the general solver to numerous particular optimization problems. CP has been successfully used to solve a variety of problems in the domains of vehicle routing, scheduling, timetabling and others. See, for example, Shaw (1998), Baptiste (2012) and Bukchin and Zaides (2016).

A CP solver solves a model by repeatedly applying and propagating its constraints to prune the domains of the decision variables and the objective function value. When no further pruning is possible, the solver divides the problem into sub-problems (branches), each with one additional constraint. This process iterated on the sub-problems until the domain of each of the decision variables is reduced to a singleton. Since the solver uses the constraints to prune the domains of the decision variable, an effective CP formulation consists of constraints that facilitate a strong reduction of these domains. That is, by pruning the domain of one variable, the constraints imply a substantial reduction of the domains of other variables. The art of CP modeling not only encompasses creating a set of constraints that defines the feasible solution set but also includes redundant constraints that express strong logical or mathematical relations between the values of the decision variable.

A CP model resembles an integer programming model in terms of syntax. It contains a deceleration of decision variables with their domains, a set of constraints, and possibly, an objective function. However, the CP modeling paradigm is much more expressive. In fact, the language is a superset of the integer linear programming modeling language. In addition to equality and inequality constraints between linear mathematical expressions, a CP model can contain non-linear expressions, logical expressions, use decision variables as indices to other vectors of decision variables, and include global constraints that capture a relationship between large sets of decision variables. Here are a few examples of valid expressions in a CP model:

Reification – any valid logical expression including equality or inequality may be used as an argument of an indicator function that returns a value of one if the expression is true and zero

otherwise. For example, consider a vector of integer decision variables x_i, $i = 1..n$. If we wish to express the number of elements in the vector with a particular value, e.g., 17, we can do this in a CP model, using the following expression:

$$\sum_{i=1}^{n} (x_i = 17).$$

If we need that at least two $x's$ to be equal to 17, we could add the following constraint to the model.

$$\sum_{i=1}^{n} (x_i = 17) \geq 2$$

Recall that equivalent constraints in MILP models require the introduction of additional decision variables. For example, we could define a binary variable y_{ij} that equals one if x_i is equal to j and zero otherwise, and introduce the following constraints:

$$\sum_{j} j y_{ij} = x_i \quad \forall i$$

$$\sum_{j} y_{ij} = 1 \quad \forall i$$

$$\sum_{i=1}^{n} y_{i,17} \geq 2.$$

All Different – this is a global constraint that allows requiring that each of the elements in a set of decision variables (e.g., a vector) obtains a different value. For example, if n workers are needed to be assigned n different jobs, a variable x_i that denotes the job assigned to worker i can be defined by

$$\text{AllDifferent}(x_1, \dots, x_n).$$

The solver is in charge of assigning valid values that satisfy this constraint (and possibly others in the model). In many cases, global constraints such as AllDifferent help prune the domains of the variables efficiently. Expressing this requirement in a MILP model would require the definition of a two-dimensional array of decisions variables and $2n$ equality constraints as in the classical formulation of the assignment problem.

Domains of decision variables – A CP model contains a declaration of the decision variables and their domains. Many solvers support only finite discrete domains and, in particular, finite

sets of integers. An important special case is the Boolean variables with the domain {0,1}. The solver uses the initial domain as a starting point and then proceeds to prune them by applying the constraint and by branching on decision variables and dividing their domains.

2.2 CP formulation for SALBP-1

In this section, we present our CP formulation of the SALBP-1. To this end, we define the following notation:

Parameters

- n Number of tasks
- ct Cycle time
- t_i Processing time of task i. We assume, without loss of generality, that t_i is an integer
- P_i Set of immediate predecessors of task i
- S_i Set of immediate successors of task i
- ub An upper bound for the number of stations in the optimal solution

The decision variables in our model are straightforward:

- x_i The number of the station to which task i is assigned
- m The total number of stations in the assembly line

Using this notation, we can define the following CP model:

$$minimize\ m \tag{1}$$

$$\sum_{i=1}^{n} (x_i = j)\, t_i \leq ct \quad \forall j = 1, \dots, ub \tag{2}$$

$$x_i \leq m \quad \forall i: S_i = \emptyset \tag{3}$$

$$x_i \leq x_j \quad \forall i, j = 1, \dots, n: i \in P_j \tag{4}$$

$$x_i \in \{1, \dots, ub\} \quad \forall i \in 1, \dots, n \tag{5}$$

$$m \in \{1, \dots, ub\} \tag{6}$$

The objective function (1) directly minimizes the number of stations in the line. Constraint (2) stipulates that the total time of tasks assigned to each station does not exceed the allowed cycle time. Here, we use the reification expression $(x_i = j)$ as an indicator function that equals one if the equation holds and zero otherwise. In Constraint 24, the value of the decision variable m is related to values of the $x's$ variables. In the optimal solution the solver will select $m = \max_i x_i$. Constraint (4) declares the precedence relationships between the tasks and, in (5) and (6), the initial domains of the decision variables are given. The upper bound on the number of stations was calculated using the randomized single path method (Arcus, 1965).

The CP formulation of the SALBP-1 is simpler and more direct that any MILP formulation because we are not limited to linear constraints here. While this formulation can be used as an input for a general purpose CP solver to tackle small instances of the problem in a reasonable amount of time, a tighter formulation can be used to solve a larger instance. To this end, we will need to define the following additional notation.

$\tilde{P}_i$ The set of *all* predecessors of task i (either direct or indirect).

$\tilde{S}_i$ The set of all successors of task i (either direct or indirect).

Clearly, $\tilde{P}$ and $\tilde{S}$ can be readily constructed from the precedence constraints given above. Based on this set, we further calculate the following auxiliary parameters

$$E_i = \left\lceil \frac{t_i + \sum_{k \in \tilde{P}_i} t_k}{ct} \right\rceil \tag{7}$$

$$L_i = \left\lfloor \frac{t_i - 1 + \sum_{k \in \tilde{S}_i} t_k}{ct} \right\rfloor \tag{8}$$

where $\lceil x \rceil$ ($\lfloor x \rfloor$) is the smallest (largest) integer that is still larger (smaller) or equal to x. E_i is a lower bound on the index of the station to which task i can be assigned and, similarly, L_i is a lower bound on the number of stations between the station of task i and the last station in the line. Clearly, these two parameters can be calculated in a pre-processing procedure. The semantic of the L_i parameter is slightly different from its common semantic in the literature, e.g., Pastor and Ferrer (2009), where L_i denotes the last possible station of task i. Note that, in (8), one is subtracted from the nominator and the ratio is rounded down. One is subtracted because if the

sum $t_i + \sum_{k\in\tilde{S}_i} t_k$ is an integer multiplication of ct, say $k = \left(t_i + \sum_{k\in\tilde{S}_i} t_k\right)/ct$ then the number of stations after the station of task i should be $k-1$ and not k.

Next, we can add the following constraints to the CP formulation:

$$E_i \le x_i \le m - L_i \quad \forall i = 1, \dots, n \tag{9}$$

This idea of limiting the earliest and latest station of each task is not a novel one; it is used by many previous MILP formulations. However, in the context of constraint programming, it is powerful because it not only reduces the initial domain of the $x's$ variables but also helps to propagate any change in the domain of m on all the domains of the $x's$.

An additional way to tighten our CP formulation is based on the notion of minimal distance (in terms of stations) between each pair of tasks for which a direct or indirect precedence relationship is applied. We define the minimal distance between a pair of tasks as follows:

$$D_{ij} = \left\lfloor \frac{t_i + t_j - 1 + \sum_{k\in\tilde{S}_i\cap\tilde{P}_j} t_k}{ct} \right\rfloor \qquad \forall i,j = 1,\dots,n: i \in \tilde{P}_j. \tag{10}$$

Note that the set $\tilde{S}_i \cap \tilde{P}_j$ consists of all tasks that should be assigned to some stations between the station of task i and the station of task j (inclusive). The sum in the numerator of (10) is the total time needed for all these tasks to be completed including tasks i and j minus 1; thus, by dividing this by the cycle time, ct and rounding down, we can obtain a lower bound on the difference $x_j - x_i$. Note that it is important to subtract 1 from the numerator; otherwise, the lower bound is not tight if the sum is an integer multiplication of *ct*. Based on this observation, we replaced the task precedence constraint, (4), with the following stronger constraint

$$x_i + D_{ij} \le x_j \qquad \forall i,j = 1,\dots,n: i \in \tilde{P}_j. \tag{4'}$$

Interestingly the "distance", D_{ij}, may violate the triangle inequality due to rounding considerations; namely, $D_{ij} > D_{ik} + D_{kj}$ is possible. For example, assume that the cycle time is $ct = 10$, there are three tasks 1,2,3 with $t_i = 4$ for all i, and the precedence relationships are $P_1 = \emptyset, P_2 = \{1\}, P_3 = \{2\}$ $\left(\tilde{P}_3 = \{1,2\}\right)$. In this case, $D_{12} = D_{23} = 0$, but $D_{13} = 1$. The best practice is to include all the instances of (4') that are associated with direct precedence relationship or with indirect ones where the triangle inequality is violated. That is,

$$x_i + D_{ij} \le x_j \qquad \forall i,j = 1,\dots,n: i \in \tilde{P}_j \wedge \left(\nexists k \in \tilde{S}_i \cap \tilde{P}_j: D_{ij} \le D_{ik} + D_{kj}\right). \tag{4"}$$

By using (4”), we include only the instances of the extended precedence constraints that have the potential to reduce the domain of some decision variables. Note that other instances of (4’) are implied by the direct precedence constraints and thus redundant and only increase the size of the model.

Finally, we tighten the above CP formulation by selecting a relatively small upper bound on the optimal number of stations (ub). While setting $ub = n$ results in a perfectly valid formulation, a substantial reduction in the run time can be obtained by allocating some computational effort to establishing a better upper bound. Inspired by Otto et al. (2013), we use a simple greedy random heuristic that identifies a relatively good allocation of the tasks for the station (see Arcus 1965). We applied this randomized procedure 100 times and took the value of the best solution as an upper bound on the optimal solution. We noted that in our benchmark instances, this heuristic typically provides a solution within 10% of the optimum, which was much smaller than using the number of tasks n as an upper bound. Clearly, a more sophisticated heuristic could be applied to obtain better upper bounds, but this is not the focus of this study.

3 Experiments

3.1 Experimental design

The purpose of the experiments is to compare the performance of the CP formulation of SALBP-1 against the Pastor and Ferrer (2009) MILP formulation (denoted from now on simply as MILP) and the Scholl and Klein (1997) SALOME custom-made branch and bound algorithm. As noted above, although SALOME has shown better performance than MILP, it is interesting to compare CP with MILP because both are generic solution approaches, which can be relatively easily adapted to other variants of the ALBP.

The selected dataset is drawn from Otto et al. (2011, 2013). The dataset includes 2100 instances that are divided according to multiple parameters. The use of this relatively new dataset is justified by Otto et al. (2013); they claim that the currently existing datasets from the literature are not randomized nor are systematically diversified and therefore cannot be used as a representative sample. Moreover, these datasets do not contain realistic instances and are not capable of performing a thorough comparison between the solution procedures (Otto et al. 2013).

In this study, we analyze the results first by the problem size parameter and then by other parameters related to the structure of the precedence diagram and the assembly time distribution. The dataset is divided into four sets of 525 instances, each for 20, 50, 100 and 1000 tasks. The first network structure parameter relates to the existence of chains and bottlenecks in the precedence diagram. A chain is defined by a series of tasks arranged in a path, in which the first has one successor, the last has one predecessor, and the rest have one predecessor and one successor each. A bottleneck task is defined as one that is the only successor of at least two other tasks and the only predecessor of at least two other tasks. The corresponding parameter has three values: (1) CH, which refers to networks that contains at least 40% chain tasks; (2) BN, which refers to networks that contain bottleneck tasks of at least a degree of eight for large scale problems and at least a degree of four for small scale problems; and (3) MIXED, which refers to all other networks for which the existence of chains/bottlenecks is not controlled. The second network structure parameter is the order strength (Thesen, 1977). The order strength (OS) expresses the ratio between the actual precedence relationships (direct and indirect) and all possible precedence relationships. The OS is equal to one if all tasks are arranged sequentially in one path and zero if no precedence exists between the tasks. The values considered in this study are 0.2, 0.6 and 0.9. The last parameter relates to the task time distribution, where we distinguish between bimodal distribution (BN) and two cases of unimodal distribution: pick at the bottom (PB) and pick in the middle (PM).

The results from the MILP and CP models were obtained on an Intel i7-4700 with 16GB RAM using Windows 7 (64-bit version). A time limit of 900 seconds was set to each of the 1000-task instances and 300 seconds for each of the other instances. The MILP and the CP models were solved using the IBM CPLEX Studio 12.6.2 using its MILP and CP solvers. The code that was used to run the experiment and the detailed solutions are available from the authors upon request.

We replicated the experiment of Otto et al. (2013) with the SALOME executable file, available from www.assembly-line-balancing.de, on the same hardware used for the CP and MILP. However, since this code was designed for smaller memory model a time limit of 90 seconds was set, in order to avoid out of memory errors.

3.2 General results

We will first report the general results on the different problem sizes. Here, we compare the CP solutions against MILP and SALOME for the 2100 instances. The results are shown in Table 1 and Table 2. As can be expected, the problem size has a major effect on the performance of all solution approaches. However, we can observe that this factor also significantly affects the relative performance of the studied methods.

Table 1 presents the performance of each approach for each problem size, where the number of optimal solutions, as well as the average and maximal relative optimality gap, in percentage, and number of stations, is given. The comparison between CP and the other methods with respect to the solution values is given in Table 2, where each value denotes the number of instances in which CP was better, worse or equal to the other methods. Note that preliminary experiments have shown that although CP is often fast in finding near optimal or optimal solutions, it was rarely able to prove the optimality of these solutions. Hence, when it was possible, we have used the lower bound and optimal solutions obtained from the other solution methods to identify CP optimal solutions.

One can observe that in 20-task problems, all 525 instances were solved to optimality by all compared methods. Hence, this set of problems can be ignored from now on.

When the number of tasks increases to 50, one can see that CP and MILP provide nearly the same number of optimal solutions (474 and 472 out of the 525 instances, respectively), which is slightly higher than those obtained by SALOME (451). When looking at the optimality gap, we can observe similar phenomena because CP and MILP provide a lower optimality gap than SALOME, both in percentages and number of stations. Note that the gap was calculated with respect to the highest known lower bound that was obtained by either SALOME or the MILP model. When comparing the solutions values, we can see that CP provides better solutions than SALOME in 41 instances, while the opposite occurs only for five instances. The comparison with MILP also shows similar performance because CP is better in eight instances, while MILP outperforms in five instances. Due to the size of the problem, the differences in the number of stations in which one method is better than other are equal to one in most cases, with a maximal value of two.

When looking at the run times of CP, we can see in Figure 1a that the distribution is bimodal; most of the cases are solved quickly, while others did not reach an optimal solution within 300

seconds. More specifically, 422 instances were solved to optimality in less than two seconds, 65 used the entire 300 seconds and the remaining 38 instances were spread across the range.

When increasing the number of tasks up to 100, we can see that the performance of MILP deteriorates compared to SALOME and CP. First, we can observe in Table 1 that SALOME and CP provide 355 and 351 optimal solutions, respectively; however, only 321 optimal solutions are obtained by using MILP. However, when looking at the optimality gap, we can see that CP performs much better than the other two solution approaches, providing an average gap of 1.49% (max 12.7%) versus 2.22% (max 16.7%) in SALOME and 2.76% (max 21.8%) in MILP. The average and maximal optimality gap in terms of the number of stations provides similar results, with 0.69 (max 7), 1.17 (max 8) and 1.34 (max 12) for CP, SALOME and MILP, respectively. The superiority of CP over SALOME and MILP is even clearer when comparing the solution value. We can observe that in the case of 100 tasks, CP provides significantly better results than SALOME and MILP because it is better than SALOME in 142 instances (worse in only 30) and better than MILP in 155 instances (worse in only 4 cases). Note that in the cases when CP is better than SALOME or MILP, the difference is up to 5 and 12 stations, respectively. The latter indicates that in some cases MILP may provide solutions that are far from optimal.

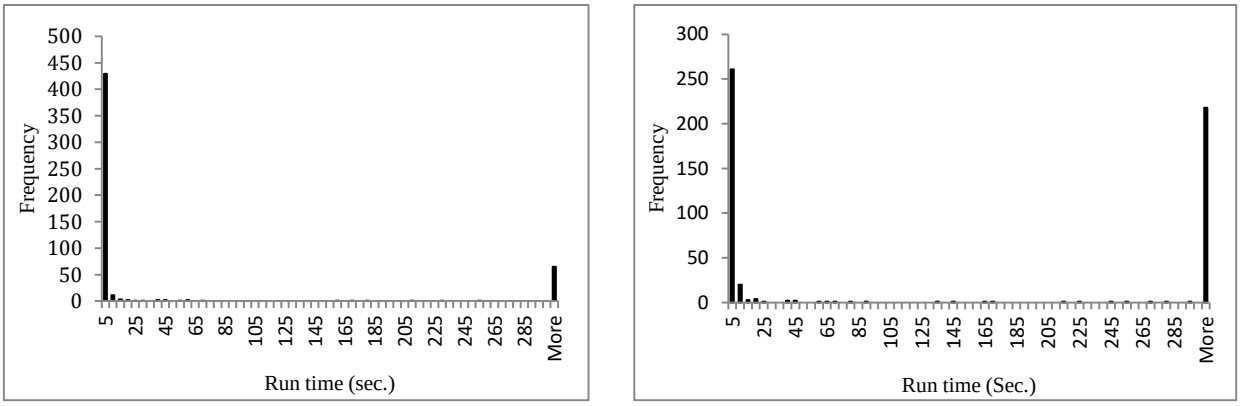
Table 1. Performance of SALOME, MILP and CP in all problem instances

No. of tasks	No. Opt. Sol.			Relative Opt. Gap (%)						Absolute Opt. Gap (Stations)					
	SALOME	MILP	CP	SALOME		MILP		CP		SALOME		MILP		CP	
				Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max
20	525	525	525	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0
50	451	474	472	0.78	14.8	0.54	12.0	0.54	12.0	0.21	4	1.139	3	0.133	3
100	355	321	351	2.22	16.7	2.76	21.8	1.49	12.7	1.17	8	1.34	12	0.69	7
1000	186	NA	4	2.88	16.2	NA	NA	4.89	22.5	13.57	82	NA	NA	21.37	116

Table 2. CP versus other methods in all problem instances

CP vs. other	50 tasks		100 tasks		1000 tasks	
	SALOME	MILP	SALOME	MILP	SALOME	MILP
CP Better	41	8	142	155	8	NA
CP Worse	5	5	30	4	475	NA
Equal	479	512	353	366	42	NA

The run time behavior of CP is similar to the one shown for the 50-task instances; however, typically, the run times are relatively longer, with a larger set of instances that use the maximal limit (see Figure 1b). In particular, 125 instances were solved to optimality in less than one second, 253 in less than four seconds, 218 used the entire time limit, and the remaining 54 instances are distributed along the range.



(a) 50-task instances (b) 100-task instances

Figure 1. Run time distribution of CP

The results obtained from the dataset of the 1000-task instances are quite interesting: the comparison between CP and MILP is in line with the previous results, but the comparison with SALOME shows an opposite trend. First, we realized that MILP is not capable of solving such large instances because no feasible solutions were obtained within the 900 second time limits. CP, on the other hand, provides relatively good solutions (though not optimal in most cases) for most instances. The best performance was obtained by SALOME, both in the number of optimal solutions, the optimality gap and the solution value comparison. Specifically, 186 instances were solved to optimality by SALOME, while the CP was able to provide an optimal solution for only four instances. An average optimality gap of 2.88% (max 16.2%) was obtained by SALOME versus 4.89% (max 22.5%) by CP. Clearly, these values in terms of the number of stations were quite similar. The superiority of SALOME over CP is even clearer when looking at the solution value comparison because SALOME provided better solutions in 475 instances, with an opposite value of only eight instances. Note that although in most cases the difference in the number of stations is relatively small (1-4 stations for solutions of 100-200 stations), this value may increase by up to several dozen when the solution value is in the range of 500-600 stations.

It should be made clear that the comparison of CP versus MILP and SALOME is not symmetric. While the two latter approaches are capable of providing a valid lower bound even when an optimal solution cannot be found (or optimality cannot be proven), the CP approach does not provide such a bound unless optimality is proven. Thus, for the larger and hardest instances that cannot be solved to optimality by exact methods, CP should be used as a heuristic. Indeed, one can use the CP to obtain a high quality solution in conjunction with other methods that are able to provide a lower bound so that the potential optimality gap can be controlled.

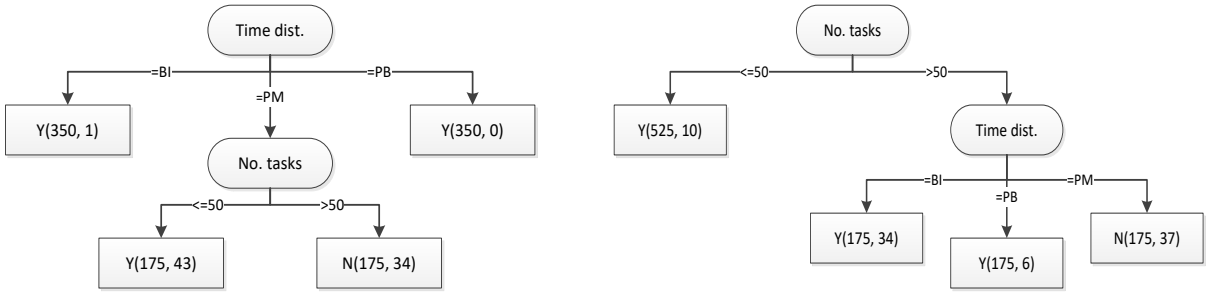
3.3 The effect of the problem parameters

As shown above, the performance of the studied methods is strongly affected by the problem size and possibly by other parameters. In this section, we examine the effect of the problem parameters on the relative performance of the solution approaches to design a recommended scheme regarding the best method(s) to use in each combination of parameters.

This problem is actually a classification problem in which the aim is to provide a prediction model to suggest the best method to use for each future instance. To this end, we have chosen to adopt a decision tree algorithm, the C4.5 (Quinlan 1993). This algorithm belongs to a group of supervised machine learning algorithms that constructs a decision tree based on maximal information gain, which is equivalent to minimum entropy (Shannon's 1948). The algorithm was executed on the WEKA machine learning software package (Hall et al. 2009).

As noted above, the 20-task problems were omitted from this study since all instances were solved to optimality by all approaches. Still, in the 50 and 100-task instances, most of the methods performed well, and most of the instances were solved to optimality. The resulting decision tree for the CP, when 50 and 100-task instances are concerned, indicated that this approach is recommended for *all* parameter combinations (namely, a tree with one node/leaf). The decision tree of SALOME and the MILP are shown in Figure 2a and b, respectively. In each leaf we can see the number of instances on the left and the number of classification mistakes on the right, where “Y” and “N” refer to the possible recommendation to use the algorithm in the associated class (set of parameters values). The absolute and relative number of mistakes can serve as an indicator of the strength of the recommendation. Both decision trees recommend using SALOME and MILP for all cases except for the case of 100 tasks and unimodal time distribution with “peak in the middle” (PM).

Last, we studied the 1000-task problems. As discussed above, in this set of problems, the SALOME outperforms CP in most of the instances, while the MILP cannot provide feasible solutions. Still, due to the advantage of using generic formulations (see above), we decided to examine the combination of parameters for which the CP can be used as an effective heuristic. To this end, we examined the optimality gap of the CP solution for each instance and defined a solution within a predefined gap as an acceptable solution. Two values of optimality gap were examined, 5% and 1%, and for each value, a decision tree was constructed. The decision tree for the former is presented in Figure 3. One can see that using CP as a heuristic is recommended for cases of unimodal time distribution with "pick at the bottom" (PB). For the bimodal time distribution, the decision depends on the order strength (OS), where the CP is recommended where the OS values are up to 0.6.



(a) SALOME (b) MILP

Figure 2. Decision tree for SALOME and MILP, 50-100 tasks

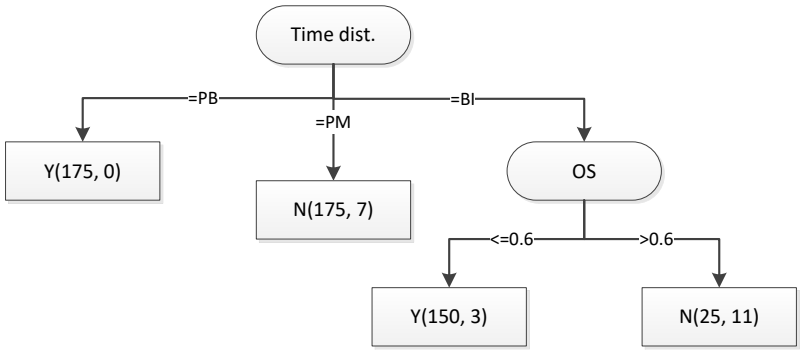


Figure 3. Decision tree for the 1000-task dataset and 5% optimality gap

When reducing the optimality gap to 1%, namely, enforcing a more rigorous condition for defining a heuristic solution as acceptable, a more complicated decision tree is obtained (see

Figure 4). Clearly, since the condition for defining efficiency is tightened, all inefficient classes that were obtained for an optimality gap of 5% should remain, and the classes that define CP as an efficient heuristic should be subsets of the above classes. Indeed, we can observe that only subsets of the PB distribution are now recommended, where the high OS instances (above 0.6), and medium OS instances with a chain graph structure were excluded. Additionally, only a subset of the BI time distribution with small to medium order strength remained recommended (small OS and BN graph structure).

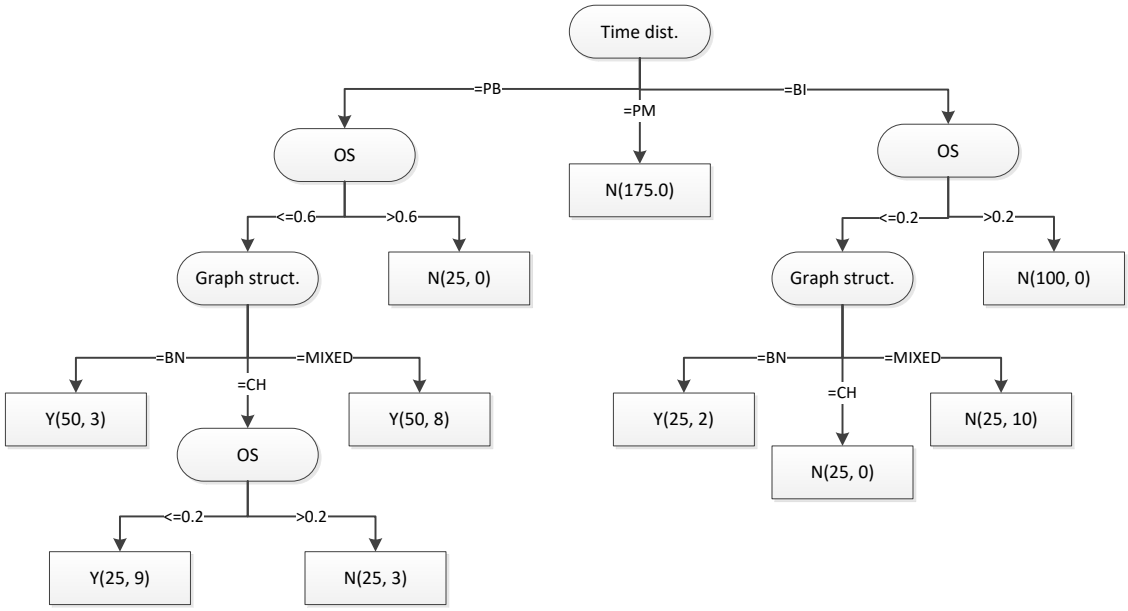


Figure 4. Decision tree for the 1000-task dataset and 1% optimality gap

4 Extensions

In the previous section we have shown that the CP solver can hardly compete with custom-made solution approach, such as SALOME, however, performs nicely when compared to MILP, another generic solution approach. The purpose of this section is twofold; first, to show that the CP paradigm allows for a relatively easy formulation of other variants of the line balancing problem. Moreover, we also demonstrate how the same successful modeling ideas presented for the SALBP-1 can be applied for these problems. Second, to solve large number of instances of each variant with CP and MILP and compare their performance. The following variants of the assembly line balancing problem were formulated:

1. The simple assembly line balancing type 2 (SALBP-2).

2. The U-shape assembly line balancing type 1 (UALBP-1). See Miltenburg and Wijingaard (1994), Urban (1998) and Scholl and Klein (1999).
3. Task assignment and equipment selection to minimize equipment cost (Bukchin and Tzur 2000).

The CP formulation of these problems is presented in the following sub-subsections.

4.1 The simple assembly line balancing - type 2 (SALBP-2)

In SALBP-2, the setting is identical to those of SALBP-1, but instead of having the cycle time as part of the problem input and aiming to minimize the total number of stations, the goal is to minimize the cycle time for a given number of stations. To formulate it, we use the same notation used for model (1)-(6) and the extensions in (9) and (4"), but we set m (the number of stations) as an input parameter and ct (the cycle time) as a decision variable. In addition, we calculate the values of E_i, L_i and D_{ij} with some upper bounds on the unknown cycle time.

We define lower bound (lb) and upper bound (ub) parameters to provide the model with the initial domain of the variable, ct. The lower bound is simply the sum of the processing times divided by m (Baybars, 1986). The upper bound was calculated using the same randomized simple path procedure (Arcus, 1965), applied repeatedly for $ct = lb, lb + 1, \ldots$ until a feasible solution with m stations was obtained. The parameters, E_i and L_i, are calculated based on ub. With this notation, the following CP model may be used to solve the SALBP-2.

$$minimize\ ct \tag{11}$$

$$\sum_{i=1}^{n} (x_i = j)\, t_i \leq ct \quad \forall j = 1, \ldots, m \tag{12}$$

$$x_i \leq x_j \qquad \forall i, j = 1, \ldots, n : i \in \tilde{P}_j \tag{13}$$

$$x_i \in \{E_i, \ldots., m - L_i\} \quad \forall i \in 1, \ldots, n \tag{14}$$

$$ct \in \{lb, \ldots, ub\} \tag{15}$$

The objective function, (11), minimizes the cycle time. Constraint (12) limits the cycle time as specified. Constraint (13) states the precedence constraints in the same strong manner as in (4"). The domains of the decision variables are specified in (14) and (15).

The CP model (11)-(15) is a valid description of the SALBP-2, but it can be further strengthened by updating the values of E_i and L_i dynamically, while the model runs, instead of using their static value obtained by the preprocessing. In fact, equations (7), (8) and (10) can be added to the model. Many general solvers can handle the dynamic definition of the variable initial domains and the quantifiers to extract information that can be used to prune the decision variable domains and allow them to converge more quickly.

4.2 The U-shape assembly line balancing problem - type 1 (UALBP-1)

The U-shaped assembly line balancing problem (UALBP) is a variant of the SALBP (Miltenburg and Wijingaard 1994, Urban 1998, Scholl and Klein 1999). This design allows for greater flexibility in the allocation of tasks to stations and hence has the potential for a shorter cycle time and/or a smaller number of stations/workers. An example is given in Figure 5, where the circles denote the tasks, and the dotted lines are the stations. We can see that the stations are numbered from left to right (Stn. 1 to Stn. 5). In general, given a line with m stations, each item is first moved from station 1 to m, and then returns back to station $m-1$, $m-2$, and so on, until it is completed at station 1. Each station except for the last one may contain tasks, which are performed before reaching station m (defined in Chiang et al. 2009 as the front of the line), and tasks that are performed after visiting station m on the way back to station 1 (defined in Chiang

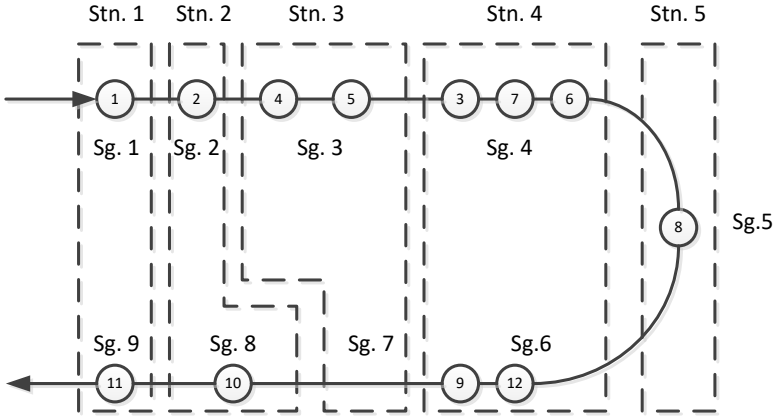


Figure 5. U-line example

et al. 2009 as the back of the line). Each such a station is called a crossover station.

In order to utilize the special constraints of SALBP-1, we define *a stage* in the line, as a set of tasks (can be also empty) that is performed by a worker before moving the item to the next worker. As shown in Figure 5, a U-shaped line with m stations, consists of $2m-1$ assembly stages; the first $m-1$ stages refer to the sets of tasks assigned to the *front* of stations 1 to $m-1$, stage m contains the tasks assigned to (the regular) station m, and stages $m+1$ to $2m-1$, refer to the sets of tasks assigned to the *back* of stations $m-1$ to 1. Hence, each station j contains two stages: stage j (in the front of the line) and stage $2m-j$ in the back of the line. Note that station m contains only one stage, since $j = 2m-j$ in this case. In the 5-station example, depicted in Figure 5, stages 1 and 9 belong to station 1, stages 2 and 8 belong to station 2, stages 3 and 7 belong to station 3, stages 4 and 6 belong to station 4 and stage 5 is solely carried out on station 5 (since the back and the front sides coincide in the last station). The advantage of using both stations and stages in the model is that the precedence constraints can be presented in the same way as in SALBP, using stages, while the cycle time constraint is then associated with the stations.

The two types of the UALBP, UALBP-1 and UALBP-2 are equivalent to the two types of the SALBP. Here we present a formulation for the type 1 problem, where the goal is to minimize the number of stations, but the adaptation to the type 2 is as straightforward as it was for the SALBP. In the CP model (16)-(21) below, we use the same notation as in our models for the SALBP-1, presented in Section 2. In particular, ub denotes an upper bound on the number of stations in an optimal solution. Any upper bound for SALBP-1 is also a valid UALBP-1 since the solution space of the former is a subset of the latter.

$$minimize\ m \tag{16}$$

$$\sum_{i=1}^{n} (x_i = j \vee x_i = 2m-j)\, t_i \leq ct \quad \forall j = 1, \dots, ub \tag{17}$$

$$x_i \leq 2m-1 \quad \forall i = 1, \dots, n \quad \forall i = 1, \dots, n \tag{18}$$

$$x_i \leq x_j \quad \forall i, j = 1, \dots, n : i \in P_j \tag{19}$$

$$x_i \in \{1, \ldots, 2ub - 1\} \quad \forall i \in 1, \ldots, n \tag{20}$$

$$m \in \{1, \ldots, ub\} \tag{21}$$

The CP model of the UALBP-1 is a slight modification of the SALBP-1 model, as the task assignment variable x_i is now associated with a stage. The objective function (16) minimizes the number of stations m. In Constraint (17), we use the ability of CP solvers to handle logical expressions directly. At each instance of this set of constraints, the processing times of the tasks assigned to the j^{th} stage and to the $(2m - j)^{th}$ stage are summed and bounded by the given cycle time. In Constraint (18), the number of stations and the indices of stages of each task are related. The precedence relationship are enforced in Constraint (19) in the same manner as in (4) of the original model.

Finally, the values of E_i, L_i and D_{ij} can be calculated exactly as in SALBP-1 and can be used to strengthen the formulation in the same way. Thus, Constraint (18) should be revised to

$$E_i \leq x_i \leq 2m - 1 - L_i \quad \forall i = 1, \ldots, n$$

and Constraint (4") of the SALBP-1, presented in Section 2, can be included in the model as is.

4.3 Task assignment and equipment selection to minimize equipment cost (TAESP)

The task assignment and equipment selection problem (TAESP), presented in Bukchin and Tzur (2000), is richer than the basic models discussed above. Here, in addition to allocating the tasks to stations along the assembly line, the planner also has to determine the type of machinery (e.g., robotic arm) installed at each station. Exactly one type is allowed at each station, and a cost is associated with each type of equipment. The processing time of each task is determined by the type of the equipment installed at its station. We denote by t_{ik} the processing time of task i when performed by equipment type k. In this model, the cycle time, ct, is given and fixed (as in SALBP-1), and the goal is to minimize the total equipment cost, which depends on the number of stations opened, m, and the type of equipment installed in each station. Note that this model can express situations in which a certain type of equipment k cannot be used to perform assembly task i. In such a case, we can set $t_{ik} > ct$.

To formulate the problem as a CP model, we use the same notation introduced in Section 2, including the decision variables, and introduce the following additional notation:

New parameters

ub An upper bounds on the number of stations in the line

lb A lower bounds on the number of stations in the line

q Number of equipment types

c_k Cost of installing equipment type k in a station

t_{ik} The processing time of assembly task i on equipment type k

A new decision variable

y_j The equipment type installed on station j

Using this notation, we can define the following CP model:

$$minimize \sum_{j=1}^{ub} c_{y_j}(j \leq m) \tag{22}$$

$$\sum_{i=1}^{n} (x_i = j)\, t_{i,y_j} \leq ct \quad \forall j = 1, \dots, ub \tag{23}$$

$$x_i \leq m \qquad \forall i = 1, \dots, n \tag{24}$$

$$x_i \leq x_j \qquad \forall i, j = 1, \dots, n : i \in P_j \tag{4}$$

$$y_j \leq 1 + (q-1) \cdot (j \leq m) \quad \forall j = 1, \dots, ub \tag{25}$$

$$x_i \in \{1, \dots, ub\} \quad \forall i \in 1, \dots, n \tag{26}$$

$$y_j \in \{1, \dots, q\} \tag{27}$$

$$m \in \{lb, \dots, ub\} \tag{28}$$

In the objective function, (22), the cost of the equipment in all the stations is minimized. Here, we exploit the ability of CP solvers to accept decision variables as indices for vectors of other decision variables. The cost of the equipment in station j is multiplied by zero if the station is beyond the end of the line. The cycle time is enforced by Constraint (23), again with the help of reification and advanced indexing. Note that this constraint is used to relate the equipment and the allocation decisions. Constraint (24) assures that no task is assigned beyond the end of the line. Constraint (4) is the same precedence constraint used in our CP model for the SALBP-1. Constraint (25) is used to reduce the symmetry in the problem and forces the (fictitious) type of the equip ment allocated to unused stations beyond to end of the line to assume type 1. In (26)-(28), the initial domains of the decision variables are specified.

4.4 Numerical experiments

In this section, we present the results of our experiment with the CP formulations of the SALBP-2, UALB-1 and the TAESP and compare them to the results obtained by standard MILP formulations available from the literature for these problems. For completeness, we present these MILP models in the appendix. We benchmarked the models with the medium and large instances (50 and 100 task problems, respectively) from Otto et al. (2013) dataset extended with relevant data where needed. Note that while this input was originally generated for the SALBP-1, their precedence graphs, processing times and cycle times are relevant for many other line balancing models. Recall that these data sets, each containing 525 instances, represent a large variety of problem parameters with different order strengths and distributions of the processing times.

The SALBP-1 data was adjusted to the other problem variants as follows. For the SALBP-2, a fixed number of stations was determined ($m = 10$), while the given cycle time, which is a decision variable in this model, was ignored. The input for the UALBP-1 is identical to the input of the SALBP-1 and was taken as is. For the TAESP we set the number of optional equipment types to four for the 50-task instances and five for the 100-task instances. The cost of each tool type was generated from $U(80,120)$. The processing times of each task on each tool (t_{ij}) were drawn from $U(0.5t_i, 1.5t_i)$ where t_i is the nominal processing time in the original datasets. All the random values were rounded to the nearest integer. In addition, for each task we selected randomly 0-2 tools that cannot perform it. The corresponding t_{ij} were set to be larger than the cycle time. The precedence graphs and the cycle times were taken from the original data set as is.

The experiment was conducted on standard Intel i7-4770 desktop with 16GB RAM. We used CP and MILP solvers from the CPLEX Studio Suite. All the data sets are available upon request from the authors as CPLEX Studio input file. Five minutes were allocated for each instance in both of the solution methods. The results of this experiment are summarized in Table 3.

For each problem (SALBP-1, UALBP-2 and TAESP), the results of two sets of data are presented (50 and 100 tasks), each containing 525 instances. The first two columns identify the dataset, namely, the problem type and the number of tasks. The third (fourth) column, associated with CP (resp., MILP), contains three values: the number of instances for which CP (resp., MILP) performed better than MILP (resp., CP), the average gap, in percentage, between CP (resp., MILP) and MILP (resp., CP) for these cases, and the maximal gap. In the next column, the number of instances in which the MILP solver could not obtained a feasible solution within the allocated five minutes is presented. Note that the CP solver could obtain a feasible solution within few seconds for each of the 3150 instances in this experiment. The right most columns present the overall average improvement achieved by the CP solver compared to the MILP in percentage. For each instance the CP improvement was calculated as

$$Improvment = \frac{MILP\ Solution - \ CP\ Solution}{MILP\ Solution}.$$

Table 3. CP versus MILP

Problem	Number of tasks	CP Better / Avg gap (%)/ Max gap (%)	MILP Better/ Avg gap (%)/ Max gap (%)	MILP Infeasible	Average improvement (%)
SALBP-2	50	59/0.09/0.17	129/0.11/0.28	0	-0.02
	100	100/0.05/0.10	173/0.05/0.15	0	-0.01
UALBP-1	50	64/4.51/9.68	0/--/--	0	0.55
	100	208/15.83/85.71	3/6.05/7.14	0	6.32
TAESP	50	287/4.93/24.44	23/1.57/5.36	0	2.62
	100	514/29.54/85.82	10/2.65/5.52	152	28.59

The average is calculated only for instances for which the MILP produced a feasible solution. All the differences between the two methods presented in the average improvement column are statistically significant with very small p-values (< 0.0005). This is not surprising considering the size of the datasets.

It is apparent from the table that the CP solver with our formulations performs substantially better than the MILP solver with the tested formulations for both the UALBP-1 and the TAESP. On the other hand, the performances of the MILP solver are marginally better for the SALBP-2. More specifically, when UALBP-1 and TAESP are concerned, CP outperforms MILP in significantly more instances than the opposite, and this effect is improved with the problem size (for example, 287 vs. 23 cases in the medium size TAESP, and 514 vs. 10 cases in the large scale TAESP). Moreover, one can see that the average and relative gap is much larger in the instances where CP outperforms MILP. When the SALBP-2 in concerned, MILP outperforms CP in more instances, however, the average and maximal gap in both directions are very small and decreases with the number of tasks in the instance. One can attribute this result, which is quite different than the result of SALBP-1 to that CP performs much better for integral objective functions with small values.

5 Concluding remarks

New effective constraint-programming (CP) formulations for solving various assembly line balancing problems are presented in this paper. In particular, the simple assembly line balancing problem type 1 and 2, the U-Shape assembly line balancing problem and the tasks assignment and equipment selection problem, are considered. CP is a generic and expressive modeling language, and its models can be solved by several commercial or open source software packages. Many problems can be formulated much more easily as a CP model compared to a well-known and widely used mixed-integer linear-programming approach. The proposed CP formulations are a conversion of MILP formulations from the literature, while exploiting the modelling flexibility of CP along with a new tightening constraint. The performance of our formulations, when solved by a commercial solver was compared to the best MILP formulations that are known to date using a state-of-the-art MILP solver and the SALOME branch & bound algorithm. The latter is known as an efficient exact algorithm for SALBP-1.

Results indicate that CP performs very well for SALBP-1 instances of up to 100 tasks. It provides optimal solutions for most of the instances and outperforms the MILP and SALOME in the number of best solutions obtained by each of the methods. In the dataset of 1000-task problem instances, SALOME was found to be superior. Still, in many cases, the CP-provided solutions are close to optimal. The MILP formulation was omitted from this experiment since it could not provide feasible solutions. A decision tree was constructed to define the parameter combinations for which CP can be used as an efficient heuristic for very large problem instances.

For the problem types other than SALB-1, our CP formulations were compared with equivalent MILP formulations, using 50 and 100 task test instances. Results demonstrate the superiority of the performances of the CP method over MILP for UALB-1 and the TAESP. For the SALBP-2 the MILP model delivers slightly better result than the CP approach but its advantage diminish with the size of the problems.

While this study is not the first to apply CP to line balancing problems, we believe that the line balancing research community has largely overlooked this paradigm, which is capable of providing very effective solutions for many generalizations of this problem. Moreover, the CP solvers have improved significantly in the last few years, emphasizing the great potential of applying CP for solving combinatorial optimization problems in general, and line balancing problems in particular.

We hope that future research will justify this statement with respect to additional variants of the assembly line balancing problems and other related problems in the domain of designing manufacturing systems. Moreover, due to the simplicity of the CP formulations and the ease of using the currently available solvers we believe that this method is very suitable for practitioners.

Appendix

In this appendix, we present the MILP formulations that were used to benchmark our CP models with respect the four line balancing variants, presented in the paper. All of these are identical or very similar to models that have already been presented in the literature and are presented here only for the sake of completeness.

The simple assembly line balancing problem - type 1 (SALBP-1)

Here we present the formulation of the SALBP-1 presented in Pastor and Ferrer (2009) that was used by Otto et al. (2013) in their extensive numerical experiment. We use the same notation, lower bound and upper bound, presented above for the CP model, and define the following decision variables.

x_{ij} A binary variable that equals 1, if task i is assigned to station j

y_j A binary variable that equals 1, if some tasks are assigned to station j.

$$minimize \sum_{j=lb+1}^{ub} jy_j \tag{29}$$

$$\sum_{j=E_i}^{m-L_i} x_{ij} = 1 \qquad \forall i = 1, \dots, n \tag{30}$$

$$\sum_{i=1}^{n} t_i x_{ij} \leq ct \qquad \forall j = 1, \dots, lb \tag{31}$$

$$\sum_{i=1}^{n} t_i x_{ij} \leq y_j ct \qquad \forall j = lb+1, \dots, ub \tag{32}$$

$$\sum_{j=E_i}^{m-L_i} jx_{ij} \leq \sum_{j=E_k}^{m-L_k} jx_{ik} \qquad \forall i, k = 1, \dots, n: i \in P_k \tag{33}$$

$$x_{i,L_i-q} \leq y_{ub-q} \qquad \forall i = 1, \dots, n, \qquad q = 0, \dots, ub - lb - 1$$

$$x_{ij} \in \{0,1\} \qquad \forall i = 1, \dots, n, \qquad j = E_i, \dots, m - L_i \tag{34}$$

$$y_j \in \{0,1\} \qquad j \in lb+1, \dots, ub \tag{35}$$

The objective function (29) is indirectly minimizing the total number of stations and also aid in breaking symmetry in the solution space; constraint (30) implies that each task i is assigned to

one and only one workstation; constraints (31) and (32) ensure that the total task processing time assigned to workstation j does not exceed the cycle time; in addition constraint (32) also relate the x and y variables. Constraint (33) imposes the precedence conditions. The domains of the decision variables are defined in (34) and (35).

The simple assembly line balancing problem - type 2 (SALBP-2)

In this section, we present a formulation for the SALBP-2 adapted from Pastor and Ferrer, (2009). We use the same notation as presented above for the CP model and define the following decision variables:

ct The cycle time (exactly as in our CP model)

x_{ij} A binary variable that equals 1, if task i is assigned to station j

$$minimize\ ct \tag{36}$$

$$\sum_{i=1}^{n} t_i x_{ij} \leq ct \quad \forall j = 1, \dots, m \tag{37}$$

$$\sum_{j=E_i}^{m-L_i} j x_{ij} \leq \sum_{j=E_k}^{m-L_k} j x_{ik} \quad \forall i, k = 1, \dots, n: i \in P_k \tag{38}$$

$$\sum_{j=E_i}^{m-L_i} x_{ij} = 1 \quad \forall i = 1, \dots, n \tag{39}$$

$$x_{ij} \in \{0,1\} \quad \forall i = 1, \dots, n, \quad j = E_i, \dots, m - L_i \tag{40}$$

The objective function (36) is minimizing the cycle time. Constraint (37) relates the ct variable to the total time of the tasks assigned to each station. Constraint (38) is the precedence constraint. Equation (39) stipulate that each task is assigned to one station in each relevant station range $\{E_i, \dots, m - L_i\}$. Note that we use the notation L_i with the same semantic as in (14) in our CP model. That is, L_i denote the minimal number of stations between the station of task i and the end of the line. E_i and L_i are calculated based on the upper bound obtained by the same

procedure as described for the CP model above. In (40) the domain of the x_{ij} variables is defined.

The U-shaped assembly line balancing problem – type 1 (UALBP-1)

Our formulation of the U-Shaped assembly line balancing problem is mathematically equivalent to standard formulation in the literature (Urban, 1998) but it is presented in a notation that is similar to the notation of our CP formulation. Namely, the tasks are assigned to production staged rather than to stations, and each worker along the line is responsible to stages that are related to both sides of the station. The parameters ub and lb represent bounds on the number of stations (and not stages) in the line. The notation of the problem parameters is identical to the notation of the SALBP-1 and we defined the following decision variables.

x_{ij} A binary variable that equals 1 if task i is assigned to production stage j

z_k A binary variable that equals 1 if station k is used

$$minimize \sum_{k=lb+1}^{ub} z_k \tag{41}$$

$$\sum_{i=1}^{n} t_i\left(x_{ik} + x_{i,2ub+1-k}\right) \leq ct \quad \forall k = 1, \dots, ub \tag{42}$$

$$\sum_{j=1}^{2ub-1} jx_{ij} \leq \sum_{j=1}^{2ub-1} jx_{ih} \quad \forall i, h = 1, \dots, n: i \in P_h \tag{43}$$

$$\sum_{j=1}^{2ub-1} x_{ij} = 1 \quad \forall i = 1, \dots, n \tag{44}$$

$$x_{ik} + x_{i,2ub+1-k} \leq z_k \quad \forall i = 1, \dots, n, k = lb+1, \dots, ub \tag{45}$$

$$z_k \geq z_{k+1} \quad \forall k = lb+1, \dots, ub-1 \tag{46}$$

$$x_{ij} \in \{0,1\} \quad \forall i = 1, \dots, n, \quad j = E_i, \dots, (2ub - 1) - L_i \tag{47}$$

$$z_k \in \{0,1\} \quad \forall k = lb + 1, \dots, ub \tag{48}$$

The objective function (41) minimizes the number of stations added to the lower bound on this number. Constraint (42) assures that the total time allocated at each station does not exceed the cycle time. Constraint (43) is a precedence constraint and since the tasks are assigned to stages it is identical to the one in the formulations of the SALBP. Constraint (44) stipulates that each task is assigned to one of the stages. Constraint (45) assures that a station is "opened" if a task is assigned to either of its sides. Constraint (46) is a symmetry breaking constraint, which may ease the search procedure. Finally, the domain of the variables is defined in (47) and (48). Note that the values of E_i and L_i are calculated based on the given cycle time, resulting a relatively weak bound.

The task assignment and equipment selection problem (TAESP)

Here we bring the formulation presented in Bukchin and Tzur (2000) and strengthen it with an additional symmetry breaking constraint that in our preliminary experimentations greatly reduced the run time. Again, the notation of the parameters of the problem is the same as in the notation introduced for the CP model in Section 4.3, and thus is not repeated here. The decision variables are defined as follow.

x_{ijk} A binary variable that equals 1, if task i is assigned to station j and processed by equipment type k.

y_{jk} A binary variable that equals 1, if equipment type k was selected for station j.

$$minimize \sum_{j=1}^{ub} \sum_{k=1}^{q} c_k y_{jk} \tag{49}$$

$$\sum_{i=1}^{n} t_{ik} x_{ijk} \leq y_{jk} ct \quad \forall j = 1, \dots, ub, k = 1, \dots, q \tag{50}$$

$$\sum_{j=1}^{ub} j \sum_{k=1}^{q} x_{ijk} \leq \sum_{j=1}^{ub} j \sum_{k=1}^{q} x_{ihk} \qquad \forall i, h = 1, \dots, n: i \in P_h \tag{51}$$

$$\sum_{j=1}^{ub} \sum_{k=1}^{q} x_{ijk} = 1 \qquad \forall i = 1, \dots, n \tag{52}$$

$$\sum_{k=1}^{q} y_{jk} \leq 1 \qquad \forall j = 1, \dots, ub \tag{53}$$

$$\sum_{k=1}^{q} y_{jk} \geq \sum_{k=1}^{q} y_{j+1,k} \qquad \forall j = 1, \dots, ub - 1 \tag{54}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i = 1, \dots, n, \qquad j \in 1, \dots, ub, k = 1, \dots, q \tag{55}$$

$$yj_k \in \{0,1\} \quad \forall j = 1, \dots, ub, \qquad k = 1, \dots, q \tag{56}$$

The objective function (49) minimizes the total cost of all the selected tools. Constraint (50) assures that the cycle time is not violated and that the process time of each task is determined by the equipment type selected for this station. Constraint (51) is the adaptation of the standard precedence constraint to this problem. Constraint (52) stipulates that each task is assigned once. Constraint (53) assures that at most one equipment type is selected for each potential station. Constraint (54) eliminates symmetry in the solution space by forcing all the active (equipped) station to the beginning of the line. Constraints (55) and (56) define the domains of the decision variables.

References

Amen, M., 2006. Cost-oriented assembly line balancing: model formulations, solution difficulty, upper and lower bounds. *European Journal of Operational Research*, 168, 747–770.

Apt, K.Y, 2003. Principles of Constraint programming, 1st Edition, Cambridge University Press, Amsterdam.

Arcus, A.L., 1965. COMSOAL: A Computer Method of Sequencing Operations for Assembly Lines, International Journal of Production Research, 4(4), 259-277.

Baptiste, P., Le Pape, C. and Nuijten, W., 2012. Constraint-based scheduling: applying constraint programming to scheduling problems (Vol. 39). Springer Science & Business Media.

Baybars, I., 1986. A survey of exact algorithms for the simple line balancing problem. *Management Science*, 32 (8), 909–932.

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

Bockmayr, A. and Pisaruk, N., 2001. Solving assembly line balancing problems by combining IP and CP. Sixth Annual Workshop of the ERCIM Working Group on Constraints.

Bowman, E.H., 1960. Assembly-line balancing by linear programming. *Operational Research*, 8, 385–389.

Bukchin, J. & Tzur, M., 2000. Design of flexible assembly line to minimize equipment cost, *IIE Transactions* 32, 585-598.

Bukchin, Y. and Zaides, I., 2016. The general consecutive multiprocessor job scheduling for minimizing the number of tardy jobs, MSc thesis, Dept. of Industrial Engineering, Tel-Aviv University.

Chiang, W. C., Kouvelis, P. & Urban, T.L., 2007, Line balancing in a just-in-time production environment: balancing multiple U-lines, *IIE Transactions*, 39(4), 347-359.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I.H., 2009. The WEKA Data Mining Software: An Update; *SIGKDD Explorations*, 11(1).

Held, M., Karp, R.M. and Shareshian, R., 1963. Assembly line balancing-Dynamic programming with precedence constraints. *Operations Research*, 11, 442–459.

Hoffmann, T.R., 1992. EUREKA: A hybrid system for assembly line balancing. *Management* Science, 38, 39-47.

Jackson, J.R., 1956. A computing procedure for a line balancing problem. *Management Science*, 2, 261–271.

Johnson R.V., 1988. Optimally balancing large assembly lines with "FABLE". *Management Science*, 34(2), 240–253.

Karp, R.M., 1972. Reducibility among Combinatorial Problems. In Miller, R.E. and J.W. Thatcher (eds.): Complexity of Computer Computation, Plenum Press, New York, 85-103.

Miltenburg, G.J. & Wijingaard, J., 1994. The U-line Line Balancing Problem, *Management Science*, 40 (10), 1378-1388.

Morrison, D.R., Sewell, E.C. & Jacobson, S.H., 2014. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset, *European Journal of Operational Research*, 236, 403-409.

Otto, A., Otto, C., Scholl, A., 2011. SALBPGen – A Systematic Data Generator for (simple) Assembly Line Balancing. Jena Research Papers in Business and Economics, 05/2011.

Otto, A., Otto, C. and Scholl, A., 2013, Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research*, 228, pp. 33–45.

Pastor, R., Corominas, A. and Lusa, A., 2004. Different ways of modelling and solving precedence and incompatibility constraints in the assembly line balancing problem. *Frontiers in Artificial Intelligence and Applications*, 113, 359–366.

Pastor, R. & Ferrer, L., 2009. An improved mathematical program to solve the simple assembly line balancing problem. *International Journal of Production Research*, 47(11), 2943-2959.

Pastor R., Ferrer L. & García A., 2007. Evaluating Optimization Models to Solve SALBP. In: Gervasi O., Gavrilova M.L. (eds) Computational Science and Its Applications – ICCSA 2007. ICCSA 2007. Lecture Notes in Computer Science, vol 4705. Springer, Berlin, Heidelberg

Patterson, J.H. and Albracht, J.J., 1975. Assembly-line balancing: zero-one programming with Fibonacci search. *Operations Research*, 23, 166–172.

Quinlan, J.R., 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Rekiek, B., Dolgui, A., Delchambre, A. and Bratcu, A., 2002. State of art of optimisation methods for assembly line design. *Annual Reviews in Control*, 26, 163–174.

Salveson, M.E., 1955. The assembly line balancing problem. *Journal of Industrial Engineering*, 6(3), 18–25.

Schaus, P., 2009. Solving Balancing and Bin-Packing problems with Constraint Programming. PhD thesis, Université catholique de Louvain Louvain-la-Neuve, Belgium.

Schaus, P., and Deville, Y., 2008. A Global Constraint for Bin-Packing with Precedences: Application to the Assembly Line Balancing Problem. Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence.

Scholl, A., 1999. *Balancing and sequencing assembly lines*, 2nd ed. Physica, Heidelberg

Scholl, A., & 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. & Klein, R., 1997. SALOME: a bidirectional branch and bound procedure for assembly line balancing. *INFORMS Journal on Computing*, 9, 319–334.

Scholl, A., & Klein, R., 1999. ULINO: Optimally balancing U-shaped JIT assembly lines, *International Journal of Production Research*, 37(4), 721-736.

Schrage, L. & Baker, K.R., 1978. Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, 26, 444–449.

Sewell, E.C. & Jacobson, S.H., 2012. A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem, *INFORMS Journal on Computing*, 24(3), 433–442.

Shannon, C.E., 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27, 379–423 and 623–656.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In International Conference on Principles and Practice of Constraint Programming (pp. 417-431). Springer Berlin Heidelberg.

Talbot, F.B. & Patterson, J.H., 1984. An integer programming algorithm with network cuts for solving the assembly line balancing problem. *Management Science*, 30, 85–99.

Thesen, A., 1977. Measures of the restrictiveness of project networks. *Networks*, 7, 193–208.

Topaloglu, S., Salum, L. & Supciller, A.A., 2012. Rule-based modeling and constraint programming based solution of the assembly line balancing problem, *Expert Systems with Applications*, 39 (3), 3484-3493.

White, W.W., 1961. Comments on a paper by Bowman. *Operations Research*, 9, 274–276.

Urban, T. L., 1998. Note. Optimal Balancing of U-Shaped Assembly Lines, *Management Science*, 44(5), 738-741.