Borba, L.; Ritt, M.; Miralles Insa, CJ. (2018). Exact and heuristic methods for solving the Robotic Assembly Line Balancing Problem. European Journal of Operational Research. 270(1):146-156. https://doi.org/10.1016/j.ejor.2018.03.011

Exact and Heuristic Methods for solving the Robotic Assembly Line Balancing Problem

Leonardo Borba, Marcus Ritt, Cristóbal Miralles

PII: S0377-2217(18)30220-0
DOI: 10.1016/j.ejor.2018.03.011
Reference: EOR 15030

To appear in: *European Journal of Operational Research*

Received date: 25 October 2016
Revised date: 5 March 2018
Accepted date: 7 March 2018

Please cite this article as: Leonardo Borba, Marcus Ritt, Cristóbal Miralles, Exact and Heuristic Methods for solving the Robotic Assembly Line Balancing Problem, *European Journal of Operational Research* (2018), doi: 10.1016/j.ejor.2018.03.011

Highlights

- We propose new algorithms for minimizing the cycle time in robotic assembly lines.
- The first is a branch-bound-and-remember method with cyclic best-first search.
- The second is an iterative beam search.
- Both methods use newly proposed lower bounds and dominance rules.
- Our methods improve the results from the literature and are faster.

Exact and Heuristic Methods for solving the Robotic Assembly Line Balancing Problem

Leonardo Borba[a], Marcus Ritt[a], Cristóbal Miralles[b]

[a]*Instituto de Informática, Universidade Federal do Rio Grande do Sul, Av. Bento Gonçalves 9500, Porto Alegre-RS, Brasil*
[b]*Departamento de Organización de Empresas, Universitat Politècnica de València, Camino de Vera, s/n, Valencia, España*

Abstract

In robotic assembly lines the task times depend on the robots assigned to each station. Robots are considered an unlimited resource and multiple robots of the same type can be assigned to different stations. Thus, the Robotic Assembly Line Balancing Problem (RALBP) consists of assigning a set of tasks and a type of robot to each station, subject to precedence constraints between the tasks. This paper proposes a lower bound, and exact and heuristic algorithms for the RALBP. The lower bound uses chain decomposition to explore the graph dependencies. The exact approaches include a novel linear mixed-integer programming model and a branch-bound-and-remember algorithm with problem-specific dominance rules. The heuristic solution is an iterative beam search with the same rules. To fully explore the different characteristics of the problem, we also propose a new set of instances. The methods and algorithms are extensively tested in computational experiments showing that they are competitive with the current state of the art.

*Corresponding author
Email addresses: lmborba@inf.ufrgs.br (Leonardo Borba), mrpritt@inf.ufrgs.br (Marcus Ritt), cmiralles@omp.upv.es (Cristóbal Miralles)

Keywords: Production, Robotic Assembly line balancing,
Branch-bound-and-remember, Beam Search

1. Introduction

The range of tasks that can be performed by robots has increased significantly in the past decades Purnell (1998); Henrich & Wörn (2000); Baeten et al. (2008). Robots are especially efficient for the repetitive small tasks, which are common in flexible assembly lines Milberg & Schmidt (1990); Pires & Sá da Costa (2000). To model this kind of problem Rubinovitz et al. (1993) have proposed the Robotic Assembly Line Balancing Problem (RALBP). The RALBP extends the Simple Assembly Line Balancing Problem (SALBP) Baybars (1986) by adding the concept of robots. In the SALBP a set of partially ordered tasks must be assigned to a linearly ordered set of stations, such that the task precedences agree with the linear order of the stations Scholl & Becker (2006). Each task t has a task time p_t and the station time of each station is given by the total time of the tasks assigned to that station. The greatest station time defines the bottleneck of the line and therefore the cycle time of the line. Possible objectives are to minimize the cycle time, or the number of stations, or both, or simply finding a valid solution. In the case of the RALBP, a robot must be additionally assigned to each workstation, and is responsible for performing the tasks assigned to that station. Robots are usually heterogeneous and the time to execute a task depends on the robot performing it.

According to the definition of Rubinovitz et al. (1993), the RALBP is composed of a set T of nonpreemptive tasks. These tasks must be assigned to a set of workstations S. There is also a set of types of robots R and one robot of type $r \in R$ must be designated to each workstation.

The time p_{tr} needed to execute the task $t \in T$ is deterministic and depends on the type r of the robot assigned to the station where task t is performed. The station time C_s is the sum of the task times p_{tr} of the tasks t assigned to station s, given that robot r is responsible for that station. The cycle time of the line then is the largest station time $C = \max_{s \in S} C_s$.

We also have a set of precedence relations A. If $(t,t') \in A$ task t precedes task t'. For such a pair, task t' can only be executed at the same station as t or at a station following the station performing t.

Based on these assumptions, the RALBP has two dependent variables: the number of workstations $|S|$ and the cycle time C of the line. In this paper we propose solutions for the minimization of C given a fixed $|S|$. This problem is called RALBP-2 in the literature.

1.1. Related Work

Assembly Line Balancing research, which was traditionally focused on the SALBP defined by Salveson (1955) through several well-known simplifying hypotheses, has been recently enriched by many realistic features that have been successively added in the literature (see the reviews of Scholl & Becker (2006); Becker & Scholl (2006); Boysen et al. (2007, 2008); Battaïa & Dolgui (2013)).

In the particular case of the consideration of heterogeneity in the resources involved, the RALBP model proposed by Rubinovitz et al. (1993) was a clear antecedent of this trend, inspiring further approaches like the Assembly Line Worker Assignment and Balancing Problem (ALWABP) first presented by Miralles et al. (2007). In this paper a new set of hypotheses motivated by assembly lines in sheltered work centres for disabled persons are defined, where all workers are considered heterogeneous. The fundamental difference to the RALBP is that the

resources available are constrained: in the RALBP the same type of robot can be assigned to multiple workstations if convenient, whereas in ALWABP there is a set of unique workers that can only be assigned once.

Despite the clear differences, exact methods for the SALBP Salveson (1955); Scholl & Becker (2006); Klein & Scholl (1996); Morrison et al. (2014); Vilà & Pereira (2013) and the ALWABP Miralles et al. (2007, 2008); Vila & Pereira (2014); Borba & Ritt (2014) can be applied to the RALBP. Notably, the SALBP lower bounds Scholl & Becker (2006), some dominance rules (e.g. the maximum load rule Jackson (1956)), and the search strategies Scholl & Becker (2006); Morrison et al. (2014); Vilà & Pereira (2013); Vila & Pereira (2014); Borba & Ritt (2014). However, many of the methods for the SALBP and ALWABP largely rely on properties of these problems and cannot be adapted to the RALBP. For instance, Jackson's dominance rule Jackson (1956), proposed for SALBP, highly depends on station-independent task times to define potential domination between the tasks. Similarly, the problem cannot be relaxed to the unbounded parallel machines scheduling problem Borba & Ritt (2014), like the ALWABP.

In the RALBP literature, two lower bounds adapted from the SALBP are used Rubinovitz et al. (1993): $LM_1 = P^-/C$ for the RALBP-1 and $LC_1 = P^-/|S|$ for the RALBP-2, where $P^- = \sum_{t\in T} \min_{r\in R} p_{tr}$ is the sum of the minimal task times Scholl & Becker (2006). These two lower bounds relax the precedence constraints and consider the tasks to be preemptive, dividing them equally among the stations. Rubinovitz et al. (1993) also proposed a best-first search branch-and-bound algorithm for the type 1 of the problem. For the RALBP-2, Levitin et al. (2006) have proposed a genetic algorithm (GA). Their algorithm uses a common genotype, and mutation and crossover operators of genetic algorithms

for the SALBP Rubinovitz & Levitin (1995). They also improved the results of this GA using a hill climbing algorithm. A Particle Swarm Optimization (PSO) method and an hybrid method of PSO and a Cuckoo Search algorithm were proposed by Mukund Nilakantan et al. (2015), with the best known results for the RALBP. They also introduced a mathematical formulation for the RALBP, here referred as M_1. Model M_1 is a quadratic mixed-integer programming model that uses two-index variables. Finally, Gao et al. (2009) solves a different RALBP problem where only one robot of each type is available and, therefore, can only be assigned once. This problem is equivalent to the ALWABP, and the ALWABP has significantly better results in the literature Miralles et al. (2007); Chaves et al. (2007); Miralles et al. (2008); Chaves et al. (2009); Blum & Miralles (2011); Moreira et al. (2012); Mutlu et al. (2013); Vila & Pereira (2014); Borba & Ritt (2014); Polat et al. (2016).

1.2. Outline of the Paper

In the next sections we will present solution procedures for the RALBP-2. Section 2 introduces the mathematical formulation of the problem. In Section 3 we investigate a novel lower bound. It uses the task dependencies to improve the lower bounds in the literature. Furthermore, a branch-bound-and-remember (BBR) method for the problem is proposed in Section 4, with a series of dominance rules adapted or created for RALBP. An iterative beam search using the same dominance rules and lower bounds is then proposed in Section 5. The computational experiments for the mixed-integer programming (MIP) models, the lower bounds, the heuristics and the branch-and-bound method are presented in Section 6, as well as a new set of instances to explore different characteristics of the problem.

2. Mathematical Formulation

We propose using a model M_2, that avoids the quadratic functions of model M_1. We also use the techniques proposed by Ritt & Costa (2015) to improve the precedence constraints. The resulting model M_2, with notation described in Table 1, can then be defined by

$$M_2 = \textbf{minimize} \quad C, \tag{1}$$

$$\textbf{subject to} \quad \sum_{t \in A_s} p_{tr} x_{ts} \leq C + M_r(1 - y_{sr}), \quad \forall s \in S, r \in R, \tag{2}$$

$$\sum_{r \in R} y_{sr} = 1, \quad \forall s \in S, \tag{3}$$

$$\sum_{s \in I_t} x_{ts} = 1, \quad \forall t \in T, \tag{4}$$

$$\sum_{k \in I_i | k \leq s} x_{ik} \geq \sum_{k \in I_j | k \leq s} x_{jk}, \quad \forall i \in T, j \in F_i, s \in S, \tag{5}$$

$$x_{ts} \in \{0,1\}, \quad \forall s \in S, t \in A_s, \tag{6}$$

$$y_{sr} \in \{0,1\}, \quad \forall s \in S, r \in R. \tag{7}$$

The model minimizes the cycle time C (1), defined in constraint (2). Since the right side of constraint (2) must be free to assume any value when y_{sr} is not set and, given a lower bound $\underline{C}$ on the cycle time, we can assume that $M_r \geq \sum_{t \in T} \{p_{tr}\} - \underline{C}$, for each robot r. Constraints (3) and (4) ensure that each task will be performed and that each station will have a robot assigned to it. Constraint (5) defines the precedence relations between the tasks. The robots do not affect the dependencies, therefore the precedence constraints for the SALBP can be directly applied to the RALBP. We ensure that the variables x_{ts} are only defined for the tasks that can be performed in a given station s ($t \in A_s$). Model M_2 is linear and has $O(|T||S| +$

Table 1: Notation used in the article.

T	set of tasks;
R	set of robots;
S	set of workstations;
E_t and L_t	the earliest and latest station, respectively, where a task can be placed;
A_s	$A_s = \{t \in T \mid E_t \leq s \leq L_t\}$, the set of tasks that can be assigned to station s;
I_t	$I_t = \{s \in S \mid E_t \leq s \leq L_t\}$, the set of stations where task t can be performed;
$G(T,A)$	precedence graph of tasks, where $(t,t') \in A$ indicates that task t must be performed before task t';
$G^*(T,A^*)$	the transitive closure of graph $G(T,A)$;
p_{tr}	execution time of task t by robot r;
$P_t \subseteq T$	set of immediate predecessors of task t;
$F_t \subseteq T$	set of immediate followers of task t;
$P_t^* \subseteq T$	set of all predecessors of task t;
$F_t^* \subseteq T$	set of all followers of task t;
C	the cycle time of a solution;
M_r	a constant equal to $\sum_{t \in T} \{p_{tr}\} - \underline{C}$;
x_{ts}	1 if task t is assigned to station s, and 0 otherwise;
y_{sr}	1 if robot r is assigned to station s, and 0 otherwise.

$|S||R|)$ variables and $O(|S||R| + |T|^2|S|)$ constraints.

3. Lower Bounds

Lower bound LC_1 Rubinovitz et al. (1993) relaxes precedence constraints. In our proposed lower bound LC_2 we maintain some of the precedence constraints such that we have a set of task chains T_c and remove all the other precedence constraints. Since the precedence constraints of this set of chains are present in the original graph, all the solutions that are valid for the original problem are valid for the adapted instance. Therefore, an optimal solution for the new instance is a valid lower bound for the original problem.

To decompose the original graph into a set of chains T_c, we iteratively select the longest chain in the graph until all tasks have been assigned to one of the chains. The longest chain is selected since this increases the chance that multiple

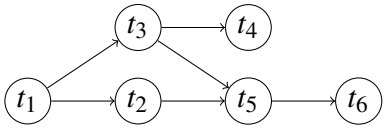

Figure 1: Example of a precedence graph of an instance with six tasks.

tasks are assigned to the same station and the same robot has to perform them.

Consider the example shown in Figure 1. We first select the longest chain (t_1, t_2, t_5, t_6) of the graph. Then, only tasks t_3 and t_4 remain and they form the second chain.

After selecting the chains we determine for each chain the smallest total task time, considering that some groups of tasks must be performed at the same station and therefore by the same robot. The minimum total task times for a set of chains is the sum of the minimum total task time of each chain. For a given chain $Q = (q_0, q_1, \ldots, q_{n-1})$ the minimum total task time can be computed by dynamic programming. Let

$$M_Q(t,s) = \begin{cases} 0, & \text{if } t = n, \\ \infty, & \text{if } t < n \wedge s = 0, \\ \min\limits_{t<t'\leq n} \min\limits_{r\in R} \sum\limits_{t\leq t''<t'} p_{t'',r} + M_Q(t', s-1), & \text{otherwise,} \end{cases} \tag{8}$$

be the minimum total task time for tasks $q_t, \ldots, q_{n-1}$ on s stations. Then $M_Q(0, |S|)$ is the minimum total task time for chain Q. There first two conditions handle the base case: if all tasks have been assigned, the sum of the remaining tasks is zero; and if there are no stations left but still tasks to assign, it is impossible to solve the problem. Otherwise we assign the tasks in the range $[t, t')$, for some $t < t' \leq n$ to the current station, and assign the robot that executes them fastest to that station.

To the time for executing these tasks we have to add the minimum total time for executing the remaining tasks starting with t' on one station less.

For example, consider the case where we have a chain $Q = (q_0, q_1, q_2, q_3)$ of length four, two stations and two robots. The tasks have times $p_{11} = 1$, $p_{12} = 2$, $p_{21} = 2$, $p_{22} = 1$, $p_{21} = 1$, $p_{22} = 2$, $p_{41} = 2$, and $p_{42} = 1$. The sum of the minimum task times is 4. However, at least two subsequent tasks must be assigned to the same robot. Indeed, the result of the minimum total task time obtained by recurrence (8) is $M_Q(0,2) = 5$.

Function M can be determined by dynamic programming in time $O(n^2|S|\,|R|)$ for a chain of length n, and to calculate the result for all the chains, the total complexity is $O(|T|^2|S|\,|R|)$. We can define lower bound LC_2 by

$$LC_2 = \sum_{Q \in T_c} M_Q(0,|S|)/|S|.$$

Lower bound LC_2 is the minimum possible sum of task times considering that some tasks must be assigned to the same station. In the best case, the total sum of the task times will be equally distributed among stations, and therefore dividing the minimum sum of task times by the cycle time we have a valid lower bound on the number of stations.

A longest path can be found in time $O(|T| + |A|)$ and this process is repeated at most $|T|$ times. Thus, computing LC_2 takes total time $O(|T|^2|S|\,|R| + |T|\,|A|)$.

4. An Iterative Branch-Bound-and-Remember Method

The optimal solution for the RALBP-2 is the smallest cycle time C for which a valid solution can be found. To find the value C, we iterate over cycle times

in the interval $[\underline{C},\overline{C}]$, where $\underline{C}$ is a lower bound and $\overline{C}$ an upper bound for the problem. We initially set the cycle time to the upper bound $C = \overline{C}$. The value $\overline{C}$ is the result of our heuristic method defined in Section 5. Afterwards the cycle time C is decremented one unit at a time until it is impossible to find a valid solution for C. The optimal cycle time C^* then is $C+1$. Therefore, we only need to prove infeasibility for the smallest tested cycle time. The problem of verifying if there is a valid solution for a fixed number of stations and a fixed cycle time C is called RALBP-F in the literature.

4.1. A Branch-Bound-and-Remember Algorithm

For the RALBP-F we propose using a station-oriented BBR algorithm. Our branching strategy consists of filling one station at a time. In the initial node of the branch-and-bound method, we generate all possible station loads for the first workstation. A branch is generated for each station load and the first station is closed. Then, the method expands the generated branches. The expansion process generates all the station loads for the first open station of the current node. A solution is valid when all tasks are assigned to less than $|S|$ stations.

To decide the order in which the branches are explored, we use a cyclic best-first search (CBFS). In the cyclic best-first search the partial solutions are divided in levels. In the RALBP, each level k contains all the partial solutions with k stations. At each iteration of the algorithm, the method selects the solution of the least lower bound and expands it, adding the new branches to the next level. Lower bounds LC_1 and LC_2 can be used to prioritize the solutions and their performance will be evaluated in Section 6.3. When lower bound LC_2 is used, the chain decomposition is computed at each node anew, to improve the bound.

In our method, level zero starts with only one partial solution with no stations

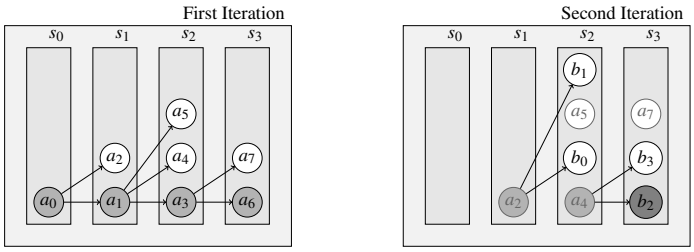

Figure 2: An example of CBFS. In the first iteration the method expands the partial solutions a_0, a_1, a_3 and a_6. In the second iteration, the first priority queue is empty and the method continues to expand partial solutions a_2, a_4 and b_2, which is a valid solution for the given cycle time. The method ends after expanding five nodes.

loaded. This branch is expanded and the new partial solutions are added to level one. After that, the method selects the best solution and expands it by assigning tasks to the second station. The method continues until the last level and if no valid solution is found the method returns to the first level with partial solutions yet to expand, starting a new iteration. The process is repeated until a solution is found or all the partial solutions have been explored. In this case, we know that there are no valid solutions for the current cycle time. The CBFS is exemplified in Figure 2.

4.2. Dominance Rules

To reduce the number of partial solutions explored, we use four dominance rules:

- **Maximal Station Load Rule Jackson (1956):** A station is said to be maximally loaded if no other task can be assigned to the current station without exceeding the cycle time. We only consider maximally loaded stations. If a partial station load is not maximal, a task can be added to the current station. The new solution with this task dominates the previous one.

- **Feasible Set Dominance Rule Schrage & Baker (1978):** Given a partial solution v_1 with a set of tasks already assigned to the first m_1 stations, if the same set of tasks has already been assigned to another set of m_2 stations in another solution v_2, with $m_2 \leq m_1$, then solution v_2 dominates solution v_1, because assigning the remaining tasks would take the same number of stations in both cases. To apply this rule, our algorithm memorizes the tasks already assigned in a given partial solution and every time a branch visits a partial solution with a set of tasks visited previously, the branch is cut.

- **Late Acceptance Dominance Rule Sewell & Jacobson (2011):** If none of the tasks of a station load has successors, this set of tasks can be assigned to any future station. A station load has no successors if no unassigned task succeeds the tasks in the current station. Therefore, to avoid multiple equivalent solutions, if it is possible to postpone the current station load to a future station, the current partial solution can be eliminated.

- **Best Robot Dominance Rule:** Since we can use the same type of robot as many times as needed, the assignment of a robot to a station is independent from the rest of the solution. Therefore for each station with a set of tasks assigned, we only need to consider the robot that executes the set of tasks fastest. This rule can be combined with the Maximal Station Load Rule and we can ignore a station load if there exists any other robot for which the current station load is not maximal.

Given these dominance rules, the number of nodes explored by the BBR algorithm can be bounded as follows. First, consider a fixed cycle time. If we decompose the precedence graph into the smallest number of chains w by a Dil-

worth chain decomposition (w is the partial order's width, see Dilworth (1950)), and the length of the longest chain in this decomposition is l, then there are at most $(l+1)^w$ partial solutions. This holds since each partial solution can be described by the already assigned tasks, which are uniquely defined by a position in $[0,l]$ for each of the w chains. By the feasible set dominance rule, each set of assigned tasks is visited only once. Therefore, since at most $\overline{C}-\underline{C}$ different cycle times must be considered, the total number of nodes is bounded by

$$O((\overline{C}-\underline{C})l^w). \tag{9}$$

5. An Iterative Beam Search

By running the BBR method with a time limit, we obtain a heuristic for the RALBP. The BBR method, however, stores branches, which will probably never be used in the case of a limited execution. A beam search reduces this problem by storing only a few of the best partial solutions found. Since in our case, our method is a cyclic best-first search, we limit the number of partial solutions stored by level. As in the cyclic best first search, the best partial solution of a given level is selected and all station loads for the next level are generated. However, the method keeps only the best b_w solutions of that level. The best solutions are those with the smallest lower bound, and the lower bounds used are the same as the lower bounds in the branch-bound-and-remember method. All the dominance rules are also applied.

The method is iterative. For each cycle time the heuristic searches for a valid solution for at most time h. Because of the limited beam width b_w the time for each iteration for small cycle times is not as high as for the branch-and-bound

algorithm and we can apply a binary search to test the cycle times. Here, to define the initial upper bound $\overline{C}$, we use the sum of the task times as performed by the best robot to perform all tasks $\min_{r\in R}\sum_{t\in T} p_{tr}$, and we set the initial lower bound to $\underline{C} = LC_1 - 1$. At each iteration of the method, we test cycle time $\lfloor(\underline{C}+\overline{C})/2\rfloor$. If a valid solution is found for the given cycle time, we set $\overline{C}$ to the current tested cycle time, otherwise we set $\underline{C}$ to the cycle time being tested.

We also do not need to generate all the station loads for a given station. The lower bound LM_1 can be improved during the generation of the station loads. If a partial load is already worse than the worst partial solution stored for the next level and the next level already has b_w partial solutions, then the solution with the current set of tasks will not be added to the next level and this branch is not explored. The partial lower bound LM'_1 can be defined by

$$LM'_1 = s + \min_{r\in R}\sum_{t\in T_s} p_{tr} + \sum_{t\in U}\min_{r\in R} p_{tr}$$

where s is the number of stations already fully loaded in the current partial solution, T_s is the set of tasks already assigned to the current station and U is the set of unassigned tasks. The lower bound LC_2 is not used in the beam search.

6. Computational results

In the literature on the RALBP only one set of 32 instances is used Gao et al. (2009). It uses eight precedence graphs from the literature of SALBP-1 (Roszieg, Gunther, Hahn, Tonge, Lutz3, Arc111, Barthol2 and Scholl) and four instances for each of the graphs. These four instances were generated using increasing WEST ratios Dar-El (1973), varying from 2 to 15. The WEST ratio defines the average

number of tasks per station. In all cases the number of stations is considered to be equal to the number of robots. Each task time p_{tr} is defined based on three values chosen uniformly at random: the difficulty d_t of performing task t, the efficiency factor e_r of robot r, and the specialization ability c_{tr} of robot r to perform task t. Given these three factors, the task times are defined as $p_{tr} = d_t/(e_t c_{tr})$.

This instance set (instance set 1) is limited for a number of reasons. First, the number of robots and stations is equal, so it is not possible to study their influence on the methods separately. Second, only one graph is used for each number of tasks. Therefore, the influence of different graph structures on the algorithms can not be evaluated.

We propose a second set of instances (instance set 2) that adapts the SALBP instance set of Otto et al. (2013) for the RALBP. The set considers three graph structures: chain graphs ("CH"), bottleneck graphs ("BN") and mixed graphs ("MX"). In a chain graph, at least 40% of the tasks are part of a chain, i.e. tasks that have at most one predecessor and one successor. A graph is a bottleneck graph if it has at least one bottleneck task. A bottleneck task has at least b tasks that precede it and have no other successors, and at least b successors that have no other predecessors. For the instances with 50 and 100 tasks explored here, the number b is set to 4. In the mixed graphs, no limitations are imposed on the graph generation. Different order strengths (OS) ranging from 20% to 90% are also represented in these instances. The order strength represents the percentage of precedence relations of the instance in comparison to the maximum number of precedence relations $|A|/(|T|(|T|-1)/2)$. Different task time distributions are considered. To define the task times, three types of distributions are used Kilbridge & Wester (1961): a normal distribution with a peak at task times of a tenth of the cycle time (PB), a

Table 2: Parameters of instance set 2.

Parameter	Levels
Number of Tasks $\|T\|$	50, 100
Graph Types	CH, BN and MX
Order Strength (OS)	20%, 60% and 90% (only for type MX)
Task Times Distribution (Dist.)	PM, PB, BM
Types of robots $\|R\|$	$\|T\|/7$, $\|T\|/4$
Number of stations $\|S\|$	$\|R\|/2$, $\|R\|$, $2\|R\|$
Task time variability *var*	$[p_t, 2p_t]$, $[p_t, 5p_t]$

normal distribution with a peak at half the cycle time (PM) and a bimodal distribution with peaks at a tenth of the cycle time and half the cycle time (BM), where the peak at a tenth of the cycle time is larger. Finally, the set contains instances with 20, 50, 100, and 1000 tasks but here we only consider instances with 50 and 100 tasks. This instance set is summarized in Table 2.

For each combination of the parameters above, Otto et al. (2013) produce 30 instances. To adapt the instances for the RALBP we select 5 of them, and generate RALBP-2 instances with two different numbers of types of robots ($|R| \in (|T|/7, |T|/4)$), three different numbers of stations ($|S| \in (|R|/2, |R|, 2|R|)$), and two task time variabilities (*var*). The task time variability defines the range from which the task times p_{tr} are uniformly selected. Given the time of a task p_t from the corresponding SALBP instance, the range for a task time p_{tr} is either $[p_t, 2p_t]$ (low variability), or $[p_t, 5p_t]$ (high variability). Therefore, for each of the 210 instances selected of Otto et al. (2013), we generate twelve instances, totaling 2520 RALBP-2 instances to be used in our experiments.

Table 3: Comparison of MIP models M_1 Mukund Nilakantan et al. (2015) and M_2 on instance set 1.

		M_1		M_2	
|T|	|R|	Dev. (%)	Time (s)	Dev. (%)	Time (s)
25	3	**0.00**	12.30	**0.00**	0.02
	4	**0.00**	1,254.11	**0.00**	0.15
	6	11.86	3,600.00	**0.00**	5.19
	9	60.55	3,600.00	**0.00**	604.40
35	4	2.93	3,600.00	**0.00**	0.41
	5	23.10	3,600.00	**0.00**	3.56
	7	36.32	3,600.00	**0.00**	330.13
	12	186.02	3,600.00	**0.00**	3,600.00
53	5	41.87	3,600.00	**0.00**	0.57
	7	21.20	3,600.00	**0.00**	30.48
	10	318.23	3,600.00	**0.00**	3,600.00
	14	561.19	3,600.00	**2.24**	3,600.00
70	7	142.27	3,600.00	**0.00**	3,600.00
	10	193.97	3,600.00	**3.45**	3,600.00
	14	332.35	3,600.00	**8.24**	3,600.00
	19	949.17	3,600.00	**9.17**	3,600.00
89	8	92.13	3,600.00	**1.85**	3,600.00
	12	462.12	3,600.00	**5.12**	3,600.00
	16	822.93	3,600.00	**3.90**	3,600.00
	21	1,021.94	3,600.00	**9.68**	3,600.00
111	9	291.85	3,600.00	**3.86**	3,600.00
	13	420.22	3,600.00	**5.88**	3,600.00
	17	809.52	3,600.00	**10.00**	3,600.00
	22	755.26	3,600.00	**15.79**	3,600.00
148	10	488.19	3,600.00	**0.00**	3,600.00
	14	721.59	3,600.00	**5.40**	3,600.00
	21	1,388.79	3,600.00	**9.42**	3,600.00
	29	1,748.68	3,600.00	**20.39**	3,600.00
297	19	611.62	3,600.00	**4.57**	3,600.00
	29	2,348.19	3,600.00	**29.22**	3,600.00
	38	1,719.03	3,600.00	**40.08**	3,600.00
	50	2,701.62	3,600.00	**48.65**	3,600.00
Avg.		**602.67**	**3414.60**	**7.41**	**2617.97**

6.1. MIP Models

We first compare our model M_2 to the model M_1 by Mukund Nilakantan et al. (2015) on instance set 1. The models were solved with CPLEX 12.5.0 using a single thread on a PC with a 3.66 GHz AMD FX-8150 processor with 32 GB of memory, and a time limit of one hour. The results are presented in Table 3.

In the table, column "Dev." shows the relative deviation $(C - C^*)/C^*$ of the cycle time C found by the model compared to the best known value C^* for each instance, and column "Time" presents the time in seconds needed to solve it. The table shows that model M_2 consistently produces smaller cycle times than model M_1 and it also proves more solutions to be optimal. Nine instances are proven optimal by model M_2 compared to only two instances by model M_1. The perfor-

mance of model M_2 is strongly influenced by the number of robots. Instances with more robots take much longer to be solved and when they are not optimally solved the relative deviations are larger than those for instances with fewer robots.

6.2. *Iterative Beam Search*

The iterative beam search has only two parameters, the beam width b_w for all the levels, and the time h, in seconds, for each of the iterations of the binary search. We have considered values of $b_w \in \{5, 10, 20, 40\}$ and $h \in \{5, 10, 20, 40\}$. For longer search times h and greater beam widths b_w, the results are better, but with a higher computational cost. Parameter h has the strongest influence on the results. The pairs of parameters with the longest search time h produce the smallest average relative deviation. The best results are found with $h = 40$ and $b_w = 40$ and are achieved in less than 15 minutes for every instance from both instance sets.

We compare the method with this set of parameters against the two heuristic methods available in the literature: the Particle Swarm Optimization (PSO) and the hybrid Cuckoo Search with Particle Swarm Optimization (CS-PSO), both proposed by Mukund Nilakantan et al. (2015). Their article presents results for 10 replications of the method but all the replications have the exact same result. The iterative beam search is deterministic and, therefore, for the same instance, always produces the same result. Thus, we only present results for one replication of the methods in Table 4.

In Table 4 we present the relative deviation from the best known cycle time $(C - C^*)/C^*$ (column "Dev.") and the running time for each of the methods (column "t"). PSO and CS-PSO were run on a PC with a 2.30 GHz Core i5 processor, while our method was run on a PC with an AMD FX-8150 processor, running

Table 4: Comparison of the Iterative Beam Search (IBS) with the PSO and CS-PSO by Mukund Nilakantan et al. (2015)

		PSO Mukund Nilakantan et al. (2015)		CS-PSO Mukund Nilakantan et al. (2015)		IBS	
|T|	|R|	Dev. (%)	t (s)	Dev. (%)	t (s)	Dev. (%)	t (s)
25	3	**0.00**	2.65	**0.00**	3.60	**0.00**	0.11
	4	12.37	2.90	12.37	3.90	**0.00**	0.05
	6	7.22	3.00	3.09	4.20	**0.00**	0.07
	9	4.59	3.25	0.92	4.50	**0.00**	0.04
35	4	0.88	4.90	**0.00**	5.20	**0.00**	0.06
	5	2.13	5.40	0.91	6.30	**0.00**	0.12
	7	6.47	6.90	4.98	6.90	**0.00**	0.10
	12	12.90	8.50	10.75	8.90	**0.00**	0.07
53	5	1.11	13.10	**0.00**	13.50	**0.00**	0.15
	7	6.36	14.90	3.89	16.80	**0.00**	0.15
	10	10.34	16.20	8.87	17.90	**0.00**	0.16
	14	8.96	19.90	5.97	20.00	**0.00**	0.14
70	7	11.08	29.00	10.82	32.90	**0.77**	2.12
	10	15.95	32.50	13.79	35.80	**0.43**	0.54
	14	17.65	39.10	14.12	43.30	**0.00**	0.35
	19	22.50	43.40	16.67	47.80	**0.83**	0.85
89	8	7.18	41.90	6.48	45.70	**0.93**	0.43
	12	21.16	50.40	9.22	51.60	**1.02**	0.57
	16	14.15	59.60	6.83	63.30	**0.00**	0.18
	21	13.55	75.30	9.68	80.50	**0.65**	0.70
111	9	12.88	82.30	12.23	85.50	**0.43**	27.69
	13	16.18	89.50	18.01	92.50	**1.10**	11.00
	17	20.95	98.50	14.29	107.40	**0.95**	5.83
	22	21.71	110.80	19.74	114.50	**1.32**	4.00
148	10	11.25	179.80	9.41	183.50	**1.48**	157.73
	14	19.32	205.50	19.03	207.90	**0.00**	46.87
	21	24.22	215.90	22.42	219.50	**0.90**	19.60
	29	25.00	230.30	24.34	242.20	**1.32**	20.24
297	19	15.81	891.80	13.14	1,118.30	**0.00**	136.78
	29	19.58	997.60	18.67	1,331.30	**0.00**	125.59
	38	19.43	1,269.90	23.48	1,593.50	**0.00**	93.62
	50	32.43	1,390.80	19.46	1,664.30	**0.00**	73.43
Avg.		13.60	194.86	11.05	233.53	**0.38**	**22.84**

at 3.60 GHz. Both processors are comparable according to the Passmark benchmarks (Passmark Software Pty. Ltd., 2017). The average time of our method (22.84s) is much smaller than the times of both methods of Mukund Nilakantan et al. (2015) (194.86s for the PSO and 233.53s for the CS-PSO). Our method is faster than the methods in the literature for every instance but also produces the best results when compared to the best known values. The average relative deviation from the best known values is 0.38%, against 13.60% and 11.05% of the PSO and CS-PSO, respectively, and in every instance, the relative deviation produced by our method is smaller or equal to that of either PSO or CS-PSO.

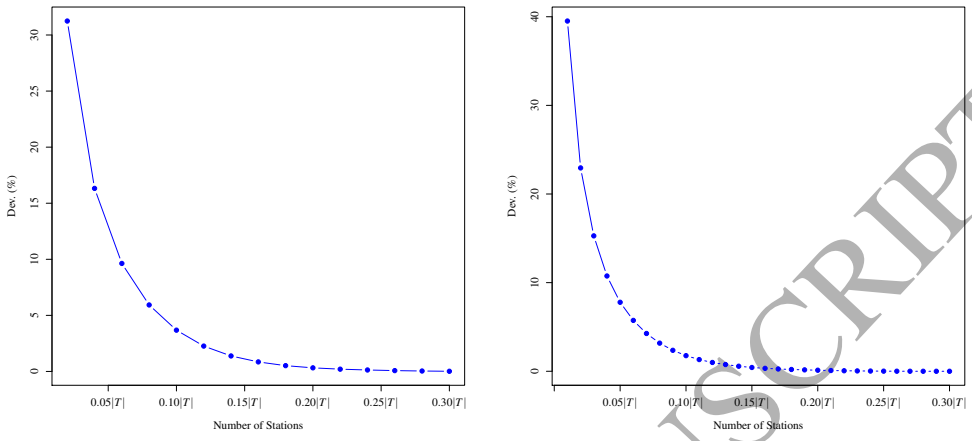

Figure 3: Comparison of the two lower bounds. The figures show the relative deviation between LC_2 and LC_1 as a function of the number of stations. On the left are the results for $n = 50$ tasks, on the right for $n = 100$ tasks.

6.3. Lower Bounds

To evaluate the lower bounds we need to consider multiple numbers of stations. We analyze $|S|$ from 1 to $0.3|T|$ for each of the instances in set 2. In Figure 3, we present the relative deviations between lower bounds LC_2 and LC_1 for 50 and 100 tasks.

In this figure, each point represents the average relative deviation $(LC_2 - LC_1)/LC_1$ of lower bound LC_2 from LC_1 for a given number of stations. The graphs show that the difference between the two lower bounds decreases for an increasing number of stations. The lower bound LC_2 has better results when multiple tasks need to be assigned to the same station for some of the chains. Therefore, LC_2 is better if there are much fewer stations than tasks. Because of this, we have studied the application of LC_2 only in cases where the number of stations is smaller than a fraction f of the number of tasks ($|S| \leq f|T|$).

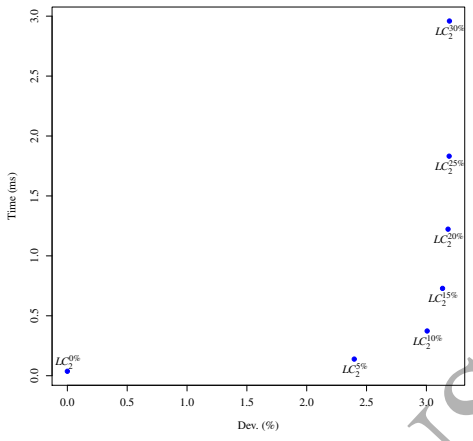

Figure 4: Results of the lower bound LC_2^f for varying values of f

Figure 4 compares the quality of the lower bound compared to the time to compute it for varying values of f from 0% to 30%. The quality is defined, as above, as the relative deviation from lower bound LC_1 on instance set 2. Both the relative deviation and the time increase with f. However, for $f \geq 20\%$, the time increases significantly with almost no improvements of the relative deviation. The time needed to compute lower bounds $LC_2^{0\%}$ and $LC_2^{5\%}$ is very similar, but their results are significantly different. The relative deviation improves to 3.2% using $f = 20\%$ and this result is obtained in 1.22 milliseconds in average. After that, the quality does not improve significantly and thus we select $f = 20\%$ for our experiments.

6.4. Branch-Bound-and-Remember

For the BBR method we evaluate two variants: the first uses the lower bound LC_1 in each of the nodes of the branch-and-bound algorithm and the second uses

Table 5: Comparison of the results of the exact solution of Model M_2 and the BBR methods.

		Model M_2		BBR method with LC_1		BBR method with $LC_2^{20\%}$	
|T|	|W|	Dev. (%)	t (s)	Dev. (%)	t (s)	Dev. (%)	t (s)
25	3	**0.00**	**0.02**	**0.00**	0.72	**0.00**	0.46
	4	**0.00**	**0.15**	**0.00**	0.59	**0.00**	0.42
	6	**0.00**	5.19	**0.00**	0.57	**0.00**	**0.42**
	9	**0.00**	604.40	**0.00**	0.64	**0.00**	**0.42**
35	4	**0.00**	**0.41**	**0.00**	0.72	**0.00**	0.65
	5	**0.00**	3.56	**0.00**	0.64	**0.00**	**0.60**
	7	**0.00**	330.13	**0.00**	0.69	**0.00**	**0.62**
	12	**0.00**	3,600.00	**0.00**	0.66	**0.00**	**0.60**
53	5	**0.00**	**0.57**	**0.00**	0.78	**0.00**	0.64
	7	**0.00**	30.48	**0.00**	0.86	**0.00**	**0.71**
	10	**0.00**	3,600.00	**0.00**	0.80	**0.00**	**0.72**
	14	2.24	3,600.00	**0.00**	0.92	**0.00**	**0.77**
70	7	**0.00**	3,600.00	**0.00**	170.14	**0.00**	**50.05**
	10	3.45	3,600.00	**0.00**	10.72	**0.00**	**9.00**
	14	8.24	3,600.00	**0.00**	**6.86**	**0.00**	9.50
	19	9.17	3,600.00	**0.00**	**13.41**	**0.00**	17.49
89	8	1.85	3,600.00	**0.00**	**12.52**	**0.00**	18.54
	12	5.12	3,600.00	**0.00**	**11.06**	**0.00**	17.15
	16	3.90	3,600.00	**0.00**	**11.30**	**0.00**	22.89
	21	9.68	3,600.00	**0.00**	**9.65**	**0.00**	13.56
111	9	4.09	**3,600.00**	**0.00**	**3,600.00**	0.86	**3,600.00**
	13	5.88	**3,600.00**	**0.00**	**3,600.00**	1.10	**3,600.00**
	17	10.00	**3,600.00**	**0.00**	**3,600.00**	0.48	**3,600.00**
	22	15.79	**3,600.00**	**0.00**	**3,600.00**	1.32	**3,600.00**
148	10	**0.00**	**3,600.00**	1.66	**3,600.00**	1.66	**3,600.00**
	14	5.40	**3,600.00**	**0.28**	**3,600.00**	**0.28**	**3,600.00**
	21	9.42	**3,600.00**	**0.90**	**3,600.00**	1.35	**3,600.00**
	29	20.39	**3,600.00**	**0.00**	**3,600.00**	1.97	**3,600.00**
297	19	4.57	**3,600.00**	**0.19**	**3,600.00**	**0.19**	**3,600.00**
	29	29.22	**3,600.00**	**0.30**	**3,600.00**	**0.30**	**3,600.00**
	38	40.08	**3,600.00**	**0.00**	**3,600.00**	0.40	**3,600.00**
	50	48.65	**3,600.00**	**0.54**	**3,600.00**	**0.54**	**3,600.00**
Avg.		**7.41**	**2617.97**	**0.12**	**1357.94**	**0.33**	**1355.16**

$LC_2^{20\%}$. Both methods were run on a PC with a 3.66 GHz AMD FX-8150 processor with 32 GB of memory, using one thread per execution. We use the IBS method to produce an initial solution for the BBR algorithm. We have tested the heuristic with different values of b_w and h. The method with $b_w = 40$ and $h = 40$ has the best results but can take up to 15 minutes without improving much the results compared to setting $b_w = 20$ and $h = 20$, which takes up to 6 minutes and half the average time of the previous parameter set. Therefore, we set $b_w = 20$ and $h = 20$ for finding the initial solutions. The time limit for each run was one hour. The memory usage of none of the runs did exceed 28 GB of main memory.

We first compare both BBR variants to model M_2 solved by CPLEX in the same computational environment using the same time limit on instance set 1. The results are presented in the Table 5. It reports the relative deviation $(C - C^*)/C^*$

of the best cycle time C from the best known value C^* (columns "Dev."), the total running time in seconds (columns "t") for each instance. We can see that both BBR methods solve all instances with up to 89 tasks and prove them to be optimal in less than three minutes. Only in some cases with a small number of robots the solution of M_2 takes less time than the BBR algorithm and in the case of M_2 the time grows exponentially with the number of robots. In average, the BBR is faster than model M_2 solved by CPLEX and the relative deviation found by the BBR method with LC_1 is better than the relative deviation found by the method with $LC_2^{20\%}$. This can be explained by the longer time to compute lower bound $LC_2^{20\%}$. For cycle times where a valid solution is found the number of explored nodes of both BBR methods is very similar. The largest difference in number of nodes between the two methods occurs in the last step, where the entire branch-and-bound tree must be explored. The method with LC_1 is faster until the last cycle time being tested, but is slower than the method with $LC_2^{20\%}$ in this last step, and therefore, it takes more time to prove a solution to be optimal. In instances where both methods prove a solution to be optimal, the method using $LC_2^{20\%}$ is, in average, about 35% faster than the method using LC_1.

The heuristic and the BBR methods presented were also executed for all the instances of instance set 2. The heuristic was configured with the same parameters used for the small instances. To present the results we divide the instance parameters in two groups: the parameters derived from Otto et al. (2013), which relate to the tasks (task times distribution and precedence graph), and the parameters related to robots and workstations.

The results related to the first set of parameters are presented in Table 6. It shows for each set of parameters with a fixed number of tasks ($|T|$), graph type

Table 6: Comparison of the BBR method and the IBS on instance set 2.

				IBS			BBR method with LC_1				BBR method with $LC_2^{20\%}$			
|T|	**Gr.**	**OS**	**Dist.**	**Gap (%)**	**Dev. (%)**	**t (s)**	**Prov. (%)**	**Gap (%)**	**Dev. (%)**	**t (s)**	**Prov. (%)**	**Gap (%)**	**Dev. (%)**	**t (s)**
50	BN	2	bimodal	7.21	1.08	**5.38**	66.67	**5.98**	**0.00**	1,748.30	**68.33**	5.99	0.01	1,701.29
			bottom	9.02	1.03	**8.18**	55.00	**7.83**	**0.00**	2,047.44	**60.00**	7.84	0.01	2,016.44
			middle	13.20	1.14	**5.28**	58.33	**11.88**	**0.00**	2,035.93	**61.67**	**11.88**	**0.00**	1,947.29
		6	bimodal	0.54	0.54	**0.36**	**100.00**	**0.00**	**0.00**	5.01	**100.00**	**0.00**	**0.00**	5.38
			bottom	0.38	0.38	**0.54**	**100.00**	**0.00**	**0.00**	8.62	**100.00**	**0.00**	**0.00**	8.61
			middle	0.58	0.58	**0.60**	**100.00**	**0.00**	**0.00**	7.11	**100.00**	**0.00**	**0.00**	7.59
	CH	2	bimodal	1.47	0.95	**2.19**	88.33	**0.51**	**0.00**	887.51	**98.33**	**0.51**	**0.00**	537.31
			bottom	2.35	0.96	**2.31**	88.33	**1.36**	**0.00**	1,034.52	**93.33**	**1.36**	**0.00**	738.63
			middle	3.62	1.01	**2.59**	75.00	**2.59**	**0.00**	1,381.73	**88.33**	**2.59**	**0.00**	1,202.95
		6	bimodal	0.63	0.63	**0.21**	**100.00**	**0.00**	**0.00**	2.86	**100.00**	**0.00**	**0.00**	2.95
			bottom	0.77	0.77	**0.24**	**100.00**	**0.00**	**0.00**	4.04	**100.00**	**0.00**	**0.00**	3.92
			middle	0.24	0.24	**0.42**	**100.00**	**0.00**	**0.00**	2.91	**100.00**	**0.00**	**0.00**	3.05
	MX	2	bimodal	9.05	1.00	**4.78**	61.67	**7.96**	**0.00**	1,734.77	**66.67**	7.97	0.00	1,668.45
			bottom	13.91	0.97	**8.51**	46.67	**12.78**	**0.00**	2,321.75	**48.33**	12.79	0.01	2,307.88
			middle	15.97	1.36	**7.05**	53.33	**14.34**	**0.00**	2,118.09	**56.67**	14.35	0.01	2,065.57
		6	bimodal	0.48	0.48	**0.30**	**100.00**	**0.00**	**0.00**	3.47	**100.00**	**0.00**	**0.00**	3.44
			bottom	0.40	0.40	**0.34**	**100.00**	**0.00**	**0.00**	3.07	**100.00**	**0.00**	**0.00**	2.88
			middle	0.34	0.34	**0.56**	**100.00**	**0.00**	**0.00**	3.57	**100.00**	**0.00**	**0.00**	3.36
		9	bimodal	0.68	0.68	**0.07**	**100.00**	**0.00**	**0.00**	1.85	**100.00**	**0.00**	**0.00**	1.84
			bottom	0.40	0.40	**0.07**	**100.00**	**0.00**	**0.00**	1.66	**100.00**	**0.00**	**0.00**	1.63
			middle	0.09	0.09	**0.12**	**100.00**	**0.00**	**0.00**	1.57	**100.00**	**0.00**	**0.00**	1.57
Avg.				**3.87**	**0.72**	**2.39**	**85.40**	**3.11**	**0.00**	**731.23**	**87.70**	**3.11**	**0.00**	**677.72**
100	BN	2	bimodal	22.62	0.81	**55.85**	**18.33**	**21.55**	**0.00**	2,940.15	**18.33**	21.93	0.32	2,940.15
			bottom	29.07	0.89	**40.22**	**16.67**	**27.84**	**0.00**	3,013.22	**16.67**	28.21	0.31	3,013.18
			middle	57.41	1.24	**44.91**	**1.67**	**55.42**	**0.00**	3,540.01	**1.67**	55.95	0.43	3,540.01
		6	bimodal	28.52	1.12	**21.79**	**18.33**	**27.04**	**0.00**	2,940.14	**18.33**	27.42	0.29	2,940.14
			bottom	29.18	1.09	**14.40**	**18.33**	**27.70**	**0.00**	2,940.15	**18.33**	28.14	0.34	2,940.14
			middle	64.51	1.40	**17.74**	**0.00**	**62.16**	**0.00**	3,600.00	**0.00**	62.56	0.30	3,600.00
	CH	2	bimodal	24.73	0.96	**38.75**	**16.67**	**23.48**	**0.00**	3,000.14	**16.67**	24.05	0.46	3,000.14
			bottom	22.43	1.00	**29.61**	**21.67**	**21.15**	**0.00**	2,820.18	**21.67**	21.72	0.47	2,820.18
			middle	59.17	1.22	**35.92**	**1.67**	**57.26**	**0.01**	3,540.06	**1.67**	57.80	0.43	3,540.06
		6	bimodal	18.65	0.85	**5.30**	46.67	**17.56**	**0.00**	2,228.71	**48.33**	17.62	0.04	2,194.82
			bottom	7.94	0.80	**4.76**	**76.67**	**7.02**	**0.00**	1,429.44	**76.67**	**7.02**	**0.00**	1,465.26
			middle	9.81	0.87	**4.61**	85.00	**8.86**	**0.00**	1,173.73	**86.67**	**8.86**	**0.00**	1,257.92
	MX	2	bimodal	23.28	0.77	**50.05**	**16.67**	**22.41**	**0.08**	3,000.14	**16.67**	22.68	0.33	3,000.14
			bottom	26.47	1.02	**38.90**	**18.33**	**25.14**	**0.01**	2,940.21	**18.33**	25.69	0.46	2,940.23
			middle	55.84	1.03	**46.92**	**0.00**	**54.22**	**0.00**	3,600.00	**0.00**	54.68	0.38	3,600.00
		6	bimodal	17.83	0.93	**8.84**	**45.00**	**16.67**	**0.00**	2,470.88	**45.00**	16.67	0.00	2,481.00
			bottom	16.99	0.82	**7.53**	**46.67**	**15.92**	**0.00**	2,356.43	**46.67**	15.93	0.01	2,356.28
			middle	36.96	1.06	**9.73**	**30.00**	**35.47**	**0.00**	2,926.36	**30.00**	35.49	0.01	2,934.46
		9	bimodal	0.42	0.42	**0.36**	**100.00**	**0.00**	**0.00**	3.19	**100.00**	**0.00**	**0.00**	3.23
			bottom	0.34	0.34	**0.34**	**100.00**	**0.00**	**0.00**	2.70	**100.00**	**0.00**	**0.00**	2.68
			middle	0.16	0.16	**0.55**	**100.00**	**0.00**	**0.00**	3.34	**100.00**	**0.00**	**0.00**	3.46
Avg.				**26.30**	**0.90**	**22.72**	**37.06**	**25.09**	**0.01**	**2403.294**	**37.22**	**25.35**	**0.22**	**2408.261**
Overall Results				**15.09**	**0.81**	**12.55**	**61.23**	**14.10**	**0.00**	**1567.26**	**62.46**	**14.23**	**0.11**	**1542.99**

(Gr.), order strength (OS), and task time distribution (Dist.), the results for the heuristic and the BBR method. In both tables in this section, we present the average time in seconds ("t") needed to solve all the instances with the given parameters, the average relative deviation $(C - LC^*)/LC^*$ of the current result from the best known lower bound LC^* for the instance (column "Gap"), the average relative deviation $(C - C^*)/C^*$ of the current result from the best known value C^* (column "Dev."), as well as the percentage of instances proven to be optimal ("*Prov.*").

First, it is possible to observe that both BBR methods consistently produce smaller relative deviations (and thus gaps) than the IBS, but need two orders of magnitude more time in average to find these results. In only 18 of the instances, the relative deviation of the IBS is better than that of one of the BBR methods, and in 1639 instances the result produced by the BBR algorithm is better than the result of the IBS. It is possible to observe the high influence of the order strength on the quality of the results. Since the number of nodes to be explored in a full branch-and-bound algorithm is larger for instances with low order strength (see Section 4.2), the nodes visited by the IBS are a smaller fraction of the complete space of solutions for the low order strength instances, and therefore are less probable to lead to optimal solutions. The type of the graph also influences the time needed to solve an instance but does not affect significantly the relative deviation found. In particular, chain graphs are the fastest to solve.

The BBR algorithm finds a provably optimal solution for 87.78% of the instances with 50 tasks, including all instances with order strength 60% or higher. Overall, 62.50% of the 2520 instances are solved and proven to be optimal, including all instances with order strength 90%, independently of the number of tasks.

The instances with chain precedence graphs are significantly faster to solve than the mixed graphs, which in turn are significantly faster to solve than the bottleneck graphs. That corroborates the time complexity presented in Section 4, since the chain graphs are composed of a few long chains, while a bottleneck task forces the decomposition in multiple small chains. Also, the time needed to solve instances with the "peak in the middle" distribution is significantly larger than the time needed for the other distributions. That happens because the task times for the "peak in the middle" distribution are larger than the task times in the other two distributions, and consequently the difference between the initial upper bound and the final result is a much greater than in the other two distributions, which leads to more iterations needed to achieve the final result.

As for instance set 1, the BBR method with $LC_2^{20\%}$ solves more instances than the version with LC_1, and in average in less time, but the relative deviations found by the former in the cases that are not proved to be optimal, are significantly worse than those found with LC_1. Despite this, in all the instances proved to be optimal by both methods, the number of nodes visited and the time to prove optimality by the solution using $LC_2^{20\%}$ are in average 6.10% and 7.55% smaller than the number of nodes visited by the solution using LC_1. This means that for instances with 50 and 100 tasks the cost of executing the $LC_2^{20\%}$ method is more significant for the result than the reduction of nodes caused by it. The largest difference in the relative deviation is found for instances with 100 tasks, especially in bottleneck graphs, because the division in chains of a *CH* graph creates larger chains than those produced in *BN* graphs, and the family of bounds LC_2 has better results for instances with large chains. This finding is corroborated by the fact that the instances with graphs *CH* and *MX* where $LC_2^{20\%}$ improves the most have the

Table 7: Comparison of the BBR method and the IBS on instance set 2, aggregated by the RALBP-specific instance parameters.

				IBS			BBR method with LC_1				BBR method with $LC_2^{20\%}$			
|T|	**|R|**	**|S|**	**Var.**	**Gap (%)**	**Dev. (%)**	**t (s)**	**Prov. (%)**	**Gap (%)**	**Dev. (%)**	**t (s)**	**Prov. (%)**	**Gap (%)**	**Dev. (%)**	**t (s)**
50	7	3	2	3.14	0.29	**12.91**	0.70	**2.84**	**0.00**	1,304.43	**0.76**	2.84	0.00	1,160.82
			2	1.31	0.46	**0.66**	0.87	**0.84**	**0.00**	811.35	**0.88**	0.84	0.00	770.05
		14	2	0.38	0.38	**0.50**	1.00	**0.00**	**0.00**	154.79	**1.00**	**0.00**	**0.00**	156.18
			5	9.85	0.99	**9.43**	**0.63**	**8.70**	**0.00**	1,436.40	**0.72**	**8.71**	**0.01**	1,236.77
		7	5	3.82	1.50	**0.62**	**0.84**	**2.25**	**0.00**	1,041.18	**0.87**	**2.25**	**0.00**	906.63
			5	0.95	0.95	**0.50**	**1.00**	**0.00**	**0.00**	113.86	**1.00**	**0.00**	**0.00**	109.92
	12	6	2	3.31	0.48	**0.99**	0.69	**2.81**	**0.00**	1,301.88	**0.71**	**2.81**	**0.00**	1,232.99
			2	5.33	0.26	**0.54**	0.96	**5.04**	**0.00**	472.85	**0.96**	**5.04**	**0.00**	471.90
		24	2	0.93	0.16	**0.40**	0.98	**0.75**	**0.00**	124.18	**0.98**	**0.75**	**0.00**	124.63
			5	10.12	1.62	**1.03**	**0.65**	**8.26**	**0.00**	1,390.23	**0.70**	**8.27**	**0.01**	1,327.23
		12	5	7.07	1.21	**0.60**	**0.94**	**5.77**	**0.00**	573.30	**0.93**	**5.78**	**0.00**	586.60
			5	0.28	0.28	**0.43**	**1.00**	**0.00**	**0.00**	50.28	**1.00**	**0.00**	**0.00**	48.89
Avg.				**3.87**	**0.72**	**2.39**	**85.40**	**3.11**	**0.00**	**731.23**	**87.70**	**3.11**	**0.00**	**677.72**
100	14	7	2	10.98	0.34	**80.26**	0.21	**10.61**	**0.01**	2,938.47	**0.21**	10.72	0.10	2,928.35
			2	6.06	0.58	**12.43**	0.28	**5.44**	**0.00**	2,784.22	**0.28**	5.75	0.29	2,800.56
		28	2	42.13	0.46	**4.28**	0.40	**41.42**	**0.00**	2,357.59	**0.40**	41.42	0.00	2,359.94
			5	36.12	1.01	**99.35**	**0.19**	**34.79**	**0.05**	3,016.83	**0.21**	**35.23**	**0.36**	2,988.83
		14	5	20.81	2.08	**9.51**	**0.24**	**18.27**	**0.00**	2,883.94	**0.24**	**19.24**	**0.80**	2,905.63
			5	30.60	1.39	**3.65**	**0.43**	**28.71**	**0.00**	2,333.32	**0.43**	**28.70**	**0.00**	2,337.06
	25	12	2	8.32	0.40	**25.59**	**0.22**	**7.88**	**0.00**	2,912.32	**0.22**	**8.11**	**0.21**	2,922.99
			2	59.53	0.46	**5.26**	**0.36**	**58.72**	**0.00**	2,539.47	**0.36**	**58.72**	**0.00**	2,540.45
		50	2	6.64	0.25	**2.04**	**0.79**	**6.33**	**0.00**	783.29	**0.79**	**6.33**	**0.00**	783.95
			5	29.48	1.76	**22.91**	**0.21**	**27.16**	**0.00**	2,916.75	**0.21**	28.25	0.83	2,935.19
		25	5	60.98	1.63	**5.33**	**0.30**	**58.22**	**0.00**	2,718.05	**0.30**	58.24	0.02	2,740.34
			5	3.97	0.40	**2.02**	**0.83**	**3.52**	**0.00**	655.28	**0.83**	3.52	0.00	655.84
Avg.				**26.30**	**0.90**	**22.72**	**37.06**	**25.09**	**0.01**	**2403.294**	**37.22**	**25.35**	**0.22**	**2408.261**
Overall Results				**15.09**	**0.81**	**12.55**	**61.23**	**14.10**	**0.00**	**1567.26**	**62.46**	**14.23**	**0.11**	**1542.99**

highest order strength.

The results for the new parameters introduced for RALBP, the number of stations and robots, as well as the task time variability are presented in Table 7. In this table, the parameters considered are the number of tasks "$|T|$", the number of robots "$|R|$", and the number of stations "$|S|$", as well as the task time variability "Var". While we can not observe an influence of the task time variability on the time needed by the BBR algorithm to optimize the problem, both the parameters $|W|$ and $|S|$ influence the result. The number of nodes given by bound (9) does not depend on the number of robots, for the instances proven to be optimal. The running time, however, depends on the number of robots r because in each node the method has to select the best robot to perform the current station in a time linear in r.

The number of stations in the worst case complexity calculated in Section 4.2 impacts directly the time of the algorithm. However we can see that instances with more stations are significantly faster to solve than instances with fewer stations and the number of solutions provably optimal also increases with the number of stations. The reason is that with more stations only a few tasks are assigned to each station, so less station loads are generated and, since the dominance rules and lower bounds are applied only to maximal station loads, much of the partial solutions are removed. It can also be observed that in general the difference between the lower bound and the upper bound for the cycle time is much smaller than the same difference in instances with a small number of stations. This difference substantially influences the time of the algorithm. In particular, the method with $LC_2^{20\%}$ is slower than the method with LC_1 for fewer stations and a larger cycle time. The time used in early iterations of the method increases the overall time by the BBR method with $LC_2^{20\%}$.

The BBR algorithm with $LC_2^{20\%}$ proves more solutions to be optimal than BBR algorithm with LC_1 because it solves more instances with 50 tasks and a low number of stations. This is expected because the family of bounds LC_2 is more efficient when there are fewer stations.

7. Conclusion

In this article a MIP model, a heuristic procedure and a branch-bound-and-remember method were presented for the Robotic Assembly Line Balancing Problem of type 2. The MIP model improves over previous models by using a better formulation of the precedence constraints and using a linear objective function instead of the quadratic objective function. The branch-bound-and-remember

method can solve all instances up to 89 tasks in instance set 1. When compared to our MIP model, the method performs well independently of the number of stations. The heuristic is shown to be efficient and able to, in less than 15 minutes, find results in average less than 1% above the best known values in the literature. Therefore, the heuristic is used to provide good initial upper bounds for the BBR method. Finally, after identifying that the instance set in the literature only consider a few characteristics of the problem, we propose a new set of instances and evaluate the algorithms on them. More than 60% of the new instances proposed with 50 and 100 tasks are solved by our branch-bound-and-remember method. The order strength and the number of tasks are the parameters that influence the results the most. The BBR method, for example, is able to solve all instances with very high order strength, as well as solving 87.78% of the instances with 50 tasks versus 37.22% of the instances with 100 tasks. Also, the number of stations and the number of robots are directly proportional to the number of instances provably optimal, but influence the results independently and need to be studied independently, which is made possible by the new instance set.

The experiments show that it is possible to obtain good results even with an exact method, and our exact methods have comparable results to the heuristics proposed in the literature. This opens an opportunity for further improvements of heuristics for the problem and the solution of larger instances. Also, identifying special cases where lower bounds and dominance rules are more efficient and applying these methods only in these cases may lead to future improvements of the proposed algorithms.

References

Baeten, J., Donné, K., Boedrij, S., Beckers, W., & Claesen, E. (2008). Autonomous Fruit Picking Machine: A Robotic Apple Harvester. In C. Laugier, & R. Siegwart (Eds.), *Field and Service Robotics SE - 51* (pp. 531–9). Springer Berlin Heidelberg volume 42 of *Springer Tracts in Advanced Robotics*. doi:10.1007/978-3-540-75404-6_51.

Battaïa, O., & Dolgui, A. (2013). A taxonomy of line balancing problems and their solutionapproaches. *Int. J. Prod. Econ.*, *142*, 259–77. doi:10.1016/j.ijpe.2012.10.020.

Baybars, I. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Manag. Sci.*, *32*, 909–32. doi:10.1287/mnsc.32.8.909.

Becker, C., & Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *Eur. J. Oper. Res.*, *168*, 694–715. doi:10.1016/j.ejor.2004.07.023.

Blum, C., & Miralles, C. (2011). On solving the assembly line worker assignment and balancing problem via beam search. *Comput. Oper. Res.*, *38*, 328–39.

Borba, L., & Ritt, M. (2014). A heuristic and a branch-and-bound algorithm for the Assembly Line Worker Assignment and Balancing Problem. *Comput. Oper. Res.*, *45*, 87–96. doi:10.1016/j.cor.2013.12.002.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *Eur. J. Oper. Res.*, *183*, 674–93. doi:10.1016/j.ejor.2006.10.010.

Boysen, N., Fliedner, M., & Scholl, A. (2008). Assembly line balancing: Which model to use when. *Int. J. Prod. Res.*, *111*, 509–28. doi:10.1016/j.ijpe.2007.02.026.

Chaves, A., Lorena, L., & Miralles, C. (2009). Hybrid Metaheuristic for the Assembly Line Worker Assignment and Balancing Problem. In *Hybrid Metaheuristics* (pp. 1–14). Springer Berlin / Heidelberg volume 5818 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-642-04918-7_1.

Chaves, A. A., Miralles, C., & Lorena, L. A. N. (2007). Clustering search approach for the assembly line worker assignment and balancing problem. In *Proc. 37th Int. Conf. Comput. Indust. Eng.* (pp. 1469–78). Alexandria, Egypt.

Dar-El, E. M. (1973). MALBA Heuristic Technique for Balancing Large Single-Model Assembly Lines. *A I I E Transactions*, *5*, 343–56. doi:10.1080/05695557308974922.

Dilworth, R. P. (1950). A decomposition theorem for partially ordered sets. *Ann. Math.*, *51*, 161–6. doi:10.2307/1969503.

Gao, J., Sun, L., Wang, L., & Gen, M. (2009). An efficient approach for type II robotic assembly line balancing problems. *Comput. Ind. Eng.*, *56*, 1065–80. doi:10.1016/j.cie.2008.09.027.

Henrich, D., & Wörn, H. (2000). *Robot Manipulation of Deformable Objects*. (1st ed.). London: Springer London. doi:10.1007/978-1-4471-0749-1.

Jackson, J. R. (1956). A Computing Procedure for a Line Balancing Problem. *Manag. Sci.*, *2*, 261–71. doi:10.1287/mnsc.2.3.261.

Kilbridge, M., & Wester, L. (1961). The Balance Delay Problem. *Manag. Sci.*, *8*, 69–84. doi:10.1287/mnsc.8.1.69.

Klein, R., & Scholl, A. (1996). Maximizing the Production Rate in Simple Assembly Line Balancing–A Branch and Bound Procedure. *Eur. J. Oper. Res.*, *91*, 367–85. doi:10.1016/0377-2217(95)00047-X.

Levitin, G., Rubinovitz, J., & Shnits, B. (2006). A genetic algorithm for robotic assembly line balancing. *Eur. J. Oper. Res.*, *168*, 811–25. doi:10.1016/j.ejor.2004.07.030.

Milberg, J., & Schmidt, M. (1990). Flexible Assembly Systems Opportunities and Challenge for Economic Production. *CIRP Annals - Manufacturing Technology*, *39*, 5–8. doi:10.1016/S0007-8506(07)60991-3.

Miralles, C., García-Sabater, J. P., Andrés, C., & Cardos, M. (2007). Advantages of assembly lines in Sheltered Work Centres for Disabled. A case study. *Int. J. Prod. Econ.*, *110*, 187–97. doi:10.1016/j.ijpe.2007.02.023.

Miralles, C., García-Sabater, J. P., Andrés, C., & Cardós, M. (2008). Branch and bound procedures for solving the Assembly Line Worker Assignment and Balancing Problem: Application to Sheltered Work centres for Disabled. *Discrete Appl. Math.*, *156*, 352–67. doi:10.1016/j.dam.2005.12.012.

Moreira, M., Ritt, M., Costa, A., & Chaves, A. (2012). Simple heuristics for the assembly line worker assignment and balancing problem. *J. Heuristics*, *18*, 505–24. doi:10.1007/s10732-012-9195-5.

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. *Eur. J. Oper. Res.*, *236*, 403–9. doi:10.1016/j.ejor.2013.11.033.

Mukund Nilakantan, J., Ponnambalam, S. G., Jawahar, N., & Kanagaraj, G. (2015). Bio-inspired search algorithms to solve robotic assembly line balancing problems. *Neural Comput. Appl.*, *26*, 1379–93. doi:10.1007/s00521-014-1811-x.

Mutlu, Ö., Polat, O., & Supciller, A. A. (2013). An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. *Comput. Oper. Res.*, *40*, 418–26. doi:10.1016/j.cor.2012.07.010.

Otto, A., Otto, C., & Scholl, A. (2013). Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *Eur. J. Oper. Res.*, *228*, 33–45. doi:10.1016/j.ejor.2012.12.029.

Passmark Software Pty. Ltd. (2017). Passmark benchmarks. URL: http://www.cpubenchmark.net http://www.cpubenchmark.net.

Pires, J., & Sá da Costa, J. (2000). Object-oriented and distributed approach for programming robotic manufacturing cells. *Robot. Comput.-Integr. Manuf.*, *16*, 29–42. doi:10.1016/S0736-5845(99)00039-3.

Polat, O., Kalayci, C. B., zcan Mutlu, & Gupta, S. M. (2016). A two-phase variable neighbourhood search algorithm for assembly line worker assignment and balancing problem type-II: an industrial case study. *Int. J. Prod. Res.*, *54*, 722–41. doi:10.1080/00207543.2015.1055344.

Purnell, G. (1998). Robotic equipment in the meat industry. *Meat Sci.*, *49*, S297–307. doi:10.1016/S0309-1740(98)90056-0.

Ritt, M., & Costa, A. M. (2015). Improved integer programming models for simple assembly line balancing and related problems. *Int. Trans. Oper. Res.*, *online*. doi:10.1111/itor.12206.

Rubinovitz, J., Bukchin, J., & Lenz, E. (1993). RALB - A Heuristic Algorithm for Design and Balancing of Robotic Assembly Lines. *CIRP Annals - Manufacturing Technology*, *42*, 497–500. doi:10.1016/S0007-8506(07)62494-9.

Rubinovitz, J., & Levitin, G. (1995). Genetic algorithm for assembly line balancing. *Int. J. Prod. Econ.*, *41*, 343–54. doi:10.1016/0925-5273(95)00059-3.

Salveson, M. E. (1955). The assembly line balancing problem. *J. Ind. Eng.*, *6*, 18–25.

Scholl, A., & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *Eur. J. Oper. Res.*, *168*, 666–93. doi:10.1016/j.ejor.2004.07.022.

Schrage, L., & Baker, K. R. (1978). Dynamic programming solution of sequencing problems with precedence constraints. *Oper. Res.*, *26*, 444–9.

Sewell, E. C., & Jacobson, S. H. (2011). A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem. *INFORMS J. Comput.*, *24*, 433–42. doi:10.1287/ijoc.1110.0462.

Vilà, M., & Pereira, J. (2013). An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time. *Eur. J. Oper. Res.*, *229*, 106–13. doi:10.1016/j.ejor.2013.03.003.

Vila, M., & Pereira, J. (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Comput. Oper. Res.*, *44*, 105–14. doi:10.1016/j.cor.2013.10.016.

Appendix A. Model M_1 by Mukund Nilakantan et al. (2015)

The model M_1, proposed by Mukund Nilakantan et al. (2015), is presented here:

$$M_1 = \textbf{minimize} \quad C, \tag{A.1}$$

$$\textbf{subject to} \quad \sum_{t \in A_s} p_{tr} x_{ts} y_{sr} \leq C, \qquad \forall s \in S, r \in R, \tag{A.2}$$

$$\sum_{r \in R} y_{sr} = 1, \qquad \forall s \in S, \tag{A.3}$$

$$\sum_{s \in I_t} x_{ts} = 1, \qquad \forall t \in T, \tag{A.4}$$

$$\sum_{s \in S} s x_{is} - \sum_{s \in S} s x_{js} \geq 0, \qquad \forall (i,j) \in A, \tag{A.5}$$

$$x_{ts} \in \{0,1\}, \qquad \forall s \in S, t \in A_s, \tag{A.6}$$

$$y_{sr} \in \{0,1\}, \qquad \forall s \in S, r \in R. \tag{A.7}$$

The objective function (A.1) and the constraints (A.3), (A.4), (A.6) and (A.7) are the same as the constraints of the model M_2 presented in Section 2. The cycle time is defined in (A.2), using a quadratic function that associates tasks and robots to stations. The other constraint that differs from model M_2 are the constraints (A.5). These constraints are dominated by constraints (5) as shown by Ritt & Costa (2015).