UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LEONARDO DE MIRANDA BORBA

Exact and Heuristic Methods for Heterogeneous Assembly Line Balancing Problems of Type 2

Ph.D. thesis

Advisor: Prof. Marcus Ritt
Coadvisor: Prof. Cristóbal Miralles

Porto Alegre
January 2018

CIP – CATALOGING-IN-PUBLICATION

Borba, Leonardo de Miranda

Exact and Heuristic Methods for Heterogeneous Assembly Line Balancing Problems of Type 2 / Leonardo de Miranda Borba. – Porto Alegre: PPGC da UFRGS, 2018.

98 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Advisor: Marcus Ritt; Coadvisor: Cristóbal Miralles.

1. Assembly Line Balancing. 2. ALWABP-2. 3. RALBP-2. 4. MMALBP-2. 5. Branch-and-Bound. 6. Beam Search. I. Ritt, Marcus. II. Miralles, Cristóbal. III. Título.

“To begin... To begin... How to start?
I’m hungry. I should get coffee. Coffee would help me think.
Maybe I should write something first, then reward myself with coffee.
Coffee and a muffin.
Okay, so I need to establish the themes.
Maybe a banana-nut. That’s a good muffin.”

—

Charlie Kaufman, in Adaptation. (2002)

AGRADECIMENTOS

Nos agradecimentos eu me reservo o direito de escrever em português. Pois foi em português que meus pais, Arlene e Eros, me educaram e é a eles que devo agradecer primeiro. Me deram todo o estímulo necessário, mesmo que isso por vezes não fosse o melhor pra eles. Me sentirei eternamente em débito com eles. Ter o direito a uma cama elástica para cair em qualquer salto errado é um privilégio de que tenho orgulho. E claro estendo este agradecimento familiar a minha irmã Aline, que é as vezes a única pessoa com quem posso me abrir, e a todos os meus parentes mais próximos, em especial Vó Alba, Vó Lete, Vô Luís, Déia, Tiloca, Babi e Fê. E perdão a todos os familiares pelos momentos de desespero e ansiedade.

Ainda no lado pessoal, quero agradecer aos meus amigos e colegas. Essa é especial para vocês, Zappa, Vitor, Biel, Ju, e Paula. Valeu por aturarem meus momentos de chatice e meus papos longos sobre cinema e esportes em geral na tentativa de fugir do estresse por algumas horas. Ainda aos amigos e colegas Tadeu (que me ajudou profundamente em duas ou três conversas), André, Mourinha, Fernando, Árton, Paulo, Friske, Victoria, Alex, Becker, Artur, João, Johnny agradeço pelo aprendizado. As conversas malucas me fizeram enxergar um mundo novo. E quero agradecer aos amigos e colegas recentes, das novas empreitadas, Bruno e Chico, bem como a todo o pessoal da Oxigênio Aceleradora e ao Lucas. Que me ajudaram a manter a sanidade enquanto eu trabalhava doze horas diárias e ainda tinha que continuar escrevendo minha tese.

Também tem um espaço de agradecimento em espanhol para os amigos do período de intercâmbio na Espanha: *Matteo, Luís, Gema, Francesco, me habéis hecho sonréir en un tiempo en que la felicidad me era muy difícil. Siempre recordaré las noches en el Bósforo.*

Como um último agradecimento do lado pessoal, um que é todo especial, gostaria de falar da Ingrid, minha namorada. Ela apareceu no melhor momento possível. Por vezes o ânimo que me faltava veio dela, em um abraço apertado. Agora, finalmente, poderemos ter um namoro normal. A ela peço perdão pelo começo turbulento e agradeço por entender o meu esforço. Amar me faz bem. Ser amado por ela me enche de felicidade.

Uma página de agradecimento me obrigaria a escrever sobre meu orientador e meu grupo de pesquisa. No meu caso isso não exige esforço. O Marcus é o pesquisador e orientador mais esforçado e competente que tive o prazer de conhecer. São incontáveis as vezes que falei dele com orgulho de ser seu orientando. A noite em Valência tomando cerveja e desenvolvendo o nosso algoritmo para o ALWABP é o momento que lembrarei com maior carinho da minha breve carreira acadêmica. Nas conversas sobre a tese e sobre outros assuntos, tive com o Marcus as maiores mudanças na minha forma de pensar. Por isso, hoje eu admiro a ciência, e a ciência é muito diferente do que eu imaginava antes do doutorado. É ótimo descobrir estar errado.

E obrigado aos grupos de pesquisa em que trabalhei, em Porto Alegre e em Valência. Obrigado especial para o Cristóbal, por escutar e me receber em Valência, e para a Luciana que

entendia o grupo de otimização como uma coisa só e nos recebia e tratava como a seus próprios orientandos.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

ALB	Assembly Line Balancing
ALWABP	Assembly Line Worker Assignment and Balancing Problem
SALBP	Simple Assembly Line Balancing Problem
UALWABP	U-Line Assembly Line Worker Assignment and Balancing Problem
PALWABP	Parallel Assembly Line Worker Assignment and Balancing Problem
MMALBP	Mixed-Model Assembly Line Balancing Problem
RALBP	Robotic Assembly Line Balancing Problem
IBS	Iterative Beam Search
IPBS	Iterative Probabilistic Beam Search
BFS	Best First Search
BB	Branch-and-Bound
BBR	Branch,Bound-and-Remember
CBFS	Cyclic Best First Search

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The difference among workstations is assumed to be negligible in traditional assembly lines. Heterogeneous assembly lines consider the problem of industries in which the task times vary according to some property to be selected for the task. In the Assembly Line Worker Assignment and Balancing Problem (ALWABP), workers are assigned to workstations and according to their abilities, they execute tasks in different amounts of time. In some cases they can even be incapable of executing some tasks. In the Robotic Assembly Line Balancing Problem (RALBP) there are different types of robots and each station must be executed by a robot. Multiple robots of the same type may be used.

We propose exact and heuristic methods for minimizing the cycle time of these two problems, for a fixed number of stations. The problems have similar characteristics that are explored to produce lower bounds, heuristic methods, mixed-integer programming models, and reduction and dominance rules. For the branching strategy of the branch-and-bound method, however, the differences among the problem force the use of two different algorithms. A task-oriented strategy has the best results for the ALWABP-2 while a station-oriented strategy has the best results for the RALBP-2. The lower bounds, heuristics, MIP models and branch-and-bound algorithms for these two problems are shown to be competitive with the state-of-the-art methods in the literature.

Keywords: Assembly Line Balancing. ALWABP-2. RALBP-2. MMALBP-2. Branch-and-Bound. Beam Search.

Métodos Exatos e Heurísticos para Problemas de Balanceamento de Linhas de Montagem Heterogêneas do Tipo 2

RESUMO

A diferença entre estações de trabalho é considerada desprezível em linhas de montagem tradicionais. Por outro lado, linhas de montagem heterogêneas consideram o problema de indústrias nas quais os tempos das tarefas variam de acordo com alguma característica a ser selecionada para a tarefa. No Problema de Balanceamento e Atribuição de Trabalhadores em Linhas de Montagem (do inglês Assembly Line Worker Assignment and Balancing Problem, ALWABP), os trabalhadores são responsáveis por estações de trabalho e de acordo com as suas habilidades, eles executam as tarefas em diferentes quantidades de tempo. Em alguns casos, os trabalhadores podem até ser incapazes de executar algumas tarefas. No Problema de Balanceamento de Linhas de Montagem Robóticas (do inglês Robotic Assembly Line Balancing Problem, RALBP), há diferentes tipos de robôs e o conjunto de tarefas de cada estação deve ser executada por um robô. Robôs do mesmo tipo podem ser usados múltiplas vezes. Nós propomos métodos exatos e heurísticos para a minimização do tempo de ciclo destes dois problemas, para um número fixo de estações. Os problemas têm características similares que são exploradas para produzir limitantes inferiores, métodos inferiores, models de programação inteira mista, e regras de redução e dominância. Para a estratégia de ramificação do método de branch-and-bound, entretanto, as diferenças entre os problemas forçam o uso de dois algoritmos diferentes. Uma estratégia orientada a tarefas tem os melhores resultados para o ALWABP-2, enquanto uma estratégia orientada a estações tem os melhores resultados para o RALBP-2. Nós mostramos que os limitantes inferiores, heurísticas, modelos de programação inteira mista e algoritmos de branch-and-bound para estes dois problemas são competitivos com os métodos do estado da arte da literatura.

Palavras-chave: Balanceamento de Linhas de Montagem, ALWABP-2, RALBP-2, Branch-and-Bound, Beam Search, MIP.

1 INTRODUCTION

In an assembly line, a set of workstations is defined, and the workers or machines at each workstation are responsible for executing product assembling tasks, while the product passes from one station to another. Assembly lines are widely applied, in automotive (CUNNINGHAM, 1980), aerospace (HEIKE et al., 2001), electronics (BALAKRISHNAN; VANDERBECK, 1993), and home appliances (PARK; PARK; KIM, 1997) industries, for instance. In this type of production of standardized products, it is easier to split the production into smaller tasks and select some specialized workers or machines to repeat them. In order to increase the production rate or decreasing the production cost, it is crucial to plan the configuration of the factory adequately (CUNNINGHAM, 1980). The assembly line balancing problems, therefore, consist in optimizing one or both of these two aspects while satisfying the constraints of an assembly line.

The assembly line balancing (ALB) problems were first formalized by Salveson (1955), who defined a simplified model for the ALB problems called the Simple Assembly Line Balancing Problem (SALBP) (BAYBARS, 1986). The simple assembly line is composed of a set of tasks T and a set of workstations S placed successively. Each task must be executed at a workstation. Also, the tasks are partially ordered. If a task t precedes another task t', then the task t must be performed before the task t', i.e. it must be assigned to a station before the station executing t' or to the same station. Each task needs a time p_t to be performed. For each station s, a station time (P_s) is also defined corresponding to the sum of the times of the tasks assigned to s. The cycle time C of the line, the time difference between finishing two products, will be greater than all station times for a sufficiently large number of units being produced, because every task has to be performed. Therefore the cycle time C equals the maximum station time over all stations.

Given these definitions, there are two main objectives in a SALBP: minimizing the number of stations or minimizing the cycle time. The most common forms of the problem minimize the number of stations for a fixed cycle time (SALBP-1) or minimize the cycle time for a fixed number of stations (SALBP-2). But we can also minimize the product of both variables (SALBP-E) or find a valid solution for a fixed number of stations and cycle time (SALBP-F).

Real assembly lines are more complex than the SALBP. Examples in the literature include, for instance, U-shaped (MILTENBURG; WIJINGAARD, 1994) and parallel (GÖKÇEN; AĞPAK; BENZER, 2006) assembly lines. In these problems the time to perform a task does not depend on the conditions in which the task is performed and is always the same. In general assembly lines, the same task can be performed by different workers, by different robots, or for different models of the product. In these cases according to the conditions in which the task is executed the time to perform it is different.

For the case with significantly distinct workers, Miralles et al. (2007) proposed the Assembly Line Worker Assignment and Balancing Problem (ALWABP), which was focused on as-

sembly lines in Sheltered Workcentres for Disabled (SWDs). SWDs are not-for-profit industries focused on giving professional training and a work opportunity for disabled people (CHAVES, 2009). Currently, there are more than 785 million persons with some kind of disability, including 110 million with severe disabilities, according to the World Health Organization (2011). They still lack conditions to pursue rights like education and employment. While the employment rate of disabled people has improved (Organisation for Economic Co-Operation and Development, 2003), the quality of the jobs is still low. Work quality impacts the lives of disabled persons, and is an important therapy for some of these workers (COSTA, 2001). The use of assembly lines in SWDs has been shown to be an important tool to achieve this goal (MIRALLES et al., 2007).

While the difference among workers is negligible in some conventional factories, this is not true for SWDs. As in the SALBP, we are given a set of workstations S and a set of tasks T, but there is also a set of workers W, with $|W| = |S|$. In the ALWABP each task t has an execution time p_{tw} according to the worker w executing it. Therefore, to solve the problem, it is necessary to assign a worker to each station and, besides that, balance the tasks among the stations. Since the main goal of SWDs is to maintain their workers active, and the increase in production rate brings funds for the growth of the work centres, then the objective of the problem is to maximize the production rate of the line, by minimizing its cycle time. Therefore we solve the type 2 of the ALWABP.

Robotic assembly lines are also a type of heterogeneous assembly line. Different types of robots take different times to execute the same task. But, on the contrary to the ALWABP, the number of robots of the same type is not limited. For instance, it is possible to use the same robot for all stations. A solution for the problem must correctly balance the line and assign a type of robot to each station following the same constraints as in the SALBP. This problem is called Robotic Assembly Line Balancing Problem (RALBP) (RUBINOVITZ; BUKCHIN; LENZ, 1993). As in the SALBP and ALWABP, types 1, 2, E and F of the RALBP can be studied. Here we work with the RALBP-2 version.

In this thesis we propose and evaluate novel methods, both exact and heuristic, for these two problems: the ALWABP-2 and the RALBP-2.

1.1 Definition of the problems

In the ALWABP, each worker $w \in W$ takes a different time p_{tw} to execute a task $t \in T$, according to his or her abilities. There is no relation between the times needed to execute a task by two workers. For instance, for a task t_1, a worker w_1 could take 30 seconds and a worker w_2 could take 15 seconds, while for a task t_2, w_1 could take 10 seconds and w_2 could take 20 seconds. In practice, a worker can even be unable to perform some tasks. In these cases, we set $p_{tw} = \infty$. Table 1.1 shows an example for six tasks and three workers.

Additionally, some tasks may depend on others to be executed. The order of the tasks is

Table 1.1 – Example of execution times p_tw for an ALWABP instance with 6 tasks and 3 workers.

$\mathbf{p_{tw}}$	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	$\mathbf{t_4}$	$\mathbf{t_5}$	$\mathbf{t_6}$
$\mathbf{w_1}$	4	4	3	1	1	6
$\mathbf{w_2}$	∞	5	6	5	2	4
$\mathbf{w_3}$	3	4	2	∞	3	∞

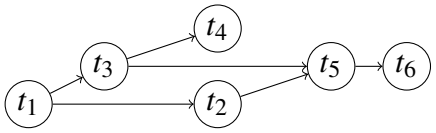


Figure 1.1 – Example of a precedence graph on six tasks.

usually represented by a transitively reduced directed acyclic graph $G(T,A)$ on the tasks, with $A \subseteq T \times T$. For an arc $(t,t') \in A$, task t should be performed before task t'. Figure 1.1 shows an example of a precedence graph $G(T,A)$, where $|T| = 6$. In the example, task t_1 precedes all the remaining tasks and should be executed first. After that, tasks t_2 and t_3 are free and may be executed. One of them is performed, some more tasks may be freed and so on, until all tasks are performed. All topological orders on $G(T,A)$ are valid sequences of tasks and some examples of orders for the graph in Figure 1.1 are presented in Figure 1.2.

Each workstation $s \in S$ at the factory is placed along a conveyor belt and is assigned to exactly one worker $w \in W$, which is responsible for executing a subset of tasks $X_w \subseteq T$. After executing its set of tasks the worker will pass the product to the next workstation. The products of station s_2 can not return to station s_1. Therefore, if there is an arc (t,t') in graph $G(T,A)$, the station that performs task t cannot be placed after that of task t' on the conveyor belt. Figure 1.3 shows a valid assignment of tasks to workers and stations.

A worker can only be assigned to one station and for a given worker w assigned to a station s, the station time of s is given by $P_s = \sum_{t \in X_w} p_{tw}$. Therefore it is possible to calculate the cycle time C of the line as the maximum station time $\max_{s \in S} P_s$. At every C seconds, a new unit will be produced at the assembly line. Thus the production rate of the line r is given by the number of products finished at each time unit ($r = 1/C$). The goal of an ALWABP of type 1 (ALWABP-1) is to minimize $|S|$ given a fixed C. For an ALWABP of type 2 (ALWABP-2), the

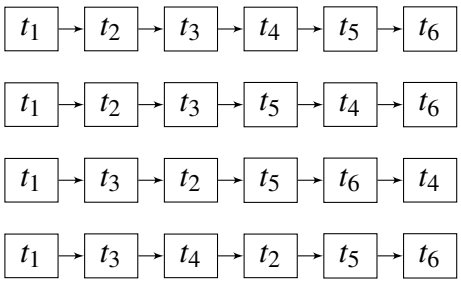
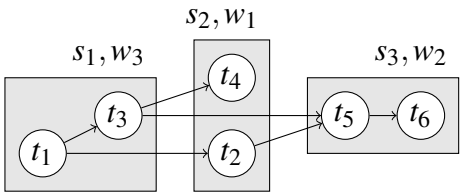
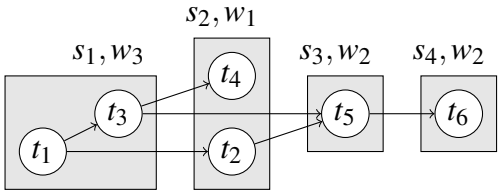


Figure 1.2 – Four examples of valid task orders for Figure 1.1.



p_{tw}	t_1	t_2	t_3	t_4	t_5	t_6
w_1	4	4	3	1	1	6
w_2	∞	5	6	5	2	4
w_3	3	4	2	∞	3	∞

Figure 1.3 – Example of an ALWABP instance and an assignment of tasks to workers and stations (in grey). Upper part: precedence constraints among the tasks. Lower part: task execution times.



p_{tw}	t_1	t_2	t_3	t_4	t_5	t_6
w_1	4	4	3	1	1	6
w_2	∞	5	6	5	2	4
w_3	3	4	2	∞	3	∞

Figure 1.4 – Example of a RALBP instance and an assignment of tasks to robots and stations (in grey). Upper part: precedence constraints among the tasks. Lower part: task execution times.

objective is to minimize C while using a fixed number of stations and workers. An ALWABP of type F (ALWABP-F) aims to find a solution with a fixed number of workers and a fixed cycle time. Finally, an ALWABP of type E (ALWABP-E) minimizes the product $C \cdot |S|$.

For the example of Figure 1.3, the station times of stations s_1, s_2 and s_3 are 5, 5 and 6, respectively. Therefore the cycle time of this solution is 6, which is also optimal for the ALWABP-2, with three stations.

For the Robotic Assembly Line Balancing Problem, the concept of a worker is replaced by types of robots. Since the same type of robot can be assigned to multiple stations, it is unnecessary to bound the number of robots of the same type to one. Therefore, for the same instance of Figure 1.1 and Table 1.1, if we consider four stations, an optimal solution would use two robots of the same type w_2 in stations s_3 and s_4, providing a cycle time of 5, as shown in Figure 1.4.

1.2 Formal Definition

The notation needed for the definition of the ALWABP and the RALBP is presented in Table 5.1.

Table 1.2 – Notation for ALWABP

S	set of workstations;
W	set of workers;
T	set of tasks;
R	set of robots;
M	set of models;
$G(T,E)$	precedence graph of tasks;
$G^*(T,E^*)$	transitive closure of graph $G(T,E)$;
p_{tw}	execution time of task t by worker w;
p_{tr}	execution time of task t by robot r;
$I_w \subseteq T$	set of tasks unfeasible for worker w;
$U_w \subseteq T$	set of tasks feasible for worker w;
$A_t \subseteq T$	set of workers or stations feasible for task t;
$P_t \subseteq T$ and $F_t \subseteq T$	set of immediate predecessors and successors, respectively, of task t in graph $G(T,E)$;
$P_t^* \subseteq T$ and $F_t^* \subseteq T$	set of all the predecessors and successors, respectively, of task t in graph $G^*(T,E^*)$;
C	a real variable, representing the cycle time of a solution;

An ALWABP-2 formulation was proposed by Miralles et al. (2008b). It uses the three sets: stations $S = \{1,2,...,n\}$ (for n stations), workers W and tasks T. Using these sets we define binary variables x_{swt} and y_{sw}. The variable x_{swt} indicates if a task $t \in T$ is assigned to the worker $w \in W$ at the workstation $s \in S$, and the variable y_{sw} indicates if a worker $w \in W$ is assigned to station $s \in S$. Using this notation, the model M_{W1} can be defined.

$$(M_{W1}) \quad \textbf{minimize} \quad C, \tag{1.1}$$

$$\textbf{subject to} \quad \sum_{t \in U_w} p_{tw} \cdot x_{swt} \leq C, \qquad \forall w \in W, s \in S, \tag{1.2}$$

$$\sum_{w \in A_t} \sum_{s \in S} x_{swt} = 1, \qquad \forall t \in T, \tag{1.3}$$

$$\sum_{s \in S} y_{sw} \leq 1, \qquad \forall w \in W, \tag{1.4}$$

$$\sum_{w \in W} y_{sw} \leq 1, \qquad \forall s \in S, \tag{1.5}$$

$$\sum_{w \in A_u} \sum_{s \in S} s \cdot x_{swu} \leq \sum_{w \in A_j} \sum_{s \in S} s \cdot x_{swj}, \qquad \forall j \in T, u \in T, (j,t) \in E \tag{1.6}$$

$$\sum_{t \in U_w} x_{swt} \leq M y_{sw}, \qquad \forall w \in W, s \in S, \tag{1.7}$$

$$x_{swt} = 0, \qquad \forall w \in W, s \in S, t \in I_w, \tag{1.8}$$

$$x_{swt} \in \{0,1\}, \qquad \forall s \in S, w \in W, t \in T, \tag{1.9}$$

$$y_{sw} \in \{0,1\}, \qquad \forall s \in S, w \in W, \tag{1.10}$$

$$C \in \mathbb{R}. \tag{1.11}$$

The objective in an ALWABP-2 problem is to minimize the cycle time, as defined by the variable C in the objective function (5.1). The value of C is defined according to constraints (1.2). Also, by the problem definition, a task and a worker must be assigned to exactly one station, as given by constraints (1.3), (1.4) and (1.5). Constraint (1.7), in turn, defines that the assignment of task t to worker w and station s is only valid if w is the responsible for station $-s$, ensuring the compatibility of the variables x and y.

The task precedence constraints are ensured by constraint (1.6), which explicitly indicates that the station assigned to a task i should be lesser or equal to the station assigned to j, if task i precedes j in the graph $G(T,A)$. Also, if a worker is incapable of doing a task, this task should not be assigned to him, as ensured by constraint (1.8). Finally, the constraints (1.9) and (1.10) define integrality of the variables x_{swt} and y_{sw}.

The variable C is scalar. The variables y_{sw} are defined for each $s \in S$ and $w \in W$, and so have size $|S||W|$. Both sets of variables are overshadowed by x_{swt}. This set of variables is defined for $s \in S$, $w \in W$ and $t \in T$ and so it has size $|T||W||S|$. Therefore, model M_{W1} has $O(|T||W||S|)$ variables.

Constraints (1.2) are defined for $w \in W$ and $s \in S$, and so, there are $|S||W|$ of such constraints. Also, there are $|T|$ constraints (1.3), $|W|$ constraints (1.4) and $|S|$ constraints (1.5). As for the constraints (1.6), they are defined for each arc in the precedence graph $G(T,E)$, and so there are $|E|$ of such constraints. Finally, $|W||S|$ constraints (1.7) are defined. Therefore, the number of constraints of model M_{W1} is $O(|T|+|E|+|W||S|)$.

The RALBP permits an unbounded usage of the same type of worker. Therefore, we obtain a model for the RALBP-2 by removing constraints (1.4) from the model of the ALWABP-2. We call this model for the RALBP-2 M_{R1}.

2 CONTRIBUTIONS OF THIS THESIS

The thesis advances the results on two problems, the Assembly Line Worker Assignment and Balancing Problem (ALWABP-2), and the Robotic Assembly Line Balancing Problem (RALBP). For the ALWABP-2, our contributions are a novel state-of-the-art MIP model with two-index variables and transitivity constraints, a iterated probabilistic beam search heuristic, the use of lower bounds for the unrelated parallel machine scheduling problem, and the task-oriented branch-and-bound that, by using these methods, we prove optimality of every instance in the literature.

For the RALBP-2, we introduce a lower bound that uses chain decomposition to relax a few precedence constraints, a iterated beam search, and a station-oriented branch-bound-and-remember method with cyclic best first search, that has the best results in the literature, and in less time than the known heuristics in the literature.

2.1 Thesis Outline

Chapter 3, will present an analysis of state-of-the-art articles about Assembly Line Balancing Problems with heterogeneous workers. It will begin by presenting classifications of assembly line problems and present some problems related to the heterogeneous assembly line problems. After that, the chapter will present the existent work on lower bounds, heuristics and exact methods for the two studied problems.

Chapter 4 presents our proposed methods for solving the ALWABP-2. Section 4.1 presents and evaluates the proposed MIP models for the ALWABP, that are shown to improve the known models for the problem in instances with up to 25 tasks. A beam search method is also presented in Section 4.2. The method is also evaluated and analyzed in this section and is shown to be competitive with the state-of-the-art heuristic methods for the problem. A branch-and-bound method for ALWABP-2 and its computational evaluation are presented in Section 4.4. With the new results found by our method, we are able to prove optimality of all the instances in the literature.

The solutions proposed for the RALBP-2 are presented in Chapter 5. After presenting a novel lower bound for the problem in Section 5.2, the exact solutions for the problem are presented: a modified MIP model that is executed using CPLEX in Section 5.1, and an iterated branch-bound-and-remember method using cyclic best-first search in Section 5.3. We also present an iterated beam search method using the cyclic best-first search approach. The iterated branch,bound-and-remember (BBR) method is shown to produce better results in a smaller amount of time than all the heuristic and exact solutions in the literature. And the Iterative Probabilistic Beam Search (IPBS) is shown to have competitive results in a limited amount of time.

Finally, general conclusions concerning the heterogeneous assembly line balancing prob-

lems are discussed in Chapter 6.

3 RELATED WORK

The RALBP was introduced early in the nineties (RUBINOVITZ; BUKCHIN; LENZ, 1993) and the ALWABP-2 came years later in the 2000s (NICOSIA; PACCIARELLI; PACIFICI, 2002; MIRALLES et al., 2007). Although these two problems are more than 20 and 10 years old, respectively, there are only a few publications on both of them. There is a lot of literature, though, on other assembly line problems, whose solutions can be used as basis for solving heterogeneous problems. Also, scheduling (GRAHAM et al., 1979; MARTELLO; SOUMIS, 1997) and bin packing problems (Coffman Jr. et al., 2013) are strongly related to heterogeneous assembly line balancing.

3.1 Classification of Assembly Line Balancing Problems

Assembly Line Balancing Problems were classified by Boysen, Fliedner and Scholl (2007) using a triple $[\alpha|\beta|\gamma]$ based on the classification of scheduling problems by Graham et al. (1979).

The α field corresponds to characteristics of the precedence graph. The β field specifies line and station characteristics, as the layout of the line or differences among the stations. Finally, the γ field represents the objective function of the problem.

The value of α is composed of six concatenated parts:

- **Product specific features:** Defines the models being produced. The assembly line could be a single-model production line ($\circ$), a mixed-model production line (*mix*) or a multi-model production line (*mult*).
- **Structure of the graph:** Specifies if the graph has a specific structure (*spec*), like chains or trees, or if the graph assumes any acyclic structure ($\circ$).
- **Processing times:** Describes if the processing times are stochastic (t^{sto}), dynamic (t^{dy}) or static and deterministic ($\circ$).
- **Sequence-dependent increments on the task processing times:** Specify if the execution of a task affects subsequent tasks. A task can affect the time of its direct successor only (δt_{dir}), of all its successors (δt_{indir}) or of none of them.
- **Assignment restrictions:** Specify permissions or prohibitions on the assignment of tasks to stations. Some tasks can have conflicts with other tasks (*inc*) or can be forced to be put together (*link*). There can be restrictions on cumulative assignment of tasks to stations (*cum*). A task can have a fixed station to be placed (fix), can have prohibited stations (*excl*) or can be assigned only to stations of a specific type ($type$). Finally a criterion of minimum (*min*) or maximum (*max*) distance between tasks assigned to the same station can be defined.
- **Processing alternatives:** Define if there are processing time alternatives for the tasks

according to the balancing of the line (pa). If there are also changes to the precedence graph along the balancing, the value of this part is pa^{prec}.

Both the RALBP and the ALWABP specify processing alternatives, according to the worker selected for a station, thus, $\alpha \supseteq \{pa\}$ for them. For ALWABP there are other restrictions, since a worker can not be reused. Thus, for ALWABP, $\alpha \supseteq \{link, cum\}$.

The value of β, is also a concatenation of six parts:

- **Movement of workpieces:** Specifies if the line is paced or unpaced. If it is unpaced, then the line can be synchronous ($unpac^{syn}$) or asynchronous ($unpac$). Otherwise, this part specifies obedience to the cycle time of the line. Each station can be limited to an upper bound ($\circ$), can be exactly equal to an upper bound ($each$) or can overflow the cycle time with a specific probability ($prob$). If there are local cycle times in the line for groups of stations, the value of this field is modified to $\circ^{div}$, $each^{div}$ or $prob^{div}$.
- **Line Layout:** Specifies if the line is serial ($\circ$) or U-shaped (u).
- **Parallelization:** Defines if there is only one line or parallel lines running together. The line can be completely parallelized ($pline$), can have the stations parallelized ($pstat$), can have parallel tasks ($ptask$), or can have parallel working places inside a workstation ($pwork$). No parallelization is represented by $\circ$.
- **Resource Assignment:** Defines how the problem treats the equipment used at each station. The problem can deal with the selection of the equipment to be used in each station ($equip$), can deal with the design of the equipment for each station (res) or can consider the equipment to be fixed for each station ($\circ$).
- **Station-dependent time increments:** It specifies if unproductive idle time at each station is considered (δt_{unp}) or not ($\circ$). Unproductive idle time is the time wasted in operations related to the station, e.g. changing the sides of the U-line or transportation of pieces.
- **Other aspects:** In the other aspects groups, we have bufferization ($buffer$), feeding of a line to another with both being balanced simultaneously ($feeder$), positioning and dimensioning of boxes of material (box), definition of machines to change workpieces positions ($change$).

The RALBP and the ALWABP have $\beta = equip$, since the main characteristic of heterogeneous workers is the selection of equipment to perform each of the stations, or in case of the ALWABP, the worker to execute them.

Finally, the objective function is defined by γ. The types 1, 2 and E defined for SALBP and ALWABP are represented by m, c and $\circ$ respectively. But other objectives are also possible, like the line efficiency (E), the cost of the line (Co), the profit of the line (Pr) or a custom composite score ($score$).

We minimize the cycle time for the two problems studied here. Therefore, we have $\gamma = c$ for

both problems. Thus, the ALWABP-2 is classified as $[pa, link, cum|equip|c]$ and the RALBP-2 is classified as $[pa|equip|c]$.

A second taxonomy was proposed by Battaïa and Dolgui (2013). They presented specifications for each characteristic of line balancing problems:

- **The industrial environment:** Defines if the problem balances a line for assembling, disassembling or applying a series of machining operations.
- **Number of products being produced:** The line can produce a single product repeatedly, multiple different products, or different models of a product.
- **Line layout:** Specifies if the workstations are placed along a basic straight line, a straight line with multiple workplaces, an U-shaped line, a line with circular transfer or an asymmetric line.
- **Characteristics of multiple lines:** When there are multiple lines performing tasks at the same time, they can be independent, can have workstations performing tasks in more than one assembly line or can be parallel lines with crossovers in which products can be transferred from one line to another.
- **Task attributes:** Defines how the tasks relates to other components of the problems and to themselves. The processing time of a task, for example, can be constant, dynamic or uncertain. According to the tasks already assigned, to which station (according to its equipment or worker) or to which line a task is assigned, the processing time can vary.
- **Workstation attributes:** Define the management of resources to workstations. Examples of resources of a workstation are the capacity of the buffer of the workstation, pieces of the equipment, workers or machines used. If the resources are shared by all tasks assigned to a station, we can use a scalar-attribute $R^S(k)$ to define the resource allocation. Otherwise, the use of resources is defined by a vector-attribute $R^V(k)$.
- **Assignment constraints:** Force or prohibit specific relations among decision variables. It specifies if all tasks must be executed, the precedence relation between tasks, if a task should be forced or prohibited to be assigned to the same station as another task, the synchronization of tasks in different lines, the distance (in number of stations or starting times) two tasks should be placed one from another or if a task should be forced or prohibited to be placed at a specific workstation.
- **Workstations constraints:** Specify if there is a constraint between workstation and task attributes or if there is a constraint relating different workstations, for example if using an equipment at a workstation prohibits its use at other workstations.
- **Performance constraints:** Calculate performance measures for the attributes of workstations and tasks and define constraints according to it.
- **Cycle time constraints:** Define a constraint based on cycle time when needed (i.e. problems of type 1). The processing time is defined both for sequentially executed tasks and simultaneously executed tasks.

- **Objective function:** Some possibilities are the minimization of the number of stations, cycle time, cost and efficiency, as well as maximization of the profit and system utilization.

According to this taxonomy, the ALWABP-2 and the RALBP-2 are defined as producing a single product repeatedly along a basic straight line, with occurrence and precedence constraints. The task time attributes depends on the worker attribute of the workstation. In these problems, all tasks assigned to the same workstation share the selected worker or robot, and so, the workstations attributes are defined by a scalar attribute using each worker or robot as an individual resource. All the problems are of type 2 and aim to minimize the cycle time.

3.2 Scheduling on Unrelated Parallel Machines

If we remove the precedence constraints of the ALWABP-2, we have a set of tasks with no particular order that should be assigned to a set of parallel machines. The tasks have a different execution time according to the machine performing it. The parallel machines are unrelated, i.e. each machine can have different, arbitrary task times. Therefore, we have a scheduling of unrelated parallel machines aiming to minimize the makespan (MARTELLO; SOUMIS, 1997).

Scheduling problems are classified according to a triple " $\alpha \mid \beta \mid \gamma$ " (GRAHAM et al., 1979). The first item α presents the relation among machines. If the execution time of a task is equal at all machines, then $\alpha = P$. If the machines have different speeds, then $\alpha = Q$. Finally, if the machines are unrelated, then $\alpha = R$. The second element of the triple indicates specific characteristics of tasks, like preemption or particularities of the tasks. Finally, the last element of the triple represents the objective of the problem. The total execution time of a machine i is called C_i and, then, the makespan of the problem is given by $C_{max} = \max_i \{C_i\}$. Therefore, the scheduling of unrelated parallel machines is called $R||C_{max}$.

Since ALWABP-2 only adds additional restrictions to the unrelated parallel machine problem, ALWABP-2 has always a greater or equal optimal solution than the optimal solution for that problem. Thus, any lower bound for the unrelated parallel machine problem can be used for ALWABP-2, if we remove its precedence constraints.

3.2.1 Lower Bounds

Simple lower bounds for the unrelated parallel machine problem can be obtained by transforming the problem to scheduling processes on identical parallel machines ($P||C_{max}$). In this problem each task has a task time, the number of parallel processors is fixed and the objective is to minimize the cycle time. Given $p_t^- = \min\{p_{tw} \mid w \in W\}$, the minimum execution time of a task in a unrelated parallel machine scheduling problem, we can create an identical parallel machine instance with the task times equal to p_t^- and the number of processors equal to the number of processors in the unrelated parallel machine problem (SELS et al., 2015).

The total processing time of this instance is given by $\sum_{t \in T} p_t^-$. Considering the best case scenario, the total processing time of the tasks will be equally divided among all the machines. Thus, dividing the total time among processors produces the first lower bound for the problem (MARTELLO; SOUMIS, 1997):

$$LC_1' = \left\lceil \frac{\sum_{t \in T} p_t^-}{|S|} \right\rceil \tag{3.1}$$

Using the same related parallel machine instance, since every task must be performed at least once, each task time is a valid lower bound for the instance and, so, we can define lower bound LC_1'':

$$LC_1'' = \max \{ p_t^- \mid t \in T \} \tag{3.2}$$

For the sake of simplicity, from now on, we use LC_1 to define the best of these two bounds:

$$LC_1 = \max \{ LC_1', LC_1'' \} \tag{3.3}$$

3.2.1.1 Linear Relaxation

The unrelated parallel machines problem can be modelled by $M_{R||C_{max}}$ (MARTELLO; TOTH, 1990):

$$\begin{aligned}
M_{R||C_{max}} = \textbf{minimize} \quad & C, && && (3.4)\\
\textbf{subject to} \quad & \sum_{w \in W} x_{tw} = 1, && \forall t \in T, && (3.5)\\
& \sum_{t \in T} p_{tw} \cdot x_{tw} \leq C, && \forall w \in W, && (3.6)\\
& x_{tw} \in \{0,1\}, && \forall t \in T, w \in W, && (3.7)\\
& C \in \mathbb{R}. && && (3.8)
\end{aligned}$$

The model uses binary variables x_{tw}, with $x_{tw} = 1$ if task t is assigned to machine w, and $x_{tw} = 0$ otherwise. Constraint (3.5) ensures that every task is executed exactly once. The value of C is guaranteed to be the makespan of the machines by constraint (3.6). The remaining two constraints define the domain of variables x_{tw} and C.

For this model, a linear relaxation removes the integrality constraints (3.7) so that variables x_{tw} can assume values in the interval from zero to one. An optimal solution for this relaxed problem (C_{LP}) produces a lower bound for the unrelated parallel machine problem.

$$LC_{LP} = \lceil C_{LP} \rceil \tag{3.9}$$

3.2.1.2 Surrogate and Lagrangian Relaxations

Lower bounds can also be obtained by other forms of relaxation such as Lagrangian relaxation (GEOFFRION, 1972) and surrogate relaxation (GREENBERG; PIERSKALLA, 1970). In problems like $R||C_{max}$ some hard constraints make the problem too slow to solve. These constraints, are thus, relaxed. In the case of the Lagrangian Relaxation, the constraints are directly penalized in the objective function. In the case of the Surrogate Relaxation, the hard constraints are joined together and each of them is penalized, producing another constraint, that is desired to be easy to solve.

Martello and Soumis (1997) and Velde (1993) have proposed similar Lagrangian and surrogate relaxations, respectively, for the unrelated parallel machine problem, that relax constraints (3.6). Here we show the process of the Lagrangian relaxation. Using the vector of Lagrangian multipliers $\lambda = (\lambda_1, ..., \lambda_w) \geq 0$, with $\lambda_i > 0$ for at least one value of $i \in W$. The dualization of the constraints (3.6) produces the model

$$L_1(\lambda) = \textbf{minimize} \qquad C + \sum_{w \in W} \lambda_w \left(\sum_{t \in T} p_{tw} x_{tw} - C \right), \tag{3.10}$$

$$\textbf{subject to} \qquad (3.5), (3.7), (3.8). \tag{3.11}$$

and we can rewrite the objective function as:

$$\textbf{minimize} \qquad C\left(1 - \sum_{w \in W} \lambda_w\right) + \sum_{w \in W} \sum_{t \in T} \lambda_w p_{tw} x_{tw} \tag{3.12}$$

Since we are interested in the multipliers that produce the maximum value for this problem, any multipliers for which

$$\sum_{w \in W} \lambda_w \neq 1 \tag{3.13}$$

must be discarded.

With this condition, we are interested in solving the remaining part of the problem:

$$L_1(\lambda) = \textbf{minimize} \qquad \sum_{w \in W} \sum_{t \in T} \lambda_w p_{tw} x_{tw}, \tag{3.14}$$

$$\textbf{subject to} \qquad (3.5), (3.7). \tag{3.15}$$

For a predefined set of Lagrangian multipliers, this problem can be polynomially solved by choosing the worker that minimizes $\lambda_w p_{tw}$ for each job t. This problem has the integrality property (GEOFFRION, 1972). In other words, if we remove the integrality constraints of this model we obtain the exact same result. For linear problems, if this property is satisfied, the result of L_1, for optimal Lagrangian multipliers, is equal to the result of the linear relaxation

LC_{LP} (GEOFFRION, 1972). Although finding the optimal Lagrangian multipliers may be too costly computationally, the result can be approximated by an ascent direction method (VELDE, 1993) or by subgradient optimization (MARTELLO; SOUMIS, 1997).

Another Lagrangian relaxation was proposed by Martello and Soumis (1997). This relaxation dualizes constraints 3.5, using a vector π of multipliers. The resulting model is:

$$L_2(\pi) = \quad \textbf{minimize} \qquad C + \sum_{t \in T} \pi_t \left(\sum_{w \in W} x_{tw} - 1 \right) \tag{3.16}$$

$$\textbf{subject to} \qquad (3.6), (3.7), (3.8). \tag{3.17}$$

For a fixed value of C, this model can be rewritten as:

$$L_2(\pi, C) = \quad \textbf{minimize} \qquad C + \sum_{w \in W} \sum_{t \in T} \pi_t x_{tw} - \left(\sum_{t \in T} \pi_t \right), \tag{3.18}$$

$$\textbf{subject to} \qquad \sum_{t \in T} p_{tw} - x_{tw} \leq C, \qquad \forall w \in W \tag{3.19}$$

$$(3.7), (3.8). \tag{3.20}$$

The objective function is composed of the terms $C - \sum_{t \in T} \pi_t$ and $\sum_{w \in W} \sum_{t \in T} \pi_t x_{tw}$. The first term is constant. Therefore we are interested in solving the second term. Since for each worker we have independent variables, we can solve the subproblem for each worker and, thus, sum all the results to produce the final result. The $|W|$ independent problems, for each machine $w \in W$ are:

$$L_2(\pi, C, w) = \quad \textbf{minimize} \qquad \sum_{t \in T} \pi_t x_{tw}, \tag{3.21}$$

$$\textbf{subject to} \qquad \sum_{t \in T} p_{tw} \cdot x_{tw} \leq C, \tag{3.22}$$

$$x_{tw} \in \{0, 1\}, \qquad \forall t \in T. \tag{3.23}$$

If we transform this problem into a maximization problem, it is equivalent to a knapsack problem. The knapsack capacity is C and for each item t, the size is p_{tw} and the cost is $-\pi_t$. While the knapsack problem is NP-Hard, it can be efficiently solved (MARTELLO; PISINGER; TOTH, 1999). Finally, after solving these knapsack problems, $L_2(\pi, C)$ can be calculated by

$$L_2(\pi, C) = C - \sum_{t \in T} \pi_t - \sum_{w \in W} L_2(\pi, C, w) \tag{3.24}$$

For a given π, the cycle time C that produces the minimum $LC_2(\pi, C)$, is a lower bound on the problem. It can be computed using binary search on the possible cycle times. As in LC_1 the optimal multipliers π can be approximated by subgradient optimization.

3.2.1.3 Additive Improvement

Suppose we have a procedure that produces a lower bound δ and residual costs $\overline{c}$ for a problem P that minimizes the product cx, for any value of x in the domain F. In this case, if

$$\delta + \overline{c}x \leq cx, \qquad \forall x \in F \tag{3.25}$$

then, given any domain $R \supseteq F$, the result of $\delta + \min\{\overline{c}x \mid x \in R\}$ is a lower bound of P (FISCHETTI; TOTH, 1989).

Considering the model $M_{R||C_{max}}$ presented in Section 3.2.1.1, we have costs $c_0 = 1$ for C, $c_{tw} = 0$ for the variables x_{tw} and $c_w = 0$ for the slack variables s_w of constraints 3.6. A solution for the linear relaxation ($\delta = LC_{LP}$) has residual costs: $\overline{c}_0 = 0$, $\overline{c}_{tw} \geq 0$ and $\overline{c}_i \geq 0$ (MARTELLO; SOUMIS, 1997). This residual costs satisfy the property of Fischetti and Toth (1989). Therefore, we can define a domain R that relaxes the domain of model $M_{R||C_{max}}$ by removing constraints (3.5), and given the initial lower bound C_L and any valid upper bound C_U.

$$C = \sum_{t \in T} p_{tw} x_{tw} + s_w \qquad \forall w \in W \tag{3.26}$$

$$C_L \leq C \leq C_U \tag{3.27}$$

$$x_{tw} \geq 0 \qquad \forall w \in W, t \in T \tag{3.28}$$

$$s_w \geq 0 \qquad \forall w \in W \tag{3.29}$$

$$s_w \in \mathbb{Z} \qquad \forall w \in W \tag{3.30}$$

$$x_{tw} \in \mathbb{Z} \qquad \forall w \in W, t \in T \tag{3.31}$$

We must minimize $\overline{c}x$ for the domain R defined. Given the reduced costs for $M_{R||C_{max}}$, we have the problem:

$$\overline{\delta} = \textbf{minimize} \qquad \sum_{w \in W} \sum_{t \in T} \overline{c}_{tw} x_{tw} + \sum_{w \in W} \overline{c}_w s_w \tag{3.32}$$

$$\textbf{subject to} \qquad (3.26), (3.27), (3.28), (3.29), (3.30), (3.31). \tag{3.33}$$

For a fixed value of C, this problem can be decomposed in $|W|$ independent knapsack problems with equality constraint. We can apply a binary search over the values of C and select the cycle time that produces the smaller result. Afterwards the result of this problem can be added to the result of the lagrangian, surrogate or linear relaxation of the problem: $L^a_{LP} = \lceil LC_{LP} + \overline{\delta} \rceil$.

3.2.2 Branch-and-bound methods

Besides the surrogate relaxation for the problem, Velde (1993) also proposed a branch-and-bound method for the unrelated parallel machine scheduling problem that branches by assigning a selected task t to all the available machines. Then, each of these branches is subdivided by assigning another task t' to all available machines. A solution is found when all tasks are assigned. The order in which the tasks are selected is predefined. Given the cost $\gamma_{tm} = \lambda_m p_{tm}$ for a task t and a machine j, with λ_m being the Lagrangian multiplier of the relaxation LC_{LP_1}, the method of Velde (1993) selects the task t whose difference between the two smallest γ_{tm} over all machines is maximum. Then, the method assigns the selected tasks to all machines starting by the machine with the smallest cost γ_{tm}. The lower bound applied in each node of the branch-and-bound tree is the LC_{LP_1}.

Another branch-and-bound, proposed by Martello and Soumis (1997), also selects a task t and creates branches for each available machine. However, instead of using a fixed order of tasks, the method selects a task according to the current partial solution. We define n_t the number of machines to which the task t can be assigned without overloading the machine. The method selects the task with greatest ω_t, where ω_t is the weighted sum of $-n_t$ and $\gamma_{tm''} - \gamma_{tm'}$, with m' and m'' being the fastest and the second fastest machine, respectively, to execute the task t.

3.2.3 Heuristics and Meta-heuristics

Multiple heuristics were explored to solve scheduling on unrelated parallel machines. Velde (1993) has proposed using the solutions found during the execution of his ascent direction method for the surrogate relaxation of the problem. At each step of the ascent direction method, a valid assignment is tested and this assignment is a valid solution for the problem. Therefore the ascent direction can be used as a heuristic for the $R||C_{max}$.

Glass, Potts and Shade (1994) presented and compared three simple methods: a tabu search, a genetic algorithm and a simulated annealing. The tabu search and the simulated annealing method use two neighborhoods: a shift move that changes a task from one machine to another, and a swap move that swaps two tasks from their machines. The genetic algorithm represents a solution using a string with $|T|$ elements with each element being the machine m to which the task i is assigned, and a two-point crossover is used to produce the offspring.

Guo et al. (2007) propose using for the tabu search and the simulated annealing, a neighborhood search. Instead of randomly selecting a valid solution, the methods with neighborhood search select a pool of valid solutions and select the one with the best result for the next iteration of the method. They also present a heuristic using the squeaky wheel meta-heuristic. This method iterates over values of C' and tries to find solutions for the given C' for k iterations, if no solution is found, the value of C' is incremented and a the method restarts.

Piersma and Dijk (1996) propose a simple local search with the same neighborhoods of the method of Glass, Potts and Shade (1994). However, in the method of Piersma and Dijk (1996), all the neighbors are analyzed and ordered from smallest to largest efficiency ($\min_{w' \in W} \{p_{tw'}\}$ for a task $t \in T$ and a worker $w \in W$). After that, they are tested in order until some of the neighbors achieves a better result than the previous incumbent. This solution is, then selected, and the procedure continues.

The local search of Glass, Potts and Shade (1994) is also applied by Sourd (2001) to improve the results of two known heuristics: the ascent direction method proposed by Velde (1993) and a truncated version of the branch-and bound also proposed by Velde (1993). This local search is also used by Lin, Pfund and Fowler (2011) to improve a simple method of LP/roundup. Cutting planes were used to approximate the result of the $R||C_{max}$ (MOKOTOFF; CHRÉTIENNE, 2002; MOKOTOFF; JIMENO, 2002). The methods of Martello and Soumis (1997) were also used in a heuristic. A recovering beam search method that uses their branching strategy and bounds were presented and analyzed by Ghirardi and Potts (2005).

More recently, Fanjul-Peyro and Ruiz (2010) improve over the previous state of the art by using an efficient iterated greedy local search method. Initially, a simple local search generates a valid solution. Then the method removes d tasks from their machines and reassigns them using a greedy method. Each task t of the removed tasks is assigned to the machine $m = argmin_{i \in W} P_i + p_{ti}$. Then the solution is improved by the local search of Glass, Potts and Shade (1994). This method was further improved by Fanjul-Peyro and Ruiz (2011) that have found that in most of the cases the machine for which a task is assigned is one of the machines that needs less time to perform the task. Therefore, the size of the instance and the number of valid assignments can be highly reduced by only allowing a task to be assigned to the k machines that perform it the fastest, with $k = 2$ and $k = 3$ in the article.

Finally, Sels et al. (2015) proposed a hybridize a tabu search and the truncated branch-and-bound of Sourd (2001). Initially a order of the tasks is defined. A neighborhood in this method consists of changing the order of a task. The truncated branch-and-bound is then used to verify if there is a valid solution for the problem considering that the branches are assignments of tasks to all the machines and the next task to be branched is selected according to the order defined in the tabu search.

3.3 The Simple Assembly Line Balancing Problem

In the Simple Assembly Line Balancing Problem (SALBP) the difference among workers is negligible and, so, the execution time of a task is always the same, regardless of the station performing it (SCHOLL; BECKER, 2006). This problem is a simplification of the ALWABP, since an equivalent ALWABP instance can be constructed from a SALBP instance by creating multiple workers that execute a task in the same amount of time. Similarly, the RALBP generalizes the SALBP.

According to the classification of Boysen, Fliedner and Scholl (2007), the SALBP-2 is defined as [| | c]. According to Battaïa and Dolgui (2013) it is classified as single line, single-model, basic straight assembly line, with occurrence and precedence constraints, minimizing the cycle time.

3.3.1 Lower Bounds

Given a RALBP or ALWABP instance, we have the minimum time $p_t^- = \min\{p_{tw} | w \in W\}$ needed to execute a task t. For such instance, we can construct a SALBP instance with the same tasks and precedence graph and with the execution times p_t for each task t equal to the minimum time p_t^- of the task in the RALBP or ALWABP instance. This SALBP instance will always have a smaller optimal solution than the RALBP or ALWABP version. Unlike scheduling on unrelated parallel machines, in SALBP the linear relaxation generally produces weak lower bounds. Instead, a series of simple and polynomial algorithms is used to obtain lower bounds for the problem.

3.3.1.1 SALBP-1

The SALBP-2 solutions can be calculated by solving the SALBP-1 multiple times. The smallest cycle time for which the SALBP-1 solution is smaller or equal to the number of stations of the SALBP-2 instance is the solution for the SALBP-2. This is also true for lower bounds. The smallest cycle time for which the lower bound for SALBP-1 is smaller or equal to the number of stations of the SALBP-2 instance is a valid lower bound for the SALBP-2. Because of this, we start by discussing lower bounds for the type 1.

The simplest lower bounds for the SALBP-1, like scheduling on unrelated parallel machines, generalize bin packing. The lower bound LM_2 considers that a task can be split in multiple parts. Thus, the sum of task times can be equally divided among workstations.

$$LM_2 = \left\lceil \frac{\sum_{t \in T} p_t}{C} \right\rceil \tag{3.34}$$

Besides that, since two tasks with execution time greater than half the cycle time can not share the same station, counting such tasks gives us a lower bound LM_3'. Also, in the best case scenario, all the tasks with execution time equal to half the cycle time will share a station and the lower bound LM_3' can be increased by half the number of station with $p_t = C/2$. This lower bound is called LM_3 (JOHNSON, 1988).

Generalizing LM_3, tasks with execution times greater than $2/3$ of the cycle time receive a weight of 1. Tasks with execution time in the interval $(1/3, 2/3)$, receive weight $1/2$. And tasks with execution time $1/3$ and $2/3$ receive weight $1/3$ and $2/3$, respectively. The remaining tasks are not weighted. The sum of the weights of all tasks gives us a new lower bound

LM_4 (SCHOLL; BECKER, 2006).

A broader concept for bin packing was proposed by Fekete and Schepers (1998). They propose the use of dual feasible functions (DFF). A dual feasible function is a function $f : [0,1] \rightarrow [0,1]$ in which for any subset of tasks $T' \subseteq T$, if

$$\sum_{t \in T'} p_t \leq 1 \tag{3.35}$$

then

$$\sum_{t \in T'} f(p_t) \leq 1 \tag{3.36}$$

If we modify an instance I of the SALBP-1 to have cycle time 1, we will divide each task by the cycle time C and obtain task times p'_t in the interval $[0,1]$. The result of this modified instance I' is the same as the instance I. This new instance can be modified again by applying the DFF to each task and producing a new instance I''. Since any subset of tasks that fits into a station in the instance I' will also fit into a station in the instance I'', the result of SALBP-1 for the instance I'' is a lower bound for the instance I' and, therefore, to I.

Fekete and Schepers (1998) also propose the use of the functions u^k, for $k \in \mathbb{N}$, and U^ϵ, for $\epsilon \in [0, 0.5]$, as DFFs for the bin packing problem.

$$u^k(x) = \begin{cases} x & \text{if } x(k+1) \in \mathbb{Z} \\ \left\lfloor \frac{(k+1)x}{k} \right\rfloor & \text{otherwise} \end{cases} \tag{3.37}$$

and

$$U^\epsilon(x) = \begin{cases} 1 & \text{if } x > 1-\epsilon \\ x & \text{if } \epsilon \leq x \leq 1-\epsilon \\ 0 & \text{if } x < \epsilon \end{cases} \tag{3.38}$$

Also, since the composition of two DFFs is also a DFF, we can construct the DFF function $DFF^{k,\epsilon}(x) = u^k(U^\epsilon(x))$. We can apply lower bound LM_2 for each instance I' produced by applying DFF to all task times of an original instance of SALBP-1 to obtain a lower bound for the original instance. Notice that the lower bounds LM_1, LM_2 and LM_3 are special cases of the lower bound produced by $DFF^{k,\epsilon}$ using $\epsilon = 0$ and k equal to ∞, 1 and 0, respectively. Fekete and Schepers (1998) propose using values of k in the range from 1 to 100 as well as ∞ and Pereira (2014) propose using every ϵ for which $p_t = \epsilon$ or $p_t = 1 - \epsilon$, for any task t. This lower bound found by all these combinations of k and ϵ is called LB_{DFF}.

Morrison, Sewell and Jacobson (2014), Pereira (2014) propose solving the Bin Packing Problem to optimality to produce lower bounds for the SALBP and methods based on linear programming for SALBP were also evaluated in the literature. Dantzig-Wolfe decomposition

approaches have been shown to produce good lower bounds for the problem (PEETERS; DE-GRAEVE, 2006; BAUTISTA; PEREIRA, 2011; PEREIRA, 2014). However, both solving the BPP optimally and approaches to the Dantzig-Wolfe decomposition are too time-consuming to apply during the execution of an enumeration process for the SALBP.

Heads and Tails

The tail of a task t is a lower bound on the number of stations that are needed to execute the tasks following t (F_t^*) in the precedence graph. Similarly, the head of t is a lower bound on the number of stations needed to execute tasks preceding t (P_t^*). A valid tail for a task t divides the execution times of all tasks succeeding t by the cycle time. The same method can be used to define the head of the task, using the preceding tasks. The sum of the head h_t and the tail t_t for any task t is a lower bound for SALBP-1 (JOHNSON, 1988).

Another example of tail calculation uses the tails of succeeding tasks recursively. The method iteratively calculates the tails of each task in reverse topological order. For a task t, the tail corresponds to the greatest lower bound applied to the instance with only the tasks in F_t^* rounded up if it is impossible to add task t without adding another station. Similarly, for calculating the heads of tasks we can apply the same method in direct topological order. The lower bound produced using these tails and heads is called LM_5 (JOHNSON, 1988).

Destructive Improvement

If there is a condition that makes it impossible to produce a feasible solution for a given lower bound, we say that the lower bound was destructed. Therefore, the lower bound can be incremented and tested for the same condition. The algorithm proceeds until the condition does not hold (KLEIN; SCHOLL, 1999). This type of method is called destructive improvement.

If we define the earliest $E_t(m')$ and latest $L_t(m')$ stations a task t can be placed, given a bound on the number of stations m', it is possible to conclude that if $E_t(m') > L_t(m')$, for any $t \in T$, then a feasible solution for m' is impossible, and m' can be incremented. A valid bound for $E_t(m')$ is given by the heads calculated for LM_5: $E_t(m') = \lceil h_t + p_t \rceil$. Similarly, a valid bound for $L_t(m')$ is given by the tail: $L_t(m') = m' + 1 - \lceil p_t + t_t \rceil$. The lower bound produced by this destructive improvement, with these bounds, will be called LM_6.

For a given interval of stations $[m_1, m_2]$ for $0 \leq m_1 \leq m_2 \leq m'$, we can define a set of tasks that could only be assigned to the stations of the interval. This can be defined based on the earliest and latest stations defined for the lower bound LM_5. Given that, we have a restricted set of tasks and if it is impossible to assign these tasks to $M' = m_2 - m_1 + 1$ stations, e.g. the sum of the task times is greater than CM', then it is impossible to find a feasible solution with m' stations. The resulting lower bound of this destructive improvement will be called LM_7.

3.3.1.2 SALBP-2

SALBP-2 can also be relaxed by removing the tasks precedence graph. In this case we obtain the problem $P||C_{max}$ of scheduling a set of tasks on identical machines minimizing the makespan. Optimal solutions for this problem, which is also NP-hard, are lower bounds for SALBP-2. Therefore, the lower bound LC_1 presented in Section 3.2.1 is a also a lower bound for the SALBP-2.

Another lower bound, LC_2, uses the pigeonhole principle. According to this principle, if we have n pigeons and m holes to place them, with $n > m$, then, at least two elements will be assigned to the same hole. We can also conclude that if $n > 2m$, at least three elements will be placed in some of the holes, and, in general, if $n > km$ then there must be one hole with at least $k+1$ elements. Therefore, if we select any set of $|W|+1$ tasks of our original set T, the sum of the times of the two smallest tasks will be a lower bound for the related parallel machine problem. Similarly, if we take any $2|W|+1$ tasks, the sum of the smallest three tasks will produce a lower bound. This method can be generalized and we can say that, for given constant V, if we select a set of $V|W|+1$ tasks, the sum of the times of the smallest $V+1$ task times is a lower bound for $P||C_{max}$. This is valid for any constant for which $|T| > V|W|$.

Therefore, to produce the best possible lower bound, we need to carefully select the set of $V|W|+1$ tasks. So, the vector of tasks is sorted from the greatest task time to the smallest task time. For each value of V, we select the first $V|W|+1$ elements of the vector and the smallest $V+1$ elements of this set produce the lower bound. Then, we iterate over all the valid values for V (from 1 to $|T|/|W|-1$) and, for each value, apply this method. The greatest lower bound found for all values of V is then selected. This lower bound is called LC_2 (KLEIN; SCHOLL, 1996).

Destructive Improvement

Similarly to SALBP-1 it is possible to define a condition based on the earliest station $E_t(C')$ and the latest station $L_t(C')$ that decides if there is a feasible solution for a given cycle time C'. $E_t(C')$ and $L_t(C')$ are calculated exactly as for the lower bound LM_5. If $E_t(C') > L_t(C')$ for any task t, then C' is destructed. This lower bound produced by destructive improvement is called LC_3.

We can also split the stations in two subsets $[1,i]$ and $[i+1,|S|]$ for any $i \in [1,|S|]$. Two bins a and b of sizes $C_0 i$ and $C_0(|S|-i)$, respectively, are created for each of these sets. So, we can divide the tasks in three sets: the tasks whose latest station $L_t(C_0)$ is at most i, the tasks whose earliest station is greater than i, and the remaining tasks. The first group of tasks will be assigned to bin a and the second group, to bin b. Therefore, a new residual problem emerges: assigning the tasks of the third set to the remaining space in bins a and b. If this is not possible, the cycle time C_0 should be incremented, following the logic of destructive improvement.

The problem of assigning items to two bins with fixed sizes is called 2Bin-F. This problem is NP-complete. Despite this, it can be solved in a reasonable time (MARTELLO; TOTH, 1990), because the number of remaining tasks is usually small. This new bound by destructive improvement is called LC_4.

The LC_4 was extended by Vilà and Pereira (2013). They define a bipartite graph between the tasks T and the stations S. Each task $t \in T$ has an arc with infinity capacity to a station $s \in S$ if $E_t(C_0) \leq s \leq L_t(C_0)$. The method also creates a source (b) and a sink (f) node for the graph. For each task t there is an arc (b,t) with capacity p_t and for each station s, there is an arc (s,f) with capacity C_0. If the maximum flow of this graph is smaller than the sum of all the task times, we know that it is impossible to assign the tasks to the stations and obtain a cycle time C_0. Therefore, the lower bound C_0 is destructed and is incremented. The lower bound produced after this destructive improvement will be called LC_5.

3.3.2 Dominance Rules

If a partial solution s' can not lead to a better solution than another partial solution s, we say that s dominates s'. A dominance rule defines if a partial solution s being evaluated is dominated by some other partial solution. This is useful for branch-and-bound methods (MORRISON et al., 2016), since we can prune partial solutions that are dominated by others. The most effective dominance rules for SALBP are listed below:

- **Maximum Load Rule (JACKSON, 1956)** A station is said to be maximally loaded, if no other task can be added without exceeding the cycle time. When the station is not maximally loaded, another task can be assigned to the current station and the new solution will dominate the non-maximally loaded solution.
- **Jackson Dominance Rule (JACKSON, 1956)** A task t' is said to potentially dominate another task t, if t and t' are unrelated in the precedence graph ($F_t \subseteq F^*_{t'}$) and $p_t \leq p_{t'}$. Given a station load s_0 with a task t, if this task t is potentially dominated by an unassigned task t', then a new load s_1 with task t replaced by t' dominates s_0 if this replacement fits in the station. If $p_t = p_{t'}$ and $F_t = F_{t'}$ then there is a symmetry that would cause both solutions to be pruned in a branch-and-bound. Thus, in this case, we say that the task with smaller index dominates the other.
- **Feasible Set Dominance Rule (SCHRAGE; BAKER, 1978)** Consider two assignments of tasks $T' \subseteq T$ and $T'' \subseteq T$ to the same set of stations $S' \subseteq S$. If the set of tasks T'' contains all tasks of set T', then T'' dominates T.

3.3.3 Reduction Rules

In addition to the dominance rules, a series of reduction rules can be applied to the SALBP. Reduction rules modify the instances such that bounds are improved or a dominance rule could be applied. The most effective reduction rules defined for SALBP are presented below:

- **Task Time Incrementing Rule (JOHNSON, 1988; SCHOLL; KLEIN, 1999)** If a task t cannot share a station with any other task, the time p_t can be incremented to C, in SALBP-1 problems.
- **Prefixing (SCHOLL; KLEIN, 1999) -** If the earliest and the latest possible station for a task t are equal, then t must be fixed to that station.
- **Additional precedence relations (FLESZAR; HINDI, 2003)** In the SALBP-1, if a task t' cannot precede a task t, i.e. $\lceil h_{t'} + p_{t'} + p_t + t_t \rceil$ is greater or equal to the current upper bound, then we can add an arc (t,t') to the precedence graph.
- **Task conjoining rule (FLESZAR; HINDI, 2003)** If all possible maximal loads that contain a task t also contain a task t', these two tasks can be merged in a single task.

3.3.4 Branch-and-Bound Algorithms

Branch-and-bound methods for the SALBP-1 are usually divided in task-oriented and station-oriented methods. In task-oriented methods, at each node a task is selected and for all stations that could execute the task, a branch is created with the task assigned to the station. The precedence and load constraints are used to determine valid assignments. Station-oriented methods, in contrast, fill each of the stations in order. The methods start by the first station and generate all possible maximum station loads. For the current station, each maximum station load generates a new branch. In the new branch, the methods repeat the procedure for the next station, until all tasks are assigned to some station.

FABLE was described by Johnson (1988) and is a task-oriented branch-and-bound method for the SALBP-1 that uses depth-first search. With an efficient tree structure to store already explored partial solutions, Nourie and Venta (1991) got better results than FABLE. Their method was called OptPack. The Eureka method (HOFFMANN, 1992) proposed a station-oriented method applied for fixed number of stations iteratively, until the optimal solution is found. Hoffmann (1992) proposed exploring the solutions in a forward and a backward manner. SALOME-1 (SCHOLL; KLEIN, 1997) joined all the features of FABLE, OptPack and Eureka, like bounds, reduction and dominance rules, to provide a faster solution. A different task-oriented method adapted from a resource-constrained project scheduling problem with better results was proposed by Sprecher (2003).

Finally, a branch-and-bound-and-remember method was proposed by Sewell and Jacobson (2012). It is a station-oriented method with memoization of already visited partial solutions.

By storing the partial solutions, we avoid visiting new partial solutions that are dominated by already visited partial solutions, following the Feasible Set Dominance Rule. They also proposed applying cyclic best first search (CBFS) for the SALBP-1. The CBFS for the SALBP-1 has a priority queue for each station. A priority queue for a station k stores valid assignments of subsets of tasks $T' \subseteq T$ to the first k stations. The highest priority in the queues is given to the partial solutions with the smallest lower bound.

At each iteration of the CBFS the method starts by selecting the first element of the first priority queue and generating all maximal station loads for the current station. These loads are inserted in the priority queue of the next station. The method then continues to the next priority queue and repeats the process until the last station. The method follows to the next iteration, removes the first element of each priority queue and adds new corresponding branches. When all queues are empty, or the upper bound is equal to the lower bound, the method stops. This process helps finding complete solutions early in the search, and helps exploring promising solutions first.

Task-oriented branch-and-bound methods for SALBP-2 are uncommon and they usually do not obtain good results. The only method of this type was proposed by Scholl (1993). For station-oriented methods, initially the problem was solved by iterative executions of the SALBP-1 with different cycle times. However this method revisits partial solutions of previous iterations multiple times. This methods were outperformed by SALOME-2 (KLEIN; SCHOLL, 1996). SALOME-2 is a station-oriented branch-and-bound method that uses local lower bounds. In this method a local lower bound on the cycle time evolves during the search. Initially, the first workstation is filled using a cycle time C_0 equal to the initial lower bound. A branch is created and explored for each maximum station load according to this cycle time C_0. If no solution equal to C_0 is found, the cycle time is incremented and new branches are created excluding maximum loads already explored. This cycle time is part of the information of a partial solution and is passed for all the branches created. In the new branches, the method starts exploring maximal station loads considering the cycle time from the branch and increments it if no solution is found for the given cycle time. When a solution is found, the incumbent is updated and the method proceeds until all the branches have been explored.

3.3.5 Heuristics and Meta-Heuristics

For the SALBP-1, Hoffmann (1963) proposed a station-oriented constructive heuristic. The method adds tasks to station by station, as a station-oriented branch-and-bound, but instead of generating a branch for each of the valid maximal station loads, it chooses only one of them: the station load that has the smallest lower bound LM_1. This method was improved by Fleszar and Hindi (2003) by guiding the search using the other lower bounds and by repeating the method of Hoffmann (1963) multiple times and selecting the best solution found. A bounded dynamic programming (BDP) using the Hoffmann Heuristic was proposed by Bautista and

Pereira (2009).

Many meta-heuristics were also applied to SALBP-1 like Tabu Search (CHIANG, 1998; LAPIERRE; RUIZ; SORIANO, 2006), Simulated Annealing (SURESH; SAHU, 1994), Genetic Search (GONÇALVES; ALMEIDA, 2004; SABUNCUOGLU; EREL; TANYER, 2000) and Ant Colony Optimization (BAUTISTA; PEREIRA, 2002), as well as a hybridization of a beam search and an Ant Colony Optimization by Blum (2008). The methods that achieve the best results, though, are partial enumerations. Any branch-and-bound method executed for a limited time, or a limited number of iterations, is a potential heuristic for the problem.

For the SALBP-2, as for the lower bounds, if for a given cycle time a SALBP-1 heuristic returns a solution with the number of stations specified for SALBP-2, then it is a valid solution for SALBP-2. Thus, a repeated application of a SALBP-1 heuristic will provide a heuristic solution for SALBP-2. Besides that, Uğurdağ, Rachamadugu and Papachristou (1997) propose a modification of the simplex method to produce valid solutions for the SALBP-2. Liu, Ong and Huang (2003) propose applying the method of Hoffmann (1963) for the SALBP-1 iteratively for increasing values of cycle time, and after finding valid solutions they apply a local search using task swap and shift moves to improve the results of the heuristic. Meta-heuristics like tabu search (SCHOLL; VOSS, 1997), ant colony optimization (BOYSEN; FLIEDNER, 2008), genetic (KIM; SONG; KIM, 2007) and other evolutionary algorithms (NEARCHOU, 2007), as well as an algorithm using Petri nets were also applied to SALBP-2.

3.4 The Assembly Line Worker Assignment and Balancing Problem

The literature on ALWABP-2 is very recent. The problem was proposed only in 2007. However, a few solutions for the problem have been proposed recently. Most of the articles propose heuristic and meta-heuristic solutions for the problem, but there are also branch-and-bound algorithms that adapt SALBP solutions for the ALWABP. As for the lower bounds, the only known methods in the literature were relaxations to $P||C_{max}$ and the use of LC_1 (MIRALLES et al., 2008a).

3.4.1 Branch-and-Bound Algorithms

The branching strategies proposed for the SALBP-2 and the unrelated parallel machine can be adapted to the ALWABP-2. The method proposed by Miralles et al. (2008b) adapts the iterative branch-and-bound proposed for the SALBP-2 (see Section 3.3.4) in its simplest form. The method iterates over the cycle time and applies a branch-and-bound for ALWABP-F until the ALWABP-F returns a feasible solution. For each iteration, the method applies a station-oriented branch-and-bound algorithm. In the first station, the method branches for each valid worker and all station loads that this worker can execute. Then, in each branch, the method proceeds to the second station and repeats the procedure. The method stops when all tasks have

been assigned, or it is clear that a feasible solution cannot be found for the current cycle time being tested.

Vilà and Pereira (2014) proposed using a variation of the iterated branch-bound-and-remember algorithm, with cyclic best first search proposed by Sewell and Jacobson (2012) (See Section 3.3.4). The method iterates over the cycle times. It begins with a cycle time C set to an upper bound of the problem. It then tests if there is a valid solution with value equal to $C-1$. If a solution is found, the cycle time is decremented, otherwise the smallest cycle time for the problem is C. Like in the method of Miralles et al. (2008b), for each cycle time, Vilà and Pereira (2014) apply a station-oriented branch-and-bound method for the ALWABP-F. Each partial solution is represented by the tasks and workers already assigned. As in the CBFS for the SALBP, the method uses a priority queue with partial solutions for each station and cyclically evaluates a branch of each queue. A branch is explored by generating all maximal stations loads for all unassigned workers and adding them to the priority queue of the next station. The method also memorizes partial solutions. If the current partial solution has the same set of tasks assigned to the same set of workers than another partial solution stored, the method prunes the current branch.

3.4.2 Upper Bounds

Based on the work of Scholl and Voß (1997), Moreira et al. (2012) proposed rule-based constructive heuristics for the ALWABP-2. The method uses iterative applications of a heuristic for ALWABP-1 with fixed cycle times C.

The method fills the stations sequentially. Starting from the first station the method tries all workers that are unassigned. For each of the workers $w \in W$, a subset of tasks $T'_w \subseteq T$ is selected by a greedy algorithm. The tasks whose dependencies are already satisfied in previous stations are sorted by one of the 16 task priority rules proposed and are assigned to T'_w. If the task added exceeds the cycle time, the task is removed from T'_w and the method proceeds to the next task, until there are no tasks available. After the method defines a T'_w for each worker, it uses one of three proposed worker priority rules to select the worker w^* that will be responsible for the current station. Thus the worker w^* and the tasks T'_{w^*} are assigned to the station and the method repeats the procedure for the next station, until all tasks are assigned. Based on tests, (MOREIRA et al., 2012) have shown that the best task and worker selection rules are $MaxPW^-$ and $MinRLB$, respectively. $MaxPW^-$ attributes for each task a minimum positional weight

$$pw_t^- = p_t^- + \sum_{t' \in F_t^*} p_{t'}^- \tag{3.39}$$

and prioritizes the task with maximum pw_t^-. $MinRLB$ calculates, for each worker, the lower bound LC_1 given the remaining tasks and workers, and selects the worker with the smallest bound.

3.4.2.1 Meta-heuristics

Chaves, Miralles and Lorena (2007) and Chaves, Lorena and Miralles (2009) propose two algorithms using Clustering Search (OLIVEIRA; LORENA, 2004), a method based on Hybrid Evolutionary Algorithms. These algorithms identify promising search space regions, given solutions found by some simpler heuristic. After that, the algorithms apply a local search only in these regions.

Moreira and Costa (2009) propose a minimalist tabu search with three simple neighborhoods: moving a task from a station to the other, swap two tasks from different stations and swap two workers from different stations. Their method allows solutions that do not satisfy the precedence constraints and solutions that assign tasks to invalid workers. These penalties are dynamically increased if the number of iterations with invalid solutions increases. With this simple procedure they have decreased the time needed to solve the problem and increased the accuracy of it, as compared to the method of Chaves, Lorena and Miralles (2009).

Besides proposing the constructive heuristic explained in Section 3.4.2, Moreira et al. (2012) also introduced a Biased Random Key Genetic Algorithm. In this type of method, given a random key value, it must be decoded to a valid solution for the problem. The genetic algorithm, then, tests different solutions by generating different keys. For each generated solution, the method also applies a local search with the same three movements applied in the Tabu Search of Moreira and Costa (2009). The algorithm presented competitive results to the previously known methods.

More recently, Mutlu, Polat and Supciller (2013) proposed an iterative genetic algorithm. The method is split in three levels. The most external level iterates over the cycle times using a binary search. This cycle time is then tested by the two other levels. The second level uses a genetic algorithm to produce task orders. Each gene is a valid topological sorting of the tasks according to the precedence graph. Given two of these genes a and b, the crossover operation selects a point $n \leq |T|$ and initializes two new genes a' and b' with the first n elements of a and b, respectively. The remaining part of the vector a' and b' is filled using the remaining tasks in the order they appear in b and a, respectively. An example of this method, introduced for SALBP by Davis (1985), is presented in Figure 3.1. This process can also be extended for two selected points in the vector of tasks (LEU; MATHESON; REES, 1994). The third level of the method is a local search to select an order of workers, and according to that and the order of tasks of the current gene, the tasks are also assigned to stations. Two neighborhoods are used in this level: a swap of two workers and an operation that reverses the order of the workers between two points.

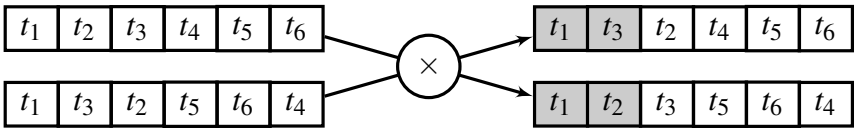


Figure 3.1 – Example of a one-point crossover as proposed by Mutlu, Polat and Supciller (2013). The example uses the second task as the point of crossover and is valid for the ALWABP-2 instance in Figure 1.1.

3.5 The Robotic Assembly Line Balancing Problem

As in the SALBP and in the ALWABP, problems of types 1 (RUBINOVITZ; BUKCHIN; of Manufacturing Engineers, 1991) and 2 (LEVITIN; RUBINOVITZ; SHNITS, 2006) of the RALBP have been studied, for, respectively, minimizing the number of stations and minimizing the cycle time, and like in the SALBP and the ALWABP, the type 2 can be solved by iteratively solving the type 1.

Despite the clear differences, exact methods for the SALBP (SALVESON, 1955; SCHOLL; BECKER, 2006; KLEIN; SCHOLL, 1996; MORRISON; SEWELL; JACOBSON, 2014; VILÀ; PEREIRA, 2013) and the ALWABP (MIRALLES et al., 2007; MIRALLES et al., 2008a; 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; SEWELL; JACOBSON, 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.

For the RALBP-1, the lower bound LM_1 of the SALBP-1, was used by Rubinovitz, Bukchin and Lenz (1993). The lower bound LM_1, as LC_1 also relaxes the indivisibility of tasks constraint. Therefore:

$$LM_1 = \left\lceil \frac{\sum_{t \in T} p_t^-}{C} \right\rceil$$

Since in the best case scenario the tasks are also executed by the fastest robot to perform it, the lower bound is also valid for the RALBP.

Rubinovitz, Bukchin and Lenz (1993) also proposed a station-oriented branch-and-bound algorithm. The method initializes the first station and creates a branch for each pair of a root task in the precedence graph and each of the robots. Thus it expands each of the branches. The expansion of a branch is made by assigning each of the available unassigned tasks to the current

station. If none of the tasks fits the station, a new station is generated and a branch for each pair of available unassigned task and available robot is created. The method continues expanding the branches. A solution is found when a branch has all tasks assigned. The method uses a best-first search, using LC_1 as lower bound to prioritize the partial solutions. In case of ties, the method selects the branch in the deepest level.

A heuristic for the RALBP-1 can also be obtained by truncating the branch-and-bound algorithm, as also proposed by Rubinovitz, Bukchin and Lenz (1993). The number of nodes evaluated at each step of the algorithm is truncated by limiting the number of levels for which the nodes are stored to k. When a new level l is reached the solutions of level $l-k$ are removed from the list of branches to be explored. No further reduction or dominance rules are applied.

For the RALBP-2, Levitin, Rubinovitz and Shnits (2006) proposed a genetic algorithm. In their method, they use the same genotype codification of tasks and crossover operations as the method of (MUTLU; POLAT; SUPCILLER, 2013) for the ALWABP-2 (See Section 3.4.2.1). Each genotype task vector v is decoded to a RALBP-2 solution by assigning the tasks to stations according to the order they appear in v and assigning robots to each station. Therefore, the vector v must be divided in $|W|$ parts, and each part of the vector will be assigned to a station. Levitin, Rubinovitz and Shnits (2006) also proposed applying local optimization by hill climbing to the solutions found by the genetic algorithm.

Particle Swarm Optimization (PSO) was also used to solve RALBP-2 and achieved competitive results (NILAKANTAN; PONNAMBALAM, 2012). In this method, each particle is a valid order of tasks for the problem, and the velocity of a particle approximates the current particle to the global best and the local best by exchanging tasks of the particle. To define the fitness value of a particle, a valid solution is defined considering the task order of the particle. The method to calculate the fitness function starts with a cycle time equal to the lower bound. Then the tasks are assigned in order to the stations until the current station is full, i.e. the next task can not be assigned to the current station. in that case the next station is started with the next task. The method proceeds until all the tasks are assigned or until there are no more stations available. In this case, the solution is impossible and the cycle time is incremented. The fitness value is the result of the cycle time for which a valid solution is found for that particle. After the particle swarm optimization method is concluded, the method improves the best solution found by swapping tasks. They also use a variation of the Cuckoo's Search (CS-PSO) (KENNEDY; EBERHART, 1995) in which the cuckoos are represented by a order of tasks, like in the PSO. New cuckoos are generated every iteration from abandoned cuckoos by using a crossover operator equal to the operator of (MUTLU; POLAT; SUPCILLER, 2013) (See Section 3.4.2.1). These cuckoos, then, move according to the velocity defined for the particle swarm optimization, and afterwards, a fraction of them is abandoned and substituted by new cuckoos. The PSO and the CS-PSO have the best results in the literature for the RALBP-2.

4 SOLUTIONS FOR THE ASSEMBLY LINE WORKER ASSIGNMENT AND BALANCING PROBLEM

In this chapter we present and evaluate solutions, both exact and heuristic, for the Assembly Line Worker Assignment and Balancing Problem of Type 2. First, a Mixed-Integer Programming model is presented. It decreases the number of variables in the model and adds continuity constraints that limit the solution space of the model. Then the lower bounds for the problem are studied. We propose using the unrelated parallel machine scheduling problem lower bounds, since it is a relaxation of the ALWABP. After that, a station-oriented beam search heuristic for the problem that produces initial solutions for our branch-and-bound algorithm. Our branch-and-bound differs from other exact solutions for the ALWABP because we use a task-oriented branching strategy instead of a station-oriented strategy. All these methods are evaluated and shown to be competitive with the literature.

4.1 Mixed-Integer Programming Formulations for the ALWABP-2

The only model in the literature for the ALWABP-2 was proposed by Miralles et al. (2008b). This model was presented in Section 1.2. Here we propose a novel model with two-index variables and this modification makes it possible to add continuity constraints that are presented in further subsections.

4.1.1 Formulation with Two-Index Variables

The assignment of tasks to workers induces relationships among workers according to the relationships among tasks. Let us assume that task t' succeeds task t, t is assigned to worker w and t' is assigned to w'. In this case, the worker w should precede worker w' in the line. Therefore, any topological order of this worker precedence graph yields a valid assignment of worker to a linear sequence of stations. If the graph has a cycle, and therefore no topological order, the current partial solution is invalid. An example is shown in Figure 4.1. First, tasks t_2 and t_3 are assigned to worker w_2 and task t_4 is assigned to worker w_3. So, according to the relationships of the tasks, worker w_2 should precede worker w_3. After that, task t_1 can not be assigned to the worker w_3, otherwise a cycle would be created in the induced graph. In our example, t_1 is assigned to worker w_1. Because of this, a relation between workers w_1 and w_2 is created. Although no relation induced by precedence graph exists among w_1 and w_3, a precedence between them follows by transitivity.

Using the fact that we only need to assign tasks to workers ensuring that there is no cycle in the induced graph, we can define a model M_{W2}. Model M_{W2} uses variables x_{wt} such that $x_{wt} = 1$ if task $t \in T$ has been assigned to worker $w \in W$, and d_{vw} such that $d_{vw} = 1$ if worker $v \in W$

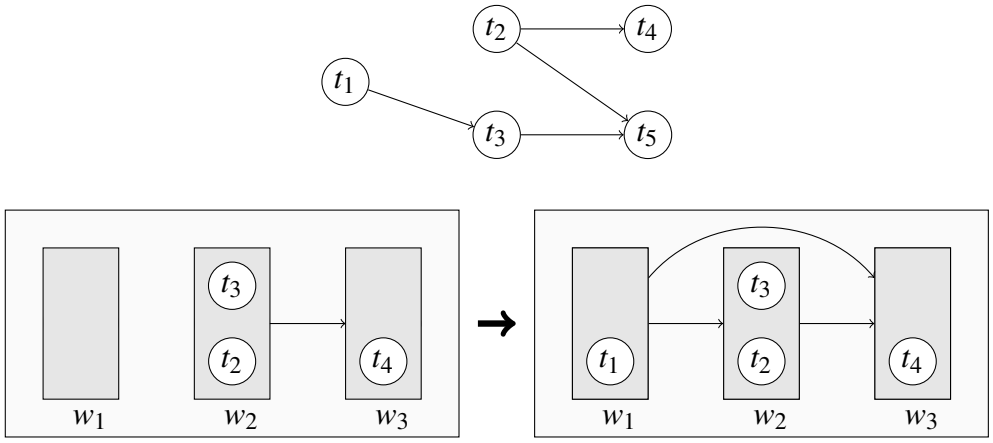


Figure 4.1 – Worker dependencies after assignment of task 1 to worker 1.

must precede worker $w \in W$. In this way, we obtain a model M_{W2} as follows:

$$\textbf{minimize} \quad C, \tag{4.1}$$

$$\textbf{subject to} \quad \sum_{t \in A_w} p_{tw} x_{wt} \leq C, \qquad \forall w \in W, \tag{4.2}$$

$$\sum_{w \in A_t} x_{wt} = 1, \qquad \forall t \in T, \tag{4.3}$$

$$d_{vw} \geq x_{vt} + x_{wt'} - 1, \qquad \forall (t,t') \in E, v \in A_t, w \in A_{t'} \setminus \{v\}, \tag{4.4}$$

$$d_{uw} \geq d_{uv} + d_{vw} - 1, \qquad \forall \{u,v,w\} \subseteq W, |\{u,v,w\}| = 3, \tag{4.5}$$

$$d_{vw} + d_{wv} \leq 1, \qquad \forall v \in W, w \in W \setminus \{v\}, \tag{4.6}$$

$$x_{wt} \in \{0,1\}, \qquad \forall w \in W, t \in A_w, \tag{4.7}$$

$$d_{vw} \in \{0,1\}, \qquad \forall v \in W, w \in W \setminus \{v\}, \tag{4.8}$$

$$C \in \mathbb{R}. \tag{4.9}$$

In this model, constraints (4.2) define the cycle time C of the problem. Each task should be executed by exactly one worker, as assured by constraints (4.3). The graph of worker precedence is induced by the task precedence graph, as specified in constraints (4.4). If two related tasks t and t' are assigned to the workers v and w respectively, then if v precedes w, d_{vw} is set to 1. The constraints (4.5) ensure transitivity of d_{vw}. Finally, constraints (4.6) enforce anti-symmetry guaranteeing that the worker precedence graph remains acyclic. As a consequence of these constraints, the order of the workers along the stations can be found in linear time by topological sorting. This model has $O(|W|^2 + |W||T|)$ variables. Similarly, the number of constraints is $O(|E||W|^2 + |W|^3 + |T|)$.

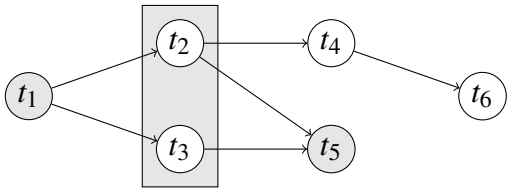


Figure 4.2 – Continuity constraints after tasks t_1 and t_5 have been assigned to the same worker w. Tasks t_2 and t_3 must also be assigned to w.

4.1.2 Continuity constraints

For SALBP-1, Peeters and Degraeve (2006) have shown that while loading a station k, if a task t is preceded and succeeded by tasks already assigned to that station, then the task t must also be assigned to station k. Generalizing this definition, we could observe that: if two tasks i and k are assigned to the same worker w, then all tasks j that are successors of i and predecessors of k must also be assigned to w. It is called a continuity constraint and it is exemplified in figure 4.2. In the Figure, the tasks t_1 and t_5 are assigned to the same worker w and, since tasks t_2 and t_3 are in between the former two, they must also be assigned to w. Otherwise the workers precedence graph would have a cycle and the solution would be invalid.

$$x_{wj} \geq x_{wi} + x_{wk} - 1, \qquad \forall i \in T, j \in F_i^*, k \in F_j^*, w \in A_i \cap A_j \cap A_k. \tag{4.10}$$

Similarly, given that a task t is assigned to a worker w, and a succeeding task t' is unfeasible for w, all tasks succeeding t' should also be unfeasible for w, since if one of them is assigned to w a cycle would be formed in the workers precedence graph. An example of this constraint is shown in Figure 4.3. Task t_1 is assigned to worker w and the task t_2 is unfeasible for w, so all tasks succeeding t_2 (namely t_4, t_5 and t_6) should also be set as unfeasible for w. This constraint is expressed by

$$x_{wk} + x_{wi} \leq 1, \qquad \forall i \in T, j \in F_i^*, k \in F_j^*, w \in A_i \cap (T \setminus A_j) \cap A_k. \tag{4.11}$$

Let model M_{W3} be model M_{W2} with additional constraints (4.10) and (4.11). Model M_3 has the same number of variables as M_2: $O(|W|(|T|+|W|))$. Also, it has $O(|E^*||T||W| + |W|^3 + |E||W|^2)$ constraints, given the constraints (4.10) and (4.11). Therefore this model has less variables but more constraints than M_{W1}.

4.1.3 Computational Experiments

In this section we will compare the performance of the MIP models M_{W2} and M_{W3} with that of model M_{W1}, that was presented in equations (1.2) to (1.11). All models were solved using

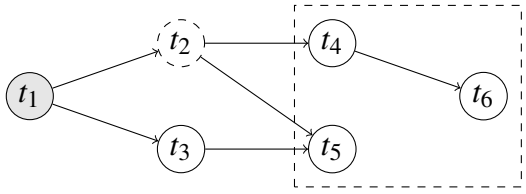


Figure 4.3 – Continuity constraints with t_1 assigned to worker w and t_2 unfeasible for worker w. Tasks t_4, t_5 and t_6 are, therefore, unfeasible for worker w.

Table 4.1 – Instance characteristics. The 320 instances are grouped by five two-level experimental factors into 32 groups of 10 instances.

Factor	**Low Level**	**High Level**
Number of tasks $\|T\|$	$25-28$	$70-75$
Order strength (OS)	22% - 23%	$59\%-72\%$
Number of workers $\|W\|$	$\|T\|/7$	$\|T\|/4$
Task time variability (Var)	$[1,t_i]$	$[1,2t_i]$
Number of infeasibilities (Inf)	10%	20%

the commercial solver CPLEX 12.3 compiled with the GNU C compiler 4.6.3 with maximum optimization. The experiments were done on a PC with a 2.8 GHz Core i7 930 processor and 12 GB of main memory, running a 64-bit Ubuntu Linux. All tests used only one core.

4.1.4 Instances

First of all, let us introduce the 320 instances for the ALWABP-2 proposed by (CHAVES; MIRALLES; LORENA, 2007). These instances are the standard benchmark suite for the problem. They consist of the four groups Roszieg, Heskia, Tonge and Wee-Mag. These four groups differ by the number of tasks and the order strength[1] (OS) of the precedence graph. Each of the instance types is divided in eight groups of ten instances, based on three parameters: the number of workers, the variability (Var) of the task execution times and the percentage of infeasibilities (Inf). For each of these parameters, a low level and a high level are defined, as shown in Table 4.1.

Roszieg and Heskia are the sets of instances with a low number of tasks (25 and 28, respectively), and order strength equal to $71,67\%$ and $22,49\%$ respectively. Tonge and Wee-Mag have a high number of tasks (70 and 75 respectively) with a high order strength ($59,42\%$) in the case of Tonge and a low order strength ($22,67\%$) in the case of Wee-Mag.

[1]The order strength is number of precedence relations of the instance in percent of all possible relations $\binom{|T|}{2}$.

Table 4.2 – Comparison of MIP models M_{W1}, M_{W2}, and M_{W3} on instances Roszieg and Heskia.

				Model					
				M_{W1}		M_{W2}		M_{W3}	
Instance	$\|W\|$	Var	Inf	Nodes	t(s)	Nodes	t(s)	Nodes	t(s)
Roszieg	4	L	10%	56.9	0.6	2340.4	1.1	37.8	0.7
			20%	1.1	0.6	936.0	0.4	11.7	0.4
		H	10%	156.6	0.8	2849.5	1.3	58.6	1.5
			20%	82.9	0.8	3268.4	1.6	53.8	0.8
	6	L	10%	2715.0	12.1	47176.3	52.4	249.9	4.6
			20%	2601.3	11.3	36555.7	29.0	168.7	2.3
		H	10%	3467.0	13.4	83900.5	66.2	389.0	6.3
			20%	2785.0	11.8	50294.3	44.2	281.5	4.5
Heskia	4	L	10%	0.0	0.6	105.2	0.2	29.8	0.3
			20%	25.0	0.6	198.6	0.3	37.5	0.2
		H	10%	65.0	0.7	136.2	0.2	49.0	0.3
			20%	24.3	0.7	130.5	0.3	45.5	0.2
	7	L	10%	1535.9	13.4	1552.2	4.6	86.8	1.0
			20%	1174.1	11.1	940.8	2.2	102.4	1.0
		H	10%	1677.8	12.9	735.9	2.5	115.4	1.1
			20%	1344.1	13.4	663.3	2.8	151.7	1.3
Averages				1107.0	6.6	14486.5	13.1	122.1	1.7

4.1.4.1 Results

Table 4.2 shows the average number of nodes and the average computation time needed to solve the instances to optimality using the three models for the 16 groups of instances with a low number of tasks, namely Roszieg and Heskia (as defined in Section 4.1.4). The instance groups Tonge and Wee-Mag with a high number of tasks are not shown, since none of them could be solved to optimality within an hour.

Overall model M_{W2} needs significantly more nodes than M_{W1}, and is a factor of about two slower. It executes more nodes per second, and has a better lower bound, but CPLEX is able to apply more cuts for model M_{W1} at the root, such that in average model M_{W2} has no advantage on the tested instances.

On the other hand, model M_{W3}, with the continuity constraints applied, needs significantly less nodes and time compared to model M_{W1} (confirmed by a Wilcoxon signed rank test with $p < 0.01$). The results show that the continuity constraints are very effective, in particular for a high order strength and for high numbers of workers.

4.2 Lower Bounds

In the next subsections we will present valid relaxations of the ALWABP-2, as well as adaptations of the relaxations of the SALBP-2 and $R \,||\, C_{\max}$, that can be applied to ALWABP-2 in a straightforward manner.

4.2.1 Relaxations to SALBP-2

All valid lower bounds for SALBP-2 apply to the relaxation of ALWABP-2 (see Section 3.3.1). In particular, we use the lower bounds

$$LC_1 = \max\left\{\max\{p_t^- \mid t \in T\}, \left\lceil \sum_{t\in T} (p_t^-)/|S| \right\rceil\right\} \quad \text{and}$$
$$LC_2 = \max\left\{\sum_{0\le i\le k} p^-_{k|S|+1-i} \;\middle|\; 1 \le k \le \left\lfloor \frac{|T|-1}{|W|} \right\rfloor\right\}.$$

(The bound LC_2 supposes that the tasks are ordered such that $p_1^- \ge \cdots \ge p_{|T|}^-$.)

We further use the SALBP-2 bounds on the earliest and latest possible station of task t for a given cycle time C

$$E_t(C) = \left\lceil \left(\sum_{j\in P_i^*} p_j^- + p_t^- \right)/C \right\rceil \quad \text{and} \tag{4.12}$$
$$L_t(C) = |S| + 1 - \left\lceil \left(\sum_{j\in F_i^*} p_j^- + p_t^- \right)/C \right\rceil \tag{4.13}$$

to obtain the lower bound LC_3, defined as the smallest cycle time C such that $E_t(C) \le L_t(C)$ for all $t \in T$.

4.2.2 Relaxation to $R \,||\, C_{\max}$

Effective lower bounds for $R \,||\, C_{\max}$ have been proposed by Martello and Soumis (1997) (see Section 3.2.1). Their lower bounds L_1 and L_2 are obtained by Lagrangian relaxation of the cycle time constraints (4.2) and the assignment constraints (4.3), respectively. Martello and Soumis (1997) further propose an additive improvement that can be applied to L_1 to obtain a bound $L_1^a \ge L_1$, as well as an improvement by cuts on disjunctions, that may be applied to L_1^a and L_2 to obtain lower bounds $\overline{L}_1^a \ge L_1^a$ and $\overline{L}_2 \ge L_2$.

4.2.3 Linear relaxation of ALWABP-2 models

Bounds obtained from linear relaxations of integer models for the SALBP-2 are usually weaker than the SALBP-2 bounds of Section 4.2.1. However, the relaxation to minimum task execution times weakens the ALWABP-2 bounds considerably. Therefore, the linear relaxations of model M_{W3} provides a useful lower bound for the ALWABP-2 (MOREIRA et al., 2012).

4.2.4 Comparison of lower bounds

The present section will analyze the lower bounds proposed above by comparing their strength. To compute the lower bound L_1 we use the ascent direction method of Velde (1993). This bound was improved to $\overline{L}_1^a$, as proposed by Martello and Soumis (1997). Their method applies a binary search for the best improved bound, which is obtained by solving $|S|$ knapsack problems of capacity C for each trial cycle time C. Different from Martello and Soumis (1997) we solve the all-capacities knapsack problem by dynamic programming only once and use the resulting table during the binary search. The knapsack problems that arise when computing L_2 and $\overline{L}_2$ by subgradient optimization are solved similarly.

Every lower bound proposed was executed for the 320 instances defined in literature for the problem. The average computational time for each bound and the average deviation from the optimal result are calculated for each method and plotted in Figure 4.4. The linear relaxation of models M_{W2} and M_{W3} present the best results over all bounds, but using the greatest average computation time. In particular, the continuity constraints, added by model M_{W3}, improve the average relative deviation by 10%, using more running time. Both models have a significantly better average relative deviation than the version with three-index variables M_{W3}.

The results for the relaxation of $R \,||\, C_{\max}$ are slightly worse than M_{W2}, and M_{W3}, but calculated two orders of magnitude faster. The strong relation of the two problems makes it possible to use the subgradient relaxations of the $R \,||\, C_{\max}$, that is calculated one order of magnitude faster. Finally, the lower bounds from the relaxation to SALBP are weaker than most of the other lower bounds, except LC_1, but another order of magnitude faster. The SALBP-2 based lower bounds LC_2 and LC_3 are weaker than the other bounds, but are the fastest lower bounds. LC_1 differently is one of the fastest methods while producing good relative deviations for the instances studied.

To estimate a possible value for the final result, the linear relaxation of M_{W3} has shown the best average result and must be used in these cases. However, when calculating the lower bound many times along the execution of a program, we usually use faster lower bounds that achieve satisfactory results. Namely, we use the lower bounds from the relaxation to SALBP (LC_1, LC_2 and LC_3) and $\overline{L}_1^a$. These bounds are complementary. LC_1, for example, obtains the best results of the bounds used in average at the root node of a branch-and-bound method, but is ineffective when the partial solution evolves.

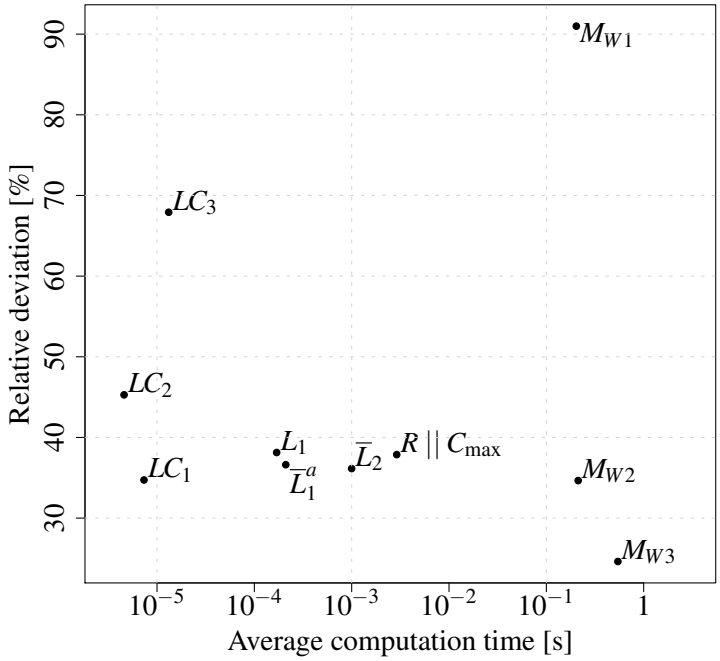


Figure 4.4 – Comparison of lower bounds.

4.3 An Iterative Probabilistic Beam Search for the ALWABP

In this section, we describe an Iterative Probabilistic Beam Search (IPBS) for the ALWABP-2. It systematically searches for a small cycle time by trying to solve the feasibility problem ALWABP-F for different candidate cycle times from an interval ending at the current best upper bound. For each candidate cycle time C, a probabilistic beam search tries to find a feasible allocation.

An upper bound search starts from a known feasible cycle time and tries to reduce it iteratively. A common strategy is to decrement it successively and try to find a better feasible solution by some heuristic algorithm. While this works for exact ALWABP-F methods, it is not as good for heuristic assignment procedures, since they are not monotone. A heuristic method could find a solution for a cycle time and could not find a solution for larger cycle times. Therefore, we propose to modify the upper bound search to examine an interval of cycle times ending at the current best upper bound. If the current lower and upper bounds on the cycle time are $\underline{C}$ and $\overline{C}$, the upper bound search will try to find a feasible solution for all cycle times between $\max\{\underline{C}, \lfloor p\overline{C} \rfloor\}$ and $\overline{C}-1$ for a given factor $p \in (0,1)$ and update $\overline{C}$ to the best cycle time found, if any. Otherwise, the upper bound search continues with the same interval. Since the beam search is probabilistic this may produce a feasible solution in a later trial. The interval search depends on three parameters: the minimum search time $t_{\min}$, the maximum search time $t_{\max}$ and the maximum number of repetitions r. The search terminates if the cycle time found equals the

lower bound, or if the maximum time or the maximum number of repetitions are exceeded, but not before the minimum search time has passed. Initially, the value of $\underline{C}$ is set to the best of all lower bounds. The initial upper bound $\overline{C}$ is determined by a single run of the beam search with a beam factor of one.

4.3.1 Probabilistic beam search for the ALWABP-F

Each time a cycle time should be tested, the ALWABP-F problem is solved heuristically using a probabilistic beam search. The basis for the probabilistic beam search is a station-based assignment procedure, which assigns tasks in a *forward* manner station by station.

For the ALWABP we have to decide which worker and which tasks to assign to the current station. This is accomplished by applying a task assignment procedure to all workers which are not yet assigned, and then choosing the best worker for the current station by a worker prioritization rule. After that decision, the next station is processed. The procedure succeeds if an assignment using at most the available number of stations is found. Station-based procedures can be also applied in a *backward* manner, assigning tasks whose successors have been assigned already. For this it is sufficient to apply a forward procedure to an instance with reversed dependencies.

Considering a specific worker the task assignment procedure repeatedly selects an available task and assigns it to the current station. A task is *available* if all its predecessors have been assigned already. If none of the available tasks fits in the current station (i.e. the execution time of the task is greater than the idle time of the current station) the station is said to be full. If there are several available tasks the highest priority task as defined by a prioritization rule is assigned next. When a station is full, the algorithm continues with the next station.

The probabilistic beam search extends the station-oriented assignment procedure in two aspects. First, when assigning tasks to the current station, it randomly chooses one of the available tasks with a probability proportional to its priority. Second, it applies beam search to find the best assignment of workers and their corresponding tasks.

Beam search is a truncated breadth-first tree search procedure (LOWERRE, 1976; OW; MORTON, 1988). When applied to the ALWABP-F, it maintains a set of partial solutions called the *beam* during the station-based assignment. The number of solutions in the beam is called its *width* γ. Beam search extends a partial solution by assigning each available worker to the next station, and for each worker, chooses the tasks to execute according to the above probabilistic rule. For each worker this is repeated several times, to select different subsets of tasks. The number of repetitions is the beam's *branching factor* f. Among all new partial solutions the algorithm selects those of highest worker priority to form the new beam. The number of solutions selected is at most the beam width.

Task and worker prioritization rules are important for the efficacy of the station-oriented assignment procedure. We have chosen the task priority rule $MaxPW^{-}$ and the worker priority

rule *MinRLB*, as proposed by Moreira et al. (2012) (see Section 3.4.2), which have been found to produce the best results in average for the problem. Particularly in this method, we use the task prioritization rule $MaxPW^-$, that gives preference to tasks with larger minimum positional weight $pw_t^- = p_t^- + \sum_{t' \in F_t^*} p_{t'}^-$. We also use the worker prioritization rule *MinRLB*, that gives preference to workers with smaller *restricted lower bound* $\sum_{t \in T_u} p_t^-(W_u)/|W_u|$, where $p_t^-(W') = \min_{w \in W'} p_{tw}$ with the set $W_u \subseteq W$ corresponding to the unassigned workers and $T_u \subseteq T$ to the set of unassigned tasks of a partial assignment. Before computing *MinRLB* we apply to each partial solution the logic of the continuity constraints (4.10) and (4.11) to strengthen the bound. If tasks i and k have been assigned already to some worker w, we also assign all tasks succeeding i and preceding k to w. Similarly, if i has been assigned to w and some successor (predecessor) j of i is infeasible for w we set $p_{kw} = \infty$ for all successors (predecessors) k of j. The probabilistic beam search is shown in Algorithm 1.

Algorithm 1: Probabilistic beam search

input : A set of stations S, a candidate cycle time C, a beam width γ and a beam factor f.
output: A valid assignment or "failed" if no valid assignment could be found.

$B \leftarrow \{\emptyset\}$; /* current set of partial assignments */
for $k \in S$ **do**
  $B' \leftarrow \emptyset$;
  for $s \in B$ **do**
    for f *times* **do**
      for *all unassigned workers* $w \in W$ **do**
        $s' \leftarrow s$ concatenated with a new empty station k;
        while *there are available tasks P that do not overload the current station* **do**
          select a task $t \in P$ with probability proportional to $MaxPW^-(t)$;
          assign t to station k in s';
        end
        if *all tasks in T are assigned in* s' **then return** *Solution* s' ;
        else if $|B'| < \gamma$ **then** $B' \leftarrow B' \cup \{s'\}$;
        else
          $o \leftarrow argmin\{MinRLB(o') \mid o' \in B'\}$;
          if $MinRLB(s') > MinRLB(o)$ **then** $B' \leftarrow B' \cup \{s'\} \setminus \{o\}$;
        end
      end
    end
  end
  $B \leftarrow B'$;
  return *"failed"*;
end

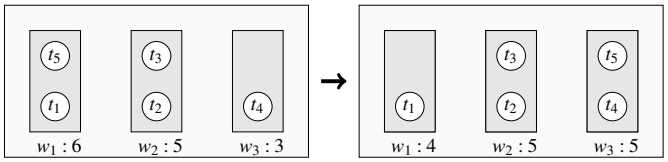


Figure 4.5 – Example of a shift operation. Task t_5 is moved from worker w_1 to worker w_3.

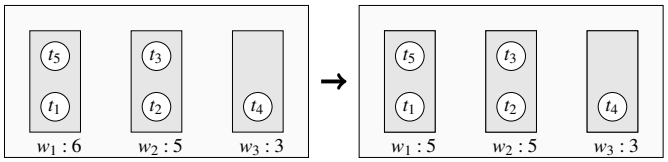


Figure 4.6 – Example of a swap operation. Task t_5 is swapped with task t_3.

4.3.2 Improvement by local search

A local search is applied to the results found by the interval search method. It focuses on *critical stations* whose load equals the cycle time of the current assignment. It tries to remove tasks from a critical station in order to reduce the cycle time. Since there can be multiple critical stations, a move is considered successful if it reduces the number of critical stations. The local search applies the following four types of moves, until the assignment cannot be improved any more.

1. A *shift* of a task from a critical station to another station, considering the precedence constraints.
2. A *swap* of two tasks. At least one of the tasks must be on a critical station, also considering the precedence constraints.
3. A sequence of two shift moves. Here the first shift move is allowed to produce a worse result than the initial assignment, also considering the precedence constraints.
4. A swap of workers between two stations without reassigning the tasks.

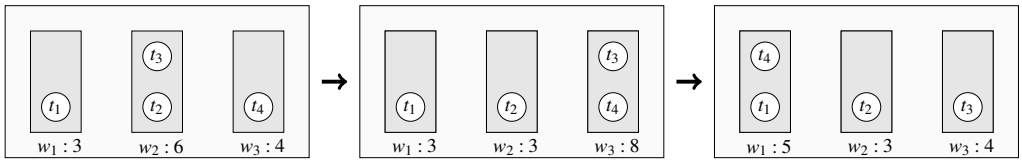


Figure 4.7 – Example of a two-shifts operation. Task t_3 is shifted to worker w_3 and then task t_4 is shifted to worker w_1.

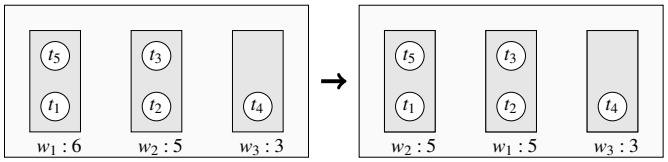


Figure 4.8 – Example of a workers swap operation. Worker w_1 is swapped with worker w_2.

Table 4.3 – Parameters of the iterative probabilistic beam search used in the computational experiments.

Beam width w	125
Branching factor f	5
Cycle time reduction for interval search p	0.95
Minimum search time t_{min} (s)	6
Maximum search time t_{max} (s)	900
Maximum number of interval searches r	20

4.3.3 Computational Results

We compare IPBS with three state of the art heuristic methods for the ALWABP-2, namely the hybrid genetic algorithm (HGA) of Moreira et al. (2012), the iterated beam search (IBS) of Blum and Miralles (2011), and the iterative genetic algorithm (IGA) of Mutlu, Polat and Supciller (2013). We additionally compare to the results obtained by the branch-and-bound-and-remember (BBR) method of Vilà and Pereira (2014), when stopped after sixty seconds.

In preliminary experiments we determined reasonable parameters for the probabilistic beam search as shown in Table 4.3. For the HGA and the IBS we compare in Table 4.4 the relative deviation from the current optimal value (Gap) and the computation time (t), in average for each group of instances and 20 replications per instance. We further report the average computation time to find the best value (t_b), and the average relative deviation of the best solution of the 20 replications (Gap_b). The total computation time of Blum and Miralles (2011) is always 120s more than the time to find the best value, and has been omitted from the table.

We can see that the problem can be considered well solved for a low number of tasks, since all three methods find the optimal solution with a few exceptions in less than ten seconds. In six instance groups the IPBS terminates in less than the minimum search time, since the solution was provably optimal. For instances with a high number of tasks, IBS produces better solutions for more workers, while the HGA is better on less workers. IPBS always achieves better results than both methods (confirmed by a Wilcoxon signed rank test with $p < 0.01$). This holds for the averages as well as the best found solutions (except the first instance group of Wee-Mag, where the best solution of IPBS is slightly worse than that of the HGA). IPBS is also very robust in the sense that the difference between average and best relative deviations is the smallest of the three methods. In average over all instances, its solutions are 1.9% over the optimal values.

To compare execution times, we have to consider that the results have been obtained on

Table 4.4 – Comparison of the proposed heuristic with a hybrid genetic algorithm (MOREIRA et al., 2012) and an iterated beam search (BLUM; MIRALLES, 2011).

				HGA				IBS			IPBS			
Instance	$\|W\|$	Var	Inf	t(s)	t_b(s)	Gap	Gap_b	t_b(s)	Gap	Gap_b	t(s)	t_b(s)	Gap	Gap_b
Roszieg	4	L	10%	3.3	0.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0
			20%	4.5	0.0	0.1	0.0	0.1	0.0	0.0	5.4	0.0	0.0	0.0
		H	10%	4.0	0.0	0.0	0.0	0.1	0.0	0.0	6.0	0.0	0.0	0.0
			20%	3.4	0.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0
	6	L	10%	3.6	0.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0
			20%	4.0	0.1	1.1	1.0	0.0	0.0	0.0	6.0	0.1	0.0	0.0
		H	10%	4.5	0.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0
			20%	4.5	0.1	0.0	0.0	0.0	0.0	0.0	6.0	0.1	0.0	0.0
Heskia	4	L	10%	6.9	0.2	0.0	0.0	8.2	0.0	0.0	6.0	0.1	0.0	0.0
			20%	9.3	0.3	0.1	0.1	3.0	0.0	0.0	6.0	0.1	0.0	0.0
		H	10%	9.2	0.3	0.0	0.0	5.6	0.0	0.0	6.0	0.1	0.0	0.0
			20%	9.5	0.5	0.3	0.0	5.2	0.0	0.0	6.0	0.2	0.0	0.0
	7	L	10%	8.0	0.2	0.5	0.0	1.1	0.0	0.0	5.4	0.2	0.0	0.0
			20%	7.4	0.3	0.6	0.0	2.5	0.0	0.0	4.3	0.2	0.5	0.3
		H	10%	6.6	0.2	0.3	0.0	1.7	0.0	0.0	2.5	0.2	0.0	0.0
			20%	9.2	1.5	0.7	0.0	2.5	0.0	0.0	3.7	0.2	0.0	0.0
Tonge	10	L	10%	205.7	34.4	5.9	2.4	86.4	6.7	4.8	56.2	19.7	1.9	0.9
			20%	241.2	34.9	4.2	2.4	92.2	4.6	3.3	58.9	14.8	2.3	1.0
		H	10%	391.0	98.6	4.4	1.8	160.1	5.5	3.5	89.2	26.8	1.4	0.9
			20%	347.5	56.9	4.3	2.7	171.4	4.7	3.8	91.0	21.4	2.2	1.2
	17	L	10%	296.9	74.0	11.3	7.8	88.0	8.2	4.7	61.2	26.2	2.8	1.9
			20%	300.0	67.1	14.3	9.2	70.5	11.5	8.6	60.5	19.4	6.9	5.6
		H	10%	446.7	129.1	10.7	5.7	124.3	7.9	5.4	96.7	31.8	3.0	2.3
			20%	469.5	105.3	10.4	7.6	156.4	8.9	5.8	107.7	31.5	4.5	3.2
Wee-Mag	11	L	10%	136.9	56.9	6.6	2.3	104.9	13.9	9.9	25.1	9.4	5.8	3.8
			20%	158.8	60.1	7.6	3.3	84.9	12.0	7.8	25.1	7.7	4.6	2.9
		H	10%	248.5	115.8	8.9	3.9	160.3	12.6	9.3	37.1	12.4	5.6	3.7
			20%	245.9	112.7	7.8	2.9	143.3	13.8	9.6	36.1	14.1	4.5	2.6
	19	L	10%	213.9	61.4	16.0	9.5	57.1	11.6	7.4	39.9	11.6	4.6	3.2
			20%	225.6	66.1	15.1	10.3	60.3	10.8	6.4	39.0	10.4	3.6	1.8
		H	10%	283.7	97.9	15.3	9.1	71.4	11.1	6.5	38.3	11.0	4.6	3.7
			20%	288.1	108.9	12.3	7.1	90.0	9.7	5.2	41.6	11.8	3.2	2.9
Total averages				143.7	40.1	4.9	2.7	54.7	4.7	3.1	31.0	8.8	1.8	1.2

different machines. The IBS was executed in a PC with a 2.2 GHz AMD64X2 4400 processor and 4GB of main memory, and the HGA in a PC with a Core 2 Duo 2.2 GHz processor in 3 GB of main memory. A conservative assumption is that their performance is within a factor of two of each other. Taking this into account, over all instances HGA and IBS have comparable computation times, and the IPBS is about a factor two faster. This holds for finding the best solution and also for the total computation time. (Remember that the total computation time of IBS is 120 s longer than the time to find the best solution.) The faster average computation times are mainly due to the instances with a high number of tasks, for which IPBS scales better. The best solutions are almost always found in less than 30 seconds.

For all heuristics, the computation time is significantly less for a low number of tasks, a low number of workers, and a low order strength. Similarly, the relative deviations are smaller for a low number of tasks and low order strength. However, the relative deviation does not depend significantly on the number of workers, except for the HGA, which produces better solution for a low number of workers. (These findings are confirmed by a Wilcoxon signed rank test at significance level $p < 0.01$.) For IBS and IPBS there is an interaction between the number of workers and the order strength: both produce better solutions for a low number of workers and a high order strength or *vice versa*.

For the IGA no detailed results are available, therefore we compare in Table 4.5 with the summarized values reported by Mutlu, Polat and Supciller (2013): the average cycle time (C), the average cycle time of the best found solution (C_b), and the average computation time to find the best value (t_b). The values are again averages for all groups of instances, but over only 10 replications for the IGA. The results for the IPBS are the same as in Table 4.4 but in absolute values. Note that this evaluation may mask large deviations in instances with low cycle times and overestimate small deviations for high cycle times. We further provide the averages of the optimal values (C).

As the other methods, the IGA solves the small instances optimally, but not the larger ones. Compared to the IPBS, its average performance is worse except for three groups of wee-mag with a low number of workers, where the average cycle time is about 0.2 lower. The comparison is similar for the best found values, where the IGA is better by 0.4 in a single group. In average over all large instances the IPBS produces a cycle time of about one unit less, and about one unit above the optimal values over all instances.

The execution times of the two methods are comparable. The results of Mutlu, Polat and Supciller (2013) have been obtained on a Intel Core 2 Duo T5750 processor running at 2.0 GHz, whose performance is within a factor of three from our machine. Taking this into account, the IPBS find the best value about 50% faster.

Finally, we compare the results of IPBS with the recent results obtained by the BBR of Vilà and Pereira (2014). BBR obtains better results than IPBS for all instance groups in comparable time (60 seconds). Even with a relative deviation from the optimal values of only 1.9% for IPBS, BBR frequently obtains a cycle time which is one or two units lower. This can be explained by

Table 4.5 – Comparison of the proposed heuristic with an iterated genetic algorithm (MUTLU; POLAT; SUPCILLER, 2013).

				IGA			IPBS		
Instance	$\|W\|$	Var	Inf	t_b(s)	C	C_b	t_b(s)	C	C_b
Roszieg	4	L	10%	0.3	20.1	20.1	0.0	20.1	20.1
			20%	0.3	31.5	31.5	0.0	31.5	31.5
		H	10%	0.3	28.1	28.1	0.0	28.1	28.1
			20%	0.2	28.0	28.0	0.0	28.0	28.0
	6	L	10%	0.5	9.7	9.7	0.0	9.7	9.7
			20%	0.5	11.0	11.0	0.1	11.0	11.0
		H	10%	0.5	16.0	16.0	0.0	16.0	16.0
			20%	0.5	15.1	15.1	0.0	15.1	15.1
Heskia	4	L	10%	0.3	102.3	102.3	0.1	102.3	102.3
			20%	0.3	122.6	122.6	0.1	122.6	122.6
		H	10%	0.3	172.5	172.5	0.1	172.5	172.5
			20%	0.2	171.3	171.2	0.2	171.3	171.2
	7	L	10%	0.5	34.9	34.9	0.2	34.9	34.9
			20%	0.5	42.6	42.6	0.2	42.8	42.7
		H	10%	0.5	75.2	75.2	0.1	75.2	75.2
			20%	0.5	67.2	67.2	0.2	67.2	67.2
Tonge	10	L	10%	47.4	94.1	93.0	19.7	92.2	91.3
			20%	40.5	110.2	109.3	14.8	109.1	107.8
		H	10%	70.8	165.2	162.4	26.8	161.7	160.8
			20%	59.4	170.1	168.4	21.4	167.5	165.9
	17	L	10%	78.0	33.1	33.1	26.2	32.5	32.2
			20%	68.4	40.4	40.1	19.4	39.4	38.9
		H	10%	68.1	66.4	66.4	31.8	64.9	64.5
			20%	78.0	64.8	64.6	31.5	63.9	63.1
Wee-Mag	11	L	10%	65.7	27.4	26.7	9.4	27.6	27.1
			20%	61.8	32.7	32.3	7.7	32.6	32.1
		H	10%	92.7	48.2	47.6	12.4	48.4	47.5
			20%	81.9	46.0	45.8	14.2	46.2	45.4
	19	L	10%	67.2	10.4	10.3	11.6	10.0	9.9
			20%	67.2	12.1	12.1	10.4	11.6	11.4
		H	10%	68.1	18.5	18.2	11.0	17.9	17.7
			20%	77.4	18.4	18.0	11.8	17.7	17.7
Averages				34.3	59.6	59.3	8.8	59.1	58.8

the enumerative approach of BBR compared to the randomized generation of station loads in IPBS. BBR achieves these results by storing all partial solutions, and thus IPBS may still be a competitive method on larger instances.

In summary, the results show that IPBS can compete with and often outperforms existing methods in solution quality as well as computation time. The enumerative approach of Vilà and Pereira (2014), however, finds better solutions in about the same time. The difference to the other methods is smallest for the large instances with a low order strength and a low number of workers. IPBS in general is very robust over the entire set of instances.

4.4 A Task-Oriented Branch-and-Bound Algorithm

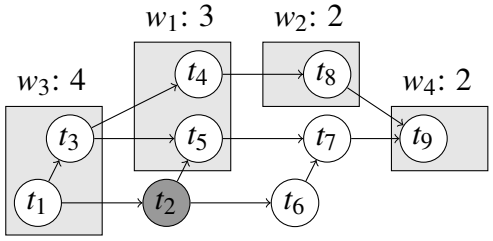
In this chapter we propose a branch-and-bound algorithm for ALWABP-2 using the bounds and the heuristic presented in the previous chapters. Our method uses a depth-first search to find solutions. In branch-and-bound algorithms for assembly line balancing three branching strategies exist. The *station-oriented*, the *task-oriented* and the *worker-oriented* method. They were explained in Chapter 3. The most effective methods for SALBP use station-oriented branching. However, for the ALWABP the additional worker selection substantially increases the branching factor of the station-oriented approach. A *worker-oriented* strategy, on the other hand, has to consider much more station loads, since all subsets of unassigned tasks which satisfy the continuity constraints (4.10) are candidates to be assigned to a worker. Therefore, we propose to use a task-oriented branching strategy.

At each node the method processes the partial solution, evaluates the lower bound according to the current information, applies pruning techniques to avoid branches that not lead to good solutions and, after that, selects a task t. The method assumes that the task that produces the smallest number of branches should be selected first. With this, few branches are generated at the beginning and at each branch, a better task selection can be made. So, for each task t of the set of unassigned tasks (U) in the current node of the branch-and-bound tree, the number of assignable workers $|A_t|$ is defined, according to Section 4.4.1, and then we select a task $t^* = \operatorname{argmin}\{|A_t| \mid t \in U\}$. In case of ties, the task with the smallest lower bound is selected. The lower bound of a task is given by the minimum lower bound found after assigning the task to each of the assignable workers. For example, given the partial solution in Figure 4.9, task t_2 could be assigned to workers w_1 and w_3, while the only invalid worker for tasks t_6 and t_7 is w_3.

After the task has been selected, it must be assigned to each of the possible workers (A_{t^*}) in a separate branch. Since the branches with a smaller lower bound tend to produce the best final solutions, we explore them first. In Figure 4.9, the task t_2 could be assigned for both w_1 and w_3. If it is assigned to worker w_1, the partial solution has cycle time 8. If it is assigned to w_3, it has partial solution with cycle time 6. Therefore, task t_2 will be first assigned to w_3 and then to w_1.

The lower bound used for the task selection and prioritization of the branches is the LC_1, as well as the cycle time for the stations already assigned. In both cases the lower bound method

Tasks Precedence Graph



Task Execution Times

p_{tw}	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
w_1	4	3	2	1	2	6	3	∞	4
w_2	∞	5	6	5	3	2	2	2	2
w_3	3	4	1	∞	3	∞	2	3	∞
w_4	5	3	∞	2	2	2	3	2	2

Figure 4.9 – Example of task selection for a partial solution for an ALWABP-2 instance with nine tasks and four workers. Task t_2 is selected.

must be applied $O(TW)$ times and so a simple and fast lower bound is needed. When the decisions are made and a new branch is created, the lower bound is updated with the results of $LC_2, LC_3, \overline{L}_1^a$. These bounds are among the fastest and produce efficient lower bounds when applied to partial solutions. The lower bounds M_{W3} and $\overline{L}_2$ are too slow to be applied during the search, although they obtain the best bounds.

The proposed task-oriented method executes the recursive procedure shown in Algorithm 2, with an initial upper bound given by the heuristic presented in Section 4.3. At each new node it applies the lower bounds (line 8). When a complete solution has been found, the algorithm updates the incumbent (line 2). Otherwise, it selects an unassigned task t (line 5) and assigns it to all feasible workers (loop in lines 6–12), when the reduction rules (line 7) have not been successful.

4.4.1 Valid assignments

To define if a task t can be assigned to a worker w the branch-and-bound algorithm applies a similar logic to that of the continuity constraints of models M_{W2} and M_{W3} (see Section 4.1). Namely, the algorithm maintains a directed graph H on the set of workers to verify efficiently if the precedence constraints are satisfied. It contains an edge (w, w') if there is some task t assigned to w and another task t_0 assigned to w_0, such that (t, t'). The graph H also contains all resulting transitive edges. For a valid assignment of tasks, H must be acyclic. If this is the case, any topological sorting defines a valid assignment of workers to stations.

Three procedures are defined for the assignment of tasks to workers. Since the assignment of task t to worker w must be verified many times when processing a node, procedure assignmentIsValid(t, w) applies the verification but does not consider the violation of transitive

Algorithm 2: branchTasks(llb, A)

Input : An upper bound gub, a set $A \subseteq T$ of assigned tasks, and a local lower bound llb.

```
1  if A = T then
2  |  if llb < gub then gub ← llb;
3  |  return
4  end
5  select a task t ∈ T \ A ;
6  foreach w ∈ W | assignmentIsValid(t, w) do
7  |  apply reduction rules;
8  |  newllb ← lower bound with new assignment (t, w);
9  |  if newllb < gub then
10 |  |  setAssignment(t, w);
11 |  |  branchTasks(newllb, A ∪ {t});
12 |  |  unsetAssignment(t, w);
13 |  end
14 end
```

dependencies in H in line 5, but only the creation of an immediate cyclic worker dependency, which results from inserting an edge (w, w') for which (w', w) is already present. This can be tested in time $O(|P_t| + |F_t|)$. The other two procedures are applied when the verification and some pruning were already applied. The first, setAssignment(t, w), assigns a task t to a worker w, and inserts the induced arcs in H as well as the arcs needed to produce the transitive closure of H. This operation can be implemented in amortized time $O(|W|)$ using a data structure proposed by Italiano (1986). Finally, when the entire branch has been explored, the partial solution is returned to its original state by applying the operation unsetAssignment(t, w).

4.4.2 Reduction rules

After a task t has been assigned to a worker w, and before branching, we apply several more costly reduction rules to strengthen the lower bounds (line 7). First, to improve the calculation of the lower bounds, we can set all $p_{tw'} = \infty$ for any $w' \neq w$. This will guarantee that the remaining workers are not used for task t.

The continuity constraints (4.10) and (4.11) of model M_{W3} can be analogously applied to prune the branch-and-bound tree. If there is another task t' already assigned to w and there is a set of tasks A between t and t' in the precedence graph G, all tasks in A must be assigned to w as well, so we can set $p_{aw'} = \infty$ for every $a \in A$ and $w' \in W \setminus \{w\}$. If there is a task $t' \in F_t^*$ assigned to another worker $w' \neq w$, it is possible to assume that every task $t'' \in F_{t'}^*$ cannot be assigned to w, and so we set $p_{t''w} = \infty$ for every $t'' \in F_{t'}^*$. This rule can be applied in both directions of the graph.

Generalizing this, we can assume that an assignment of task t' to worker w is invalid if the time of this task, the task t and all tasks between them, sum the current upper bound or more than that. Then we can set $p_{t'w} = \infty$ if the the total execution time $p_{tw} + p_{t'w} + \sum_{u \in i(t,t')} p_{uw}$ is

greater or equal than the upper bound for tasks $i(t,t') = (P_t^* \cap F_{t'}^*) \cup (F_t^* \cap P_{t'}^*)$ between t and t'. The rules can be applied iteratively until no rule changes the times of tasks.

4.4.3 Computational Results for the Branch-and-Bound Algorithm

We evaluated the branch-and-bound algorithm presented on the 320 test instances used in the literature (please refer to Section 4.1.4). IPBS was used to produce an initial heuristic solution. It was made deterministic by fixing a random seed of 42 and configured with a minimum search time of 0 s and a maximum search time of $|T||W|/10$ s. During the search the number of iterations of the ascent direction method to compute L_1 has been limited to 50, and the number of iterations for the subgradient optimization to compute L_2 to 20. The branch-and-bound algorithm was implemented in C++, and compiled with the GNU C compiler 4.6.3 with maximum optimization. For lower bounds using linear relaxation, CPLEX 12.3 was used. The experiments were executed on a PC with a 2.8 GHz Core i7 930 processor and 12 GB of main memory, running a 64-bit Ubuntu Linux. All tests used only one core.

The first branch-and-bound algorithm in the literature, proposed by Miralles et al. (2008b) for the ALWABP-2, has been found inferior to model M_{W1} by Chaves, Lorena and Miralles (2009) in tests with CPLEX (version 10.1). We therefore limit our comparison to the MIP models. We first compare our approach to CPLEX on the best model M_{W3} on the instances with a low number of tasks in Table 4.6. In Table 4.7 we then present the results of the branch-and-bound algorithm with a time limit of one hour on the larger instances. CPLEX is not able to solve any of the models on the instances with a high number of tasks within this time limit.

Table 4.6 shows the average solving time and the average number of nodes in the branch-and-bound tree for all instance groups with a low number of workers. On these instances both methods have a similar performance, solving all instances in a few seconds, and are even competitive with the heuristic methods. In most cases the branch-and-bound algorithm needs fewer nodes than CPLEX, except for five groups with a low number of workers. Computation times are also comparable, although the time of the branch-and-bound algorithm is dominated by the initial heuristic.

Table 4.7 shows the results of our branch-and-bound algorithm on the larger instances, and compares them to the results of Vilà and Pereira (2014). We report the number of optimal solutions found (Opt) and the number of solutions proven to be optimal ($Prov$), the average computation time (t), the average relative deviation from the optimal value (Gap), and the average cycle time for each group of instances (C). Vilà and Pereira (2014) do not report average computation times.

In about 75% of the instances the optimal solution was found, and about 62% of the solutions could be proven to be optimal within the time limit. All instances Tonge except four instances with a high order strength were solved. The average relative deviation over all 320 instances is 0.7%, about one-third of the average of the best heuristic.

Table 4.6 – Comparison of model M_{W3} to the branch-and-bound algorithm on instances with a low number of workers.

				Model M_{W3}		**B&B**	
Instance	$\|W\|$	Var	Inf	t (s)	Nodes	t (s)	Nodes
Roszieg	4	L	10%	0.7	37.8	0.1	32.7
			20%	0.4	11.7	0.1	15.5
		H	10%	1.5	58.6	0.2	40.9
			20%	0.8	53.8	0.1	35.9
	6	L	10%	4.6	249.9	0.4	120.8
			20%	2.3	168.7	0.3	78.0
		H	10%	6.3	389.0	0.5	189.3
			20%	4.5	281.5	0.4	131.0
Heskia	4	L	10%	0.3	29.8	0.2	34.7
			20%	0.2	37.5	0.2	39.8
		H	10%	0.3	49.0	0.2	52.5
			20%	0.2	45.5	0.2	59.5
	7	L	10%	1.0	86.8	1.2	15.0
			20%	1.0	102.4	1.4	20.1
		H	10%	1.1	115.4	1.2	13.9
			20%	1.4	151.7	1.4	18.1
Averages				1.7	116.8	0.5	56.1

Table 4.7 – Results of the B&B algorithm on instances with a high number of workers.

Instance	$\|W\|$	Var	Inf	**BBR**				**B&B**				
				Opt.	Prov.	*Gap*	*C*	Opt.	Prov.	*t*(s)	*Gap*	*C*
Tonge	10	L	10%	10	10	0.0	90.6	10	10	165.4	0.0	90.6
			20%	10	10	0.0	106.7	10	10	134.2	0.0	106.7
		H	10%	10	10	0.0	159.3	10	10	362.1	0.0	159.3
			20%	10	10	0.0	163.9	10	10	233.9	0.0	163.9
	17	L	10%	9	0	0.3	31.7	10	10	789.9	0.0	31.6
			20%	10	1	0.0	36.9	10	9	822.5	0.0	36.9
		H	10%	10	0	0.0	63.2	9	7	1438.2	0.3	63.4
			20%	10	0	0.0	61.2	10	10	1294.8	0.0	61.2
Wee-mag	11	L	10%	10	10	0.0	26.8	5	2	3316.6	2.6	26.8
			20%	10	10	0.0	32.2	3	1	3534.2	3.2	32.2
		H	10%	10	10	0.0	47.5	3	1	3295.5	3.6	47.5
			20%	10	10	0.0	45.2	4	3	2929.8	1.9	45.2
	19	L	10%	10	9	0.0	9.9	7	3	3504.6	3.2	9.9
			20%	10	7	0.0	11.4	8	4	2727.8	1.9	11.4
		H	10%	9	8	0.6	17.6	6	6	2251.9	3.0	17.6
			20%	10	5	0.0	17.6	6	4	2677.1	2.3	17.6
Totals/Averages				158	110	0.9		111	95	1896.0	1.20	

As expected, the solution times are higher than those of the heuristic methods but for the instances with a high order strength only about an order of magnitude, in average. The solving time depends mainly on the number of tasks, the number of workers, and the order strength (as confirmed by a Kruskal Wallis test followed by Wilcoxon signed rank post hoc tests at significance level $p < 0.01$). The instances with a high order strength or a low number of workers are easier to solve, because the reduction rules are more effective.

We further investigated the dependence of the algorithm on the initial solutions found by IPBS, by repeating the experiments with solutions obtained by a single run of IPBS with a beam factor of 1, which is equivalent to a simple station-oriented constructive heuristic. This substantially increases the relative deviation from the optimal value of the initial solutions from 1.8% to 17.8%. The results obtained by the branch-and-bound algorithm with initial solutions from IPBS are, as expected, better than those starting from the simple heuristic. The final relative deviation increases from 0.7% to 2.9%, and the number of optimal and provably optimal solutions reduces to 98 and 82, respectively. This shows that good initial upper bounds help to find good solution and, in particular, prove their optimality.

Compared to the BBR of Vilà and Pereira (2014) we find that our method can prove optimality of a comparable number of instances. The results of BBR have been obtained on a PC with a 3.2 GHz Intel Core i5 processor which is slightly faster than our machine. Indeed, if we run our algorithm for 2 hours, we can prove optimality of 114 instances, i.e., the number of instances provably optimal is about the same, within reasonable time limits. The solution quality of BBR is better, reaching the optimal value in all except two instances. This confirms the findings of the comparison of the heuristics, that BBR is able to find good solutions fast. The methods are complementary since BBR is station-oriented and strongly exploits memory-based dominance rules, while our method is task-oriented, and focuses on good lower bounds. As a result, BBR proves more instances with a low order strength and short processing times optimal, which typically permit lots of equivalent solutions and have weak lower bounds, while our method performs better for high order strengths. In particular, the 40 instances with high order strength and a high number of workers our method proves 36 solutions optimal and BBR only 1. Note that in this case BBR is limited by the size of the memory, which exhausts before reaching the time limit.

4.4.4 Parallelization

An approach to decrease the time needed to execute the method is to parallelize the method. The technique proposed for the parallelization modifies the depth first search and for each new branch has two options: First, at each branch, a new thread is created, until all processors are occupied. When all processors are occupied, the new branches must be executed on the same processor as its parent.

Each processor executes until all nodes of its branch have been explored. Then, some branch

will have to be transferred to the now-free processor. When a processor is freed, the remaining processors will provide each a new branch to the empty processor. The simplest solution selects the first branch and executes it on the free processor. However, a better approach is possible. Since branches with better lower bounds are more likely to produce good solutions, we chose to explore the branch with the smallest lower bound in an empty processor.

To implement parallelism we start P processors that wait for new branches. Every time a new branch is created, the method evaluates if there is any empty processor. If there is such a processor and the current branch has the best lower bound among all branches being executed, then the branch is sent to the empty processor. Every time one of this branches ends, its processor begins to wait for the remaining processors to produce branches.

4.4.5 Computational Results for the Parallel Branch-and-Bound Algorithm

The Parallel Task-Oriented Branch-and-Bound for the ALWABP (PTOBB) was implemented using C++ and compiled in a GNU C compiler 4.63 using the flag -O3 for maximum optimization. The threads library from Boost was used for parallelization. The experiments were run on a PC with a 3.60 GHz AMD FX-8150 8-Core Zambezi processor and 32 GB of main memory, running a 64-bit Ubuntu Linux. The method was also applied to the 320 instances from the literature (see Section 4.1.4).

4.4.5.1 Impact of the number of processors

First of all, we will analyze the effect of multiple processors in PTOBB. For the small instances Roszieg and Heskia all instances are solved and are proven to be optimal in a few seconds. Table 4.8 shows the number of nodes needed to prove a solution optimal (column "Nodes"), the time needed ("t(c)"), and the time needed per node ("ms/node").

For Roszieg instances, the number of nodes visited in each of the tests equal for all cases. The time varies only in the two groups with a high number of workers and high time variability. The time needed to execute some instances of these two groups is slightly greater than one second for one processor. For two or more processors all Roszieg instances are solved in less than a second. For the Heskia instances, however, there is a variation of time between the one, two and four processor cases. In these cases the number of nodes, as expected, varies. Since the branching tree may be different according to the current incumbent solution, the number of processors affects the number of nodes. The time needed to prove optimality decreases by 37.05% from one to two processors and 22.81% from two to four processors and so does the time needed per node. The eight processor case has very similar values to the four processor case. It indicates that the speed up is limited to four processors. This tests show the efficacy of using multiple processors until a limit of four for small instances.

Table 4.9 shows the number of solutions proven optimal in an hour ("Prov.") and the gap of

Table 4.8 – Multiple processors results for the small instances: Roszieg and Heskia.

				Number of Processors											
				1			2			4			8		
Instance	$\|W\|$	Var	Inf	Nodes	t(s)	ms/node	Nodes	t(s)	ms/node	Nodes	t(s)	ms/node	Nodes	t(s)	ms/node
Roszieg	4	L	10%	32.0	0.0	0.00	32.0	0.0	0.00	32.0	0.0	0.00	32.0	0.0	0.00
			20%	16.5	0.0	0.00	16.5	0.0	0.00	16.5	0.0	0.00	16.5	0.0	0.00
		H	10%	41.8	0.0	0.00	41.8	0.0	0.00	41.8	0.0	0.00	41.8	0.0	0.00
			20%	37.7	0.0	0.00	37.7	0.0	0.00	37.7	0.0	0.00	37.7	0.0	0.00
	6	L	10%	119.9	0.0	0.00	119.9	0.0	0.00	119.9	0.0	0.00	119.9	0.0	0.00
			20%	79.9	0.0	0.00	79.9	0.0	0.00	79.9	0.0	0.00	79.9	0.0	0.00
		H	10%	195.8	0.3	1.53	195.8	0.0	0.00	195.8	0.0	0.00	195.8	0.0	0.00
			20%	130.2	0.1	0.77	130.2	0.0	0.00	130.2	0.0	0.00	130.2	0.0	0.00
Heskia	4	L	10%	49.6	0.0	0.00	51.1	0.0	0.00	51.1	0.0	0.00	51.1	0.0	0.00
			20%	40.1	0.0	0.00	40.4	0.0	0.00	40.3	0.0	0.00	40.4	0.0	0.00
		H	10%	48.8	0.1	2.05	48.8	0.0	0.00	48.8	0.0	0.00	48.8	0.0	0.00
			20%	57.4	0.0	0.00	57.6	0.0	0.00	59.1	0.0	0.00	59.0	0.0	0.00
	7	L	10%	15.8	1.6	101.27	15.8	1.1	69.62	15.8	0.9	56.96	15.8	0.9	56.96
			20%	15.6	2.1	134.62	15.6	1.4	89.74	15.6	1.0	64.10	15.6	1.0	64.10
		H	10%	14.7	2.2	149.66	14.7	1.3	88.44	14.7	1.0	68.03	14.7	1.0	68.03
			20%	19.3	2.5	129.53	19.3	1.5	77.72	19.3	1.2	62.18	19.3	1.2	62.18
Averages				57.2	0.6	32.46	57.3	0.3	20.34	57.4	0.3	14.68	57.4	0.3	15.70

Table 4.9 – Multiple processors solutions for the bigger instances: Tonge and Wee-mag.

			-	Number of Processors							
				1		2		4		8	
Instance	$\|W\|$	Var	Inf	Prov.	Gap	Prov.	Gap	Prov.	Gap	Prov.	Gap
Tonge	10	L	10%	10	0.0	10	0.0	10	0.0	10	0.0
			20%	10	0.0	10	0.0	10	0.0	10	0.0
		H	10%	10	0.0	10	0.0	10	0.0	10	0.0
			20%	10	0.0	10	0.0	10	0.0	10	0.0
	17	L	10%	9	0.0	10	0.0	10	0.0	10	0.0
			20%	9	0.6	9	0.0	10	0.0	10	0.0
		H	10%	7	0.6	6	0.5	9	0.0	10	0.0
			20%	9	0.0	10	0.0	10	0.0	10	0.0
Wee-Mag	11	L	10%	3	1.8	2	1.4	4	1.1	6	0.3
			20%	1	2.0	3	2.0	5	2.0	6	0.0
		H	10%	2	2.1	2	1.9	4	1.7	6	1.3
			20%	5	1.5	5	0.9	5	0.9	8	0.0
	19	L	10%	1	3.2	3	3.2	7	2.1	9	1.1
			20%	5	0.9	6	0.9	7	0.9	7	0.9
		H	10%	5	1.9	7	0.6	9	0.6	10	0.0
			20%	5	1.7	5	1.7	7	1.1	8	0.6
Averages				101	1.0	108	0.8	127	0.6	140	0.3

-

the solutions to the best known value in literature (“Gap”).

We see in Table 4.9 an increase of the number of solutions proven optimal and also a decrease of the gap to the best known values according to the number of processors used. Table 4.10 explains this behavior. The columns are the number of nodes visited by the program (“Nodes”), the time needed to prove optimality of a solution (“t(c)”), one hour if the program reaches the time limit before finding an optimal solution) and the time executing a node ($ms/node$).

Table 4.10 – Multiple processors results for the large instances: Tonge and Wee-Mag.

				Number of Processors											
				1			2			4			8		
Instance	$\|W\|$	Var	Inf	Nodes	t(s)	ms/n	Nodes	t(s)	ms/n	Nodes	t(s)	ms/n	Nodes	t(s)	ms/n
Tonge	10	L	10%	238756.1	242.1	1.01	207048.3	146.4	0.71	181562.9	103.7	0.57	186121.2	86.8	0.47
			20%	137843.0	161.2	1.17	148674.6	119.3	0.80	155009.9	97.0	0.63	144443.8	84.1	0.58
		H	10%	399743.5	392.0	0.98	381224.3	251.6	0.66	380483.7	151.3	0.40	394189.2	117.4	0.30
			20%	248597.5	269.4	1.08	233062.6	157.0	0.67	235285.2	121.3	0.52	233930.3	99.3	0.42
	17	L	10%	660634.5	1034.8	1.57	557170.2	521.4	0.94	608507.7	306.3	0.50	614217.8	203.1	0.33
			20%	632140.9	997.4	1.58	995805.0	773.0	0.78	1115603.9	491.0	0.44	1170254.5	301.7	0.26
		H	10%	996078.7	1595.4	1.60	1768098.1	1651.8	0.93	2181709.8	976.7	0.45	2327228.5	592.9	0.25
			20%	1017553.0	1458.0	1.43	1133200.3	925.6	0.82	1067227.8	514.2	0.48	1087780.7	328.7	0.30
Wee-Mag	11	L	10%	4254359.8	3005.6	0.71	8002007.9	3002.0	0.38	13489828.1	2499.3	0.19	22287226.0	2151.8	0.10
			20%	5138178.7	3540.2	0.69	8107863.9	3088.0	0.38	13998188.6	2658.1	0.19	20194857.4	1924.4	0.10
		H	10%	4357923.3	3024.6	0.69	8360791.3	2969.9	0.36	14361423.6	2687.0	0.19	22720737.7	2182.1	0.10
			20%	3645888.1	2681.3	0.74	6479562.4	2339.8	0.36	10811350.2	2082.8	0.19	18443534.2	1819.3	0.10
	19	L	10%	4019388.5	3258.6	0.81	6352264.0	3057.2	0.48	10071860.9	2466.8	0.24	14622372.6	1824.2	0.12
			20%	3172054.8	2572.2	0.81	5571889.8	2331.5	0.42	7823388.6	1739.9	0.22	13077777.0	1521.6	0.12
		H	10%	3087639.0	2402.9	0.78	3767251.9	1765.6	0.47	5664634.1	1342.2	0.24	7559162.1	936.4	0.12
			20%	3360713.1	2505.3	0.75	5813440.0	2373.5	0.41	7901963.9	1833.6	0.23	10974821.1	1364.3	0.12
Averages				2210468.0	1821.3	1.02	3617460.0	1592.1	0.60	5628002.0	1254.5	0.35	8502416.0	971.1	0.24

First, it is possible to identify that the time needed to prove a solution to be optimal always decreases when the number of processors increases. Yet, the number of nodes increases with the number of processors. The format of the branching tree depends on the incumbent solution found until the nodes are branched. In particular, in cases with wide and shallow branching trees, a good incumbent solution accelerates the process of proving a solution optimal. This happens because in this type of tree bad decisions taken early affect a larger part of the tree. With multiple processors, we are searching bad branches early, without good incumbent solutions. So, especially in the case of Wee-Mag instances, there is an increase of visited nodes. For the Tonge instances, the behavior of the number of nodes is inconclusive, since there are both increases and decreases of the number of nodes when the number of processors is greater. It is also possible to perceive that the time per node is halved when the number of processors is doubled, showing that the nodes are well distributed among processors.

4.4.5.2 Comparison with related work

Here we compare the parallel branch-and-bound algorithm with the two sequential state-of-the-art exact methods: the station-oriented branch, bound-and-remember (BBR) Vilà and Pereira (2014) and the single-threaded task-oriented branch-and-bound (TOBB). Table 4.11 shows the results for our method (PTOBB) and the two methods in literature. The columns are similar to those of Section 4.4.5.1. Column "t(c)" specifies the time needed to execute a task (bounded to an hour if the instance lasted more than an hour). The number of solutions proven to be optimal in an hour are shown in column "Prov.". The gap between the solution found after an hour and the best known value in literature are shown in column "Gap". Since no running times were presented for the BBR method, only the times of TOBB and PTOBB are shown.

The BBR was executed on a 3.2 GHz Core i5 Processor and the TOBB on a 2.8 GHz Core i7 processor. The performance of the three machines differs from one another by a factor of at most 1.5. The table shows that PTOBB produces solutions in less time than the original TOBB, even considering this factor. Also, the PTOBB solves all Tonge instances in an hour. This is not the case for the other methods. BBR, on the other hand, solves only 41 Tonge instances and TOBB solves 76. For the Wee-Mag instances, the BBR solves more instances to optimality but for the instances with a high number of workers the PTOBB produces better results. Overall PTOBB is the method that proves more solutions optimal (241). The Tonge instances are solved by the PTOBB, but for the Wee-Mag case, some instances have a small gap. The gap of PTOBB is much smaller than the original TOBB but is slightly larger than that of the BBR.

Table 4.11 – Comparison with state-of-the-art methods

				Methods							
				BBR		TOBB			PTOBB		
Instance	$\|W\|$	Var	Inf	Prov.	Gap	Prov.	Gap	t(s)	Prov.	Gap	t(s)
Tonge	10	L	10%	10	0.0	10	0.0	165.4	10	0.0	86.8
			20%	10	0.0	10	0.0	134.2	10	0.0	84.1
		H	10%	10	0.0	10	0.0	362.1	10	0.0	117.4
			20%	10	0.0	10	0.0	233.9	10	0.0	99.3
	17	L	10%	0	0.3	10	0.0	789.9	10	0.0	203.1
			20%	1	0.0	9	0.0	822.5	10	0.0	301.7
		H	10%	0	0.0	7	0.3	1438.2	10	0.0	592.9
			20%	0	0.0	10	0.0	1294.8	10	0.0	328.7
Wee-Mag	11	L	10%	10	0.0	2	2.6	3316.6	6	0.3	2151.8
			20%	10	0.0	1	3.2	3534.2	6	0.0	1924.4
		H	10%	10	0.0	1	3.6	3295.5	6	1.3	2182.1
			20%	10	0.0	3	1.9	2929.8	8	0.0	1819.3
	19	L	10%	9	0.0	3	3.2	3504.6	9	1.1	1824.2
			20%	7	0.0	4	1.9	2727.8	7	0.9	1521.6
		H	10%	8	0.6	6	3.0	2251.9	10	0.0	936.4
			20%	5	0.0	4	2.3	2677.1	8	0.6	1364.3
Averages				110	0.1	100	1.4	1842.4	140	0.3	971.1

5 SOLUTIONS FOR THE ROBOTIC ASSEMBLY LINE BALANCING PROBLEM

In the next sections we will present solution procedures for the RALBP-2. We introduce a mathematical formulation for the problem. We also investigate a novel lower bound, that 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, with a series of dominance rules adapted or created for RALBP. An iterative beam search using these same dominance rules and lower bounds is then shown. All the methods are evaluated by computational experiments using the known instances and a new set of instances that explore different characteristics of the problem.

5.1 Mathematical Formulation

The model M_{R1} (See Section 1.2) has quadratic constraints that make it impossible to solve in conventional solvers. Therefore we propose using a model M_{R2}, that avoids the quadratic functions of model M_{R1}. We also use the techniques proposed by (RITT; COSTA, 2015) to improve the precedence constraints. The resulting model M_{R2}, with notation described in Table 5.1, can then be defined by

$$M_{R2} = \textbf{minimize} \quad C, \tag{5.1}$$

$$\textbf{subject to} \quad \sum_{t \in A_s} p_{tr} x_{ts} \leq C + M_r(1 - y_{sr}), \qquad \forall s \in S, r \in R, \tag{5.2}$$

$$\sum_{r \in R} y_{sr} = 1, \qquad \forall s \in S, \tag{5.3}$$

$$\sum_{s \in I_t} x_{ts} = 1, \qquad \forall t \in T, \tag{5.4}$$

$$\sum_{k \in I_i | k \leq s} x_{ik} \geq \sum_{k \in I_j | k \leq s} x_{jk}, \qquad \forall i \in T, j \in F_i, s \in S, \tag{5.5}$$

$$x_{ts} \in \{0,1\}, \qquad \forall s \in S, t \in A_s, \tag{5.6}$$

$$y_{sr} \in \{0,1\}, \qquad \forall s \in S, r \in R. \tag{5.7}$$

The model minimizes the cycle time C (5.1), defined in constraint (5.2). Since the right side of constraint (5.2) 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 (5.3) and (5.4) ensure that each task will be performed and that each station will have a robot assigned to it. Constraint (5.5) 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_{R2} is linear and has $O(|T||S| + |S||R|)$ variables and $O(|S||R| + |T|^2|S|)$ constraints.

Table 5.1 – 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.

5.1.1 Computational Experiments

In the literature of the RALBP, only one set of 32 instances is provided (GAO et al., 2009). We call it Instance Set 1. It uses eight precedence graphs from the literature of the SALBP-1 (Roszieg, Gunther, Hahn, Tonge, Lutz3, Arc111, Barthol2 and Scholl) and four instances for each of the graphs. These four instances are 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 the 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 randomly chosen values: the difficulty of performing the task t ($d_t \in [100,200]$), the efficiency factor of the robot r ($e_r \in [1,2]$), and the specialization ability of the robot r to perform the task t ($c_{tr} \in [1,3]$). Given these three factors, we define $p_{tr} = d_t/(e_t c_{tr})$.

We first compare our model M_{R2} to the model M_{R1} 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 5.2.

In the table, column "Gap" 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_{R2} consistently produces smaller cycle times than model M_{R1} and it also proves more solutions to be optimal. Nine instances are proven optimal by model M_{R2} compared to only two instances by model M_{R1}. The performance of model M_{R2} 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 gaps are larger than instances with a smaller number of robots.

Table 5.2 – Comparison of MIP models M_{R1} (Mukund Nilakantan et al., 2015) and M_{R2} on instance set 1.

		M_{R1}		M_{R2}	
|T|	**|R|**	**Gap (%)**	**Time (s)**	**Gap (%)**	**Time (s)**
25	3	**0.00**	12.30	**0.00**	0.10
	4	**0.00**	1,254.11	**0.00**	0.15
	6	11.34	3,600.00	**0.00**	3.98
	9	60.55	3,600.00	**0.00**	22.01
35	4	2.93	3,600.00	**0.00**	0.15
	5	22.80	3,600.00	**0.00**	1.43
	7	36.32	3,600.00	**0.00**	351.72
	12	186.02	3,600.00	**5.38**	3,600.00
53	5	41.65	3,600.00	**0.00**	0.36
	7	21.20	3,600.00	**0.00**	9.27
	10	318.23	3,600.00	**0.00**	1,859.77
	14	561.19	3,600.00	**2.99**	3,600.00
70	7	142.27	3,600.00	**0.00**	1,332.32
	10	193.97	3,600.00	**3.88**	3,600.00
	14	331.76	3,600.00	**5.29**	3,600.00
	19	949.17	3,600.00	**7.50**	3,600.00
89	8	92.13	3,600.00	**0.46**	3,600.00
	12	461.77	3,600.00	**1.71**	3,600.00
	16	822.93	3,600.00	**3.42**	3,600.00
	21	1,021.29	3,600.00	**7.10**	3,600.00
111	9	291.63	3,600.00	**3.00**	3,600.00
	13	419.85	3,600.00	**8.09**	3,600.00
	17	809.05	3,600.00	**5.71**	3,600.00
	22	755.26	3,600.00	**15.13**	3,600.00
148	10	488.01	3,600.00	**2.40**	3,600.00
	14	718.98	3,600.00	**2.27**	3,600.00
	21	1,388.79	3,600.00	**12.11**	3,600.00
	29	1,748.68	3,600.00	**15.79**	3,600.00
297	19	610.27	3,600.00	**10.27**	3,600.00
	29	2,311.87	3,600.00	**15.77**	3,600.00
	38	1,718.62	3,600.00	**26.72**	3,600.00
	50	2,701.08	3,600.00	**0.20**	3,600.00
Avg.		**602.67**	**3414.60**	**7.41**	**2617.97**

5.2 Lower Bounds

The lower bound LC_1 (RUBINOVITZ; BUKCHIN; LENZ, 1993) relaxes all the precedence constraints of the problem. We generalize this method and improve the result of the lower bound by maintaining some of the precedence constraints in the lower bound calculation. In this method, any set of disjoint task chains T_c is selected and all the remaining task precedence constraints are relaxed. 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 select the set of task chains T_c, we iteratively select the longest chain in the graph until all tasks have been assigned to one of the chains. Initially, the longest chain of the graph is calculated, they form the first chain of tasks of the set T_c. The tasks of this chain are, then removed from the original graph and the next chain is calculated using the same procedure. The longest chain is always selected since this increases the chance that multiple tasks are assigned to the same station and the same robot has to perform them.

Consider the example shown in Figure 1.4. 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.

To calculate the smallest total task time of the instance, we sum the smallest total task time of all chains. The smallest total task time of a chain is given by $\sum_{t \in T} p_{t(r_t)}$, where r_t is the robot selected to perform task t in the best case scenario. In the best case scenario, for a given chain $Q \in T_c$, at most $|S|$ robots must be selected and if $|T|$ is greater than $|S|$, at least a number of tasks must be assigned to the same station, and therefore, to the same robot. Thus, 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

$$M_Q(t,s) = \begin{cases} 0, & \text{if } t = n, \\ \infty, & \text{if } t < n \wedge s = 0, \\ \min\limits_{t<t' \leq n} \min\limits_{r \in R} \sum\limits_{t \leq t'' < t'} p_{t'',r} + M_Q(t', s-1), & \text{otherwise,} \end{cases} \tag{5.8}$$

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.

The 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 (5.8) is $M_Q(0,2) = 5$.

Function M can be determined by dynamic programming in time $O(n^2sr)$ for a chain of length n, and to calculate the result for all the chains, the total complexity is $O(|T|^2sr)$. We can define lower bound LC_{R2} by

$$LC_{R2} = \sum_{Q \in T_c} M_Q(0,|S|)/C.$$

Lower bound LC_{R2} 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_{R2} takes total time $O(|T|^2sr + |T|\,|A|)$.

5.2.1 Computational Results

To evaluate the lower bounds we need to consider instance characteristics that are not considered by the set of 32 instances in the literature. First, the number of robots and stations is equal, so it is not possible to study their influence 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 to adapt the SALBP instance set of Otto, Otto and Scholl (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 all the precedence relations possible: $|A|/(|T|^2 - |T|)$. 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 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

Table 5.3 – Parameters of the Instance Set of Otto, Otto and Scholl (2013)

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

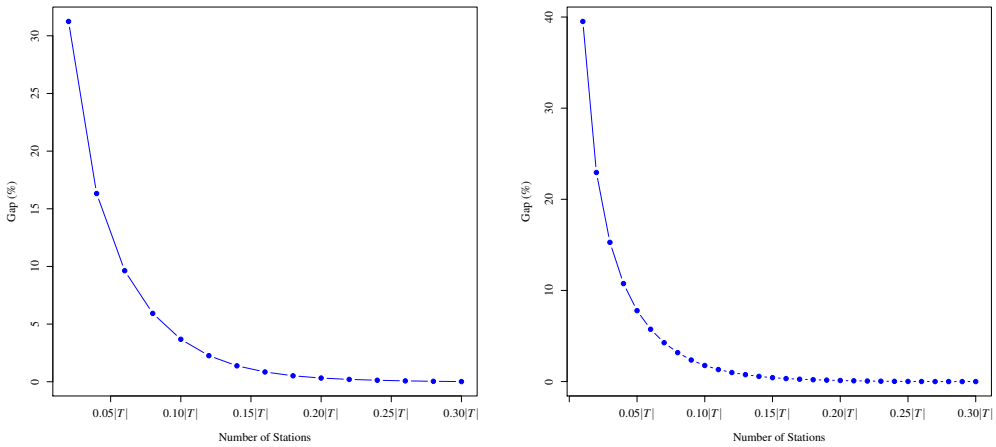


Figure 5.1 – Comparison of the two lower bounds. The figures show the gap between LC_{R2} and LC_1 as a function of the number of stations. On the left are the results for $n = 50$ tasks, on the right for $n = 100$ tasks.

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 5.3.

For each combination of the parameters above, Otto, Otto and Scholl (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 (OTTO; OTTO; SCHOLL, 2013) SALBP instance, the range for a task time p_{tr} can be $[p_t, 2p_t]$ (low variability), or $[p_t, 5p_t]$ (high variability). Therefore for each of the 210 instances selected of Otto, Otto and Scholl (2013), we generate twelve instances, totaling 2520 RALBP-2 instances to be used in our experiments.

Specifically for the lower bounds, we consider even more numbers of stations. We analyze $|S|$ from 1 to $0.3|T|$ for each of the instances in our benchmark set of instances. In Figure 5.1, we present the gaps between lower bounds LC_{R2} and LC_1 for 50 and 100 tasks.

In this figure, each point represents the average relative deviation $(LC_{R2} - LC_1)/LC_1$ of lower bound LC_{R2} 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

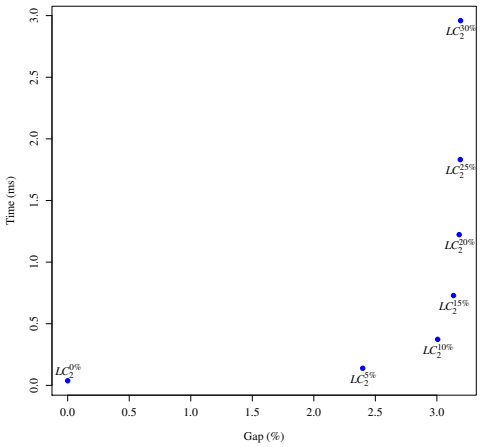


Figure 5.2 – Results of the lower bound LC_{R2}^{f} for varying values of f

LC_{R2} has better results when multiple tasks need to be assigned to the same station for some of the chains. Therefore, LC_{R2} is better if there are much fewer stations than tasks. Because of this, we have studied the application of LC_{R2} only in cases where the number of stations is smaller than a fraction f of the number of tasks ($|S| \leq f|T|$).

Figure 5.2 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 gap and the time increase with f. However, for $f \geq 20\%$, the time increases significantly with almost no improvements of the gap. The time needed to compute lower bounds $LC_{R2}^{0\%}$ and $LC_{R2}^{5\%}$ is very similar, but their results are significantly different. The gap is improved 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.

5.3 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 5.3.3. 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 one 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.

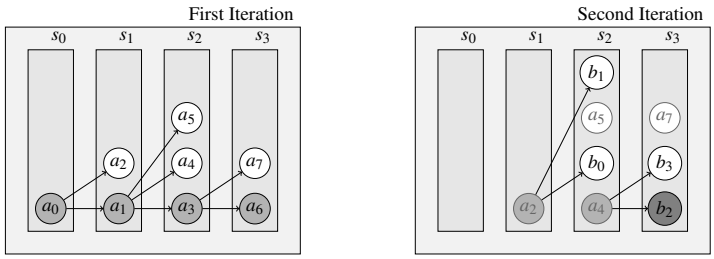


Figure 5.3 – An example of CBFS. In the first iteration the method expands the partial solutions a_0, a_1, a_3 and a_6. In the second iteration, the first priority queue is empty and the method continues to expand partial solutions a_2, a_4 and b_2, which is a valid solution for the given cycle time. The method ends after expanding five nodes.

5.3.1 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_{R2} can be used to prioritize the solutions and their performance will be evaluated in Section 5.3.4. When lower bound LC_{R2} 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 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 5.3.

5.3.2 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 Dilworth 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 $O((\overline{C}-\underline{C})l^w)$.

5.3.3 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 $\underline{C} = LC_1$. At each iteration of the method, we test the cycle time $(\underline{C}+\overline{C})/2$. 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_{R2} is not used in the beam search.

5.3.4 Computational results

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 gap. 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 5.4.

In Table 5.4 we present the relative deviation from the best known cycle time $(C - C^*)/C^*$ (column "Gap") 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 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 gap produced by our method is smaller or equal to that of either PSO or CS-PSO.

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 $LC_{R2}^{20\%}$. 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 5.5. It reports the relative deviation $(C - C^*)/C^*$ of the best cycle time C from the best known value C^* (columns "Gap"), 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 gap found by the BBR method with LC_1 is better than the gap found by the method with $LC_{R2}^{20\%}$. This can be explained by the longer time to compute lower bound $LC_{R2}^{20\%}$. 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 $LC_{R2}^{20\%}$ in this last step, and therefore, it takes more time to prove a solution to be optimal.

Table 5.4 – Comparison of the Iterative Beam Search (IBS) with the PSO and CS-PSO by (Mukund Nilakantan et al., 2015)

		PSO		CS-PSO		IBS	
|T|	**|R|**	**Gap (%)**	**t (s)**	**Gap (%)**	**t (s)**	**Gap (%)**	**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**

Table 5.5 – Comparison of the results of Model M_2 and the BBR methods.

		Model M_2		**BBR method with LC_1**		**BBR method with $LC_{R2}^{20\%}$**	
|T|	**|W|**	**Gap(%)**	**t (s)**	**Gap(%)**	**t (s)**	**Gap(%)**	**t (s)**
25	3	**0.00**	**0.10**	**0.00**	1.72	**0.00**	1.61
	4	**0.00**	**0.15**	**0.00**	1.89	**0.00**	1.60
	6	**0.00**	3.98	**0.00**	1.71	**0.00**	**1.43**
	9	**0.00**	22.01	**0.00**	1.64	**0.00**	**1.56**
35	4	**0.00**	**0.15**	**0.00**	1.48	**0.00**	1.55
	5	**0.00**	**1.43**	**0.00**	1.48	**0.00**	1.86
	7	**0.00**	351.72	**0.00**	**1.54**	**0.00**	1.62
	12	5.38	3,600.00	**0.00**	2.42	**0.00**	**1.75**
53	5	**0.00**	**0.36**	**0.00**	1.61	**0.00**	1.65
	7	**0.00**	9.27	**0.00**	2.07	**0.00**	**1.79**
	10	**0.00**	1,859.77	**0.00**	**1.68**	**0.00**	1.96
	14	2.99	3,600.00	**0.00**	**1.85**	**0.00**	1.96
70	7	**0.00**	1,332.32	**0.00**	239.94	**0.00**	**117.87**
	10	3.88	3,600.00	**0.00**	**19.62**	**0.00**	22.91
	14	5.29	3,600.00	**0.00**	**12.52**	**0.00**	12.75
	19	7.50	3,600.00	**0.00**	**25.30**	**0.00**	25.49
89	8	0.46	3,600.00	**0.00**	**26.05**	**0.00**	31.06
	12	1.71	3,600.00	**0.00**	**21.97**	**0.00**	33.09
	16	3.42	3,600.00	**0.00**	**21.63**	**0.00**	22.19
	21	7.10	3,600.00	**0.00**	17.19	**0.00**	**17.06**
111	9	3.00	**3,600.00**	**0.00**	**3,600.00**	0.64	**3,600.00**
	13	8.09	**3,600.00**	**1.10**	**3,600.00**	**1.10**	**3,600.00**
	17	5.71	**3,600.00**	**0.48**	**3,600.00**	0.95	**3,600.00**
	22	15.13	**3,600.00**	**0.66**	**3,600.00**	**0.66**	**3,600.00**
148	10	2.40	**3,600.00**	**1.66**	**3,600.00**	**1.66**	**3,600.00**
	14	2.27	**3,600.00**	**0.28**	**3,600.00**	**0.28**	**3,600.00**
	21	12.11	**3,600.00**	**1.35**	**3,600.00**	**1.35**	**3,600.00**
	29	15.79	**3,600.00**	**1.32**	**3,600.00**	**1.32**	**3,600.00**
297	19	10.27	**3,600.00**	**0.19**	**3,600.00**	**0.19**	**3,600.00**
	29	15.77	**3,600.00**	**0.30**	**3,600.00**	**0.30**	**3,600.00**
	38	26.72	**3,600.00**	**0.40**	**3,600.00**	**0.40**	**3,600.00**
	50	**0.20**	**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**

Table 5.6 – Comparison of the BBR method and the IBS on the instance set 2.

				IBS			BBR method with LC_1				BBR method with $LC_{R2}^{20\%}$			
|T|	Gr.	OS	Dist.	Δ_{lb}(%)	Δ_{bkv}(%)	t (s)	Prov. (%)	Δ_{lb}(%)	Δ_{bkv}(%)	t (s)	Prov. (%)	Δ_{lb}(%)	Δ_{bkv}(%)	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**

In instances where both methods prove a solution to be optimal, the method using $LC_{R2}^{20\%}$ 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, Otto and Scholl (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 5.6. It shows for each set of parameters with a fixed number of tasks ($|T|$), graph type (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 ("Δ_{lb}") between the current result and the best known lower bound for the instance, the average relative deviation ("Δ_{bkv}") between the current result and the best known value, 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 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 gap of the IBS is better than the gap by 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 5.3.2), 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 gaps 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 5.3, 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 $LC_{R2}^{20\%}$ solves more instances than the version with LC_1, and in average in less time, but the gaps 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 $LC_{R2}^{20\%}$ 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 $LC_{R2}^{20\%}$ 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_{R2} has better results for instances with large chains. This finding is corroborated by the fact that the instances with graphs *CH* and *MX* where $LC_{R2}^{20\%}$ improves the most have the highest order strength.

Table 5.7 – 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 $LC_{R2}^{20\%}$**			
|T|	**|R|**	**|S|**	**Var.**	**Δ_{lb}(%)**	**Δ_{bkv}(%)**	**t (s)**	**Prov. (%)**	**Δ_{lb}(%)**	**Δ_{bkv}(%)**	**t (s)**	**Prov. (%)**	**Δ_{lb}(%)**	**Δ_{bkv}(%)**	**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**

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 5.7. 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 (5.3.2) 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 5.3.2 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 tp 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 $LC_{R2}^{20\%}$ 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 $LC_{R2}^{20\%}$.

The BBR algorithm with $LC_{R2}^{20\%}$ 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_{R2} is more efficient when there are fewer stations.

6 CONCLUSIONS

This work proposed methods for two heterogeneous assembly line balancing problems: the Assembly Line Worker Assignment and Balancing Problem, and the Robotic Assembly Line Balancing Problem. Both problems studied were of type 2, i.e. they minimize the cycle time of the line. While they share some characteristics, they are very different to solve. In the ALWABP-2, a worker can only be assigned to one station, and in the RALBP-2 the robots are an unlimited resource.

We have proposed MIP models, lower bounds, heuristics and branch-and-bound methods that are the state-of-the-art for both problems. For the MIP models, we have used a very similar model as a basis for both problems. These models use two-index variables and it made possible the use of continuity constraints in the ALWABP-2. We have shown the importance of the continuity constraints to improve the results of the models. As for the lower bounds, the relation with the SALBP-2 is very strong for both problems, in different manners, and therefore variations of the lower bounds of the SALBP-2 were used in both of them. Also, specifically for the ALWABP-2, we observed that the precedence constraints influence the lower bounds less than the heterogeneities. Thus we proposed using lower bounds that relax the ALWABP-2 to the Unrelated Parallel Machine Scheduling Problem. For the RALPB-2, it was shown that there is a sufficiently fast algorithm that relaxes only a fraction of the precedence constraints. The results obtained by not relaxing all the precedence constraints were shown to be better than the results of the SALBP-2 lower bounds. This observation is stronger in cases with a low number of stations.

As for the heuristics, for the size of instances that are being solved in the literature, we have observed that beam search methods derived from station-oriented branch-and-bound methods are the best way to solve the problems. We used, in both cases, a station-oriented beam search with slight variations. In the case of the ALWABP-2, where it was impossible to generate all the station loads for a given station, we also used a sampling of the possible assignments to a station to decrease the time needed to generate the loads. These heuristics produce good results in a very small amount of time, and though they are competitive with the state-of-the-art heuristics, they are only used to produce initial upper bounds for our exact methods.

For the branch-and-bound methods, two branching strategies are used in the literature: the station-oriented methods and the task-oriented methods. For the ALWABP-2 the two methods have competitive results as we have seen by the comparison with the work of Vilà and Pereira (2014). Also, the two are complementary and we have shown that by using both we were able to solve and prove optimality of every instance in the literature of the ALWABP-2. The task-oriented method proposed by us is shown to be more efficient in instances with a high order strength, while the station-oriented method is more efficient in instances with a low order strength. For the RALBP-2 we have shown excellent results with an iterated station-oriented branch-bound-and-remember algorithm that uses cyclic best first search to find valid solutions

for fixed cycle times and since the method finds valid solutions early in the search, it can rapidly validate solutions and decrease the cycle time. This method has the best result for almost every instance in the literature and also proves solutions to be optimal in less time than most of the heuristics take to find valid solutions that are 10% worse. Two variants of the branch,bound-and-remember for the RALBP-2 are shown, one using the SALBP-2 lower bounds and one using our proposed lower bound. The results show that while the method with SALBP-2 bounds produces the smallest average gap, with a time limit of one hour, the method with the novel lower bound proves more solutions to be optimal in the same amount of time.

6.1 Future Work

While much work has been done to solve the instances in the literature and we have generated instances to evaluate the RALBP-2 algorithms, the current instances are mostly solved by our methods for both problems. Even for the RALBP-2 instances with 100 tasks, we have proved optimality of many of them and have found results that are very near to the lower bound of the problem. Therefore, to evaluate the limits of the algorithms, it is necessary to expand these instances. Both the branch-and-bound and the beam search methods are not scalable to instances with a large number of tasks. Thus, new heuristics based on local improvement instead of constructive heuristics should emerge to solve the larger instances.

The lower bounds LC_{R2} propose a concept that could be translated for other problems like the ALWABP-2 and the Assembly Line Design Problem (ALDP) (BUKCHIN; TZUR, 2000). The ALDP is very similar to the SALBP-1, but instead of minimizing the number of stations $\sum_{s\in S}\sum_{r\in R} y_{sr}$, where y_{sr} indicates if a robot r is assigned to a station s, the ALDP minimizes the cost of the line $\sum_{s\in S}\sum_{r\in R} EC_r y_{sr}$, where EC_r is the cost of robot r. Most of the solutions presented for the RALBP-F can be adapted with slight modifications for the ALDP. To adapt the branch,bound-and-remember, the only modification is in the memoization process. Instead of storing the number of stations with the tasks already assigned, the method for the ALDP should store the current cost of the line.

Besides that, further study on the use of task-oriented branch-and-bound methods could lead to interesting results. While the method has shown good results for the ALWABP-2, it has been outperformed by the station-oriented methods in the SALBP and the RALBP. Therefore, it is necessary to identify in which contexts the task-oriented branch-and-bound method should be used. As an example, there is the Mixed-Model Assembly Line Balancing Problem. In this problem, the tasks take a different time to be performed according to the model that is being produced. The cycle time of the line is given by the weighted average of the cycle times of the models. Therefore, a MMALBP instance of type 1, has a fixed cycle time, but it may lead to a combinatorial number of selections of cycle times for the models. Since the station-oriented method relies on the cycle times to use the maximal station load dominance rule, and solving the problem a combinatorial number of times for different cycle times is impracticable, the

task-oriented method emerges as a possibility to solve the MMALBP.

REFERENCES

BALAKRISHNAN, A.; VANDERBECK, F. *A Tactical Planning Model for Mixed-Model Electronics Assembly Operations*. [S.l.], 1993.

BATTAÏA, O.; DOLGUI, A. A taxonomy of line balancing problems and their solution-approaches. *International Journal of Production Economics*, v. 142, n. 2, p. 259–277, apr 2013.

BAUTISTA, J.; PEREIRA, J. Ant Algorithms for Assembly Line Balancing. In: DORIGO, M.; Di Caro, G.; SAMPELS, M. (Ed.). *Ant Algorithms SE - 6*. [S.l.]: Springer Berlin Heidelberg, 2002, (Lecture Notes in Computer Science, v. 2463). p. 65–75.

BAUTISTA, J.; PEREIRA, J. A dynamic programming based heuristic for the assembly line balancing problem. *European Journal of Operational Research*, v. 194, n. 3, p. 787–794, may 2009.

BAUTISTA, J.; PEREIRA, J. Procedures for the Time and Space constrained Assembly Line Balancing Problem. *European Journal of Operational Research*, v. 212, n. 3, p. 473–481, aug 2011.

BAYBARS, I. A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, v. 32, n. 8, p. 909–932, aug 1986.

BLUM, C. Beam-ACO for Simple Assembly Line Balancing. *INFORMS Journal on Computing*, v. 20, n. 4, p. 618–627, 2008.

BLUM, C.; MIRALLES, C. On solving the assembly line worker assignment and balancing problem via beam search. *Computers and Operations Research*, v. 38, n. 1, p. 328–339, 2011.

BORBA, L.; RITT, M. A heuristic and a branch-and-bound algorithm for the Assembly Line Worker Assignment and Balancing Problem. *Computers & Operations Research*, v. 45, n. 0, p. 87–96, 2014.

BOYSEN, N.; FLIEDNER, M. A versatile algorithm for assembly line balancing. *European Journal of Operational Research*, v. 184, n. 1, p. 39–56, jan 2008.

BOYSEN, N.; FLIEDNER, M.; SCHOLL, A. A classification of assembly line balancing problems. *European Journal of Operational Research*, v. 183, n. 2, p. 674–693, dec 2007.

BUKCHIN, J.; TZUR, M. Design of flexible assembly line to minimize equipment cost. *IIE Transactions (Institute of Industrial Engineers)*, Kluwer Academic Publishers, v. 32, n. 7, p. 585–598, 2000.

CHAVES, A.; LORENA, L.; MIRALLES, C. Hybrid Metaheuristic for the Assembly Line Worker Assignment and Balancing Problem. In: BLESA, M. et al. (Ed.). *Hybrid Metaheuristics*. [S.l.]: Springer Berlin / Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5818). p. 1–14.

CHAVES, A. A. *Uma meta-heurística híbrida com busca por agrupamentos aplicada a problemas de otimização combinatória*. 197 p. Thesis (PhD) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2009.

CHAVES, A. A.; MIRALLES, C.; LORENA, L. A. N. Clustering Search Approach for the Assembly Line Worker Assignment and Balancing Problem. *Proceedings of ICC&IE*, p. 1469–1478, 2007.

CHIANG, W.-C. The application of a tabu search metaheuristic to the assembly line balancing problem. *Annals of Operations Research*, Kluwer Academic Publishers, v. 77, n. 0, p. 209–227, 1998.

Coffman Jr., E. G. et al. Bin Packing Approximation Algorithms: Survey and Classification. In: PARDALOS, P. M.; DU, D.-Z.; GRAHAM, R. L. (Ed.). *Handbook of Combinatorial Optimization SE - 35*. [S.l.]: Springer New York, 2013. p. 455–531.

COSTA, V. A. D. *Formação na perspectiva da teoria crítica da sociedade: As experiências dos trabalhadores deficientes visuais do serviço federal de processamento de dados*. Thesis (PhD) — Pontifícia Universidade Católica de São Paulo, São Paulo, 2001.

CUNNINGHAM, J. A. Management: Using the learning curve as a management tool: The learning curve can help in preparing cost reduction programs, pricing forecasts, and product development goals. *Spectrum, IEEE*, v. 17, n. 6, p. 45–48, 1980.

DAR-EL, E. M. MALB - A Heuristic Technique for Balancing Large Single-Model Assembly Lines. *A I I E Transactions*, Taylor & Francis, v. 5, n. 4, p. 343–356, dec 1973. ISSN 0569-5554. Available from Internet: <http://dx.doi.org/10.1080/05695557308974922>.

DAVIS, L. Applying Adaptive Algorithms to Epistatic Domains. In: *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985. (IJCAI'85), p. 162–164. ISBN 0-934613-02-8, 978-0-934-61302-6.

DILWORTH, R. P. A Decomposition Theorem for Partially Ordered Sets. *The Annals of Mathematics*, Annals of Mathematics, v. 51, n. 1, 1950.

FANJUL-PEYRO, L.; RUIZ, R. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, v. 207, n. 1, p. 55–69, nov 2010.

FANJUL-PEYRO, L.; RUIZ, R. Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers & Operations Research*, v. 38, n. 1, p. 301–309, jan 2011.

FEKETE, S. P.; SCHEPERS, J. New Classes of Lower Bounds for Bin Packing Problems. In: BIXBY, R.; BOYD, E.; RÍOS-MERCADO, R. (Ed.). *Integer Programming and Combinatorial Optimization SE - 20*. [S.l.]: Springer Berlin Heidelberg, 1998, (Lecture Notes in Computer Science, v. 1412). p. 257–270.

FISCHETTI, M.; TOTH, P. An Additive Bounding Procedure for Combinatorial Optimization Problems. *Operations Research*, v. 37, n. 2, p. 319–328, 1989.

FLESZAR, K.; HINDI, K. S. An enumerative heuristic and reduction methods for the assembly line balancing problem. *European Journal of Operational Research*, v. 145, p. 606–620, 2003.

GAO, J. et al. An efficient approach for type II robotic assembly line balancing problems. *Computers & Industrial Engineering*, v. 56, n. 3, p. 1065–1080, apr 2009.

GEOFFRION, A. *Lagrangean Relaxation and Its Uses in Integer Programming*. [S.l.]: Defense Technical Information Center, 1972.

GHIRARDI, M.; POTTS, C. N. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, v. 165, n. 2, p. 457–467, sep 2005.

GLASS, C.; POTTS, C.; SHADE, P. Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling*, v. 20, n. 2, p. 41–52, jul 1994.

GÖKÇEN, H.; AĞPAK, K.; BENZER, R. Balancing of parallel assembly lines. *International Journal of Production Economics*, v. 103, n. 2, p. 600–609, oct 2006.

GONÇALVES, J. F.; ALMEIDA, J. R. de. A Hybrid Genetic Algorithm for Assembly Line Balancing. *Journal of Heuristics*, v. 8, n. 6, p. 629–642, 2004.

GRAHAM, R. L. et al. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In: P.L. Hammer, E. L. J.; MATHEMATICS, B. H. K. B. T. A. of D. (Ed.). *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and V*. [S.l.]: Elsevier, 1979. Volume 5, p. 287–326.

GREENBERG, H. J.; PIERSKALLA, W. P. Surrogate mathematical programming. *Operations Research*, INFORMS, v. 18, n. 5, p. 924–939, 1970.

GUO, Y. et al. Minimizing the Makespan for Unrelated Parallel Machines. *International Journal on Artificial Intelligence Tools*, WORLD SCIENTIFIC PUBLISHING, v. 16, n. 3, p. 399, 2007.

HEIKE, G. et al. Mixed model assembly alternatives for low-volume manufacturing: The case of the aerospace industry. *International Journal of Production Economics*, v. 72, n. 2, p. 103–120, jul 2001.

HOFFMANN, T. R. Assembly Line Balancing with a Precedence Matrix. *Management Science*, v. 9, n. 4, p. 551–562, 1963.

HOFFMANN, T. R. Eureka: A Hybrid System for Assembly Line Balancing. *Management Science*, v. 38, n. 1, p. 39–47, 1992.

ITALIANO, G. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, v. 48, p. 273–281, jan 1986.

JACKSON, J. R. A Computing Procedure for a Line Balancing Problem. *Management Science*, v. 2, n. 3, p. 261–271, 1956.

JOHNSON, R. V. Optimally Balancing Large Assembly Lines With'FABLE'. *Management Science*, JSTOR, p. 240–253, 1988.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. [S.l.: s.n.], 1995. v. 4, p. 1942–1948 vol.4.

KILBRIDGE, M.; WESTER, L. The Balance Delay Problem. *Management Science*, INFORMS, v. 8, n. 1, p. 69–84, oct 1961. ISSN 0025-1909. Available from Internet: <http://dx.doi.org/10.1287/mnsc.8.1.69>.

KIM, Y. K.; SONG, W. S.; KIM, J. H. A mathematical model and a genetic algorithm for two-sided assembly line balancing. *Computers and Operations Research*, v. 36, n. 3, p. 853–865, 2007.

KLEIN, R.; SCHOLL, A. Maximizing the Production Rate in Simple Assembly Line Balancing–A Branch and Bound Procedure. *European Journal of Operational Research*, Elsevier, v. 91, n. 2, p. 367–385, 1996.

KLEIN, R.; SCHOLL, A. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European Journal of Operational Research*, v. 112, n. 2, p. 322–346, 1999.

LAPIERRE, S. D.; RUIZ, A.; SORIANO, P. Balancing assembly lines with tabu search. *European Journal of Operational Research*, v. 168, n. 3, p. 826–837, feb 2006.

LEU, Y.-Y.; MATHESON, L. A.; REES, L. P. Assembly Line Balancing Using Genetic Algorithms with Heuristic-Generated Initial Populations and Multiple Evaluation Criteria*. *Decision Sciences*, Blackwell Publishing Ltd, v. 25, n. 4, p. 581–605, 1994.

LEVITIN, G.; RUBINOVITZ, J.; SHNITS, B. A genetic algorithm for robotic assembly line balancing. *European Journal of Operational Research*, v. 168, n. 3, p. 811–825, feb 2006.

LIN, Y. K.; PFUND, M. E.; FOWLER, J. W. Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computers & operations research*, Elsevier, v. 38, n. 6, p. 901–916, 2011.

LIU, S. B.; ONG, H. L.; HUANG, H. C. Two bi-directional heuristics for the assembly line type II problem. *The International Journal of Advanced Manufacturing Technology*, Springer-Verlag, v. 22, n. 9-10, p. 656–661, 2003.

LOWERRE, B. *The Harpy Speech Recognition System*. Thesis (PhD), 1976.

MARTELLO, S.; PISINGER, D.; TOTH, P. Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem. *Management Science*, v. 45, n. 3, p. 414–424, 1999.

MARTELLO, S.; SOUMIS, F. Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete applied mathematics*, v. 75, n. 2, p. 169–188, 1997.

MARTELLO, S.; TOTH, P. *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

MILTENBURG, G. J.; WIJINGAARD, J. The U-Line Line Balancing Problem. *Management Science*, INFORMS, v. 40, n. 10, p. pp. 1378–1388, 1994.

MIRALLES, C. et al. Advantages of assembly lines in Sheltered Work Centres for Disabled. A case study. *International Journal of Production Economics*, v. 110, n. 1-2, p. 187–197, oct 2007.

MIRALLES, C. et al. Branch and bound procedures for solving the Assembly Line Worker Assignment and Balancing Problem: Application to Sheltered Work centres for Disabled. *Discrete Applied Mathematics*, v. 156, n. 3, p. 352–367, feb 2008.

MIRALLES, C. et al. Branch and Bound Procedures for Solving the Assembly Line Worker Assignment and Balancing Problem: Application to Sheltered Work Centres for Disabled. *Discrete Applied Mathematics*, Elsevier, v. 156, n. 3, p. 352–367, 2008.

MOKOTOFF, E.; CHRÉTIENNE, P. A cutting plane algorithm for the unrelated parallel machine scheduling problem. *European Journal of Operational Research*, v. 141, n. 3, p. 515–525, 2002.

MOKOTOFF, E.; JIMENO, J. L. Heuristics Based on Partial Enumeration for the Unrelated Parallel Processor Scheduling Problem. *Annals of Operations Research*, Kluwer Academic Publishers, v. 117, n. 1-4, p. 133–150, 2002.

MOREIRA, M. et al. Simple heuristics for the assembly line worker assignment and balancing problem. *Journal of Heuristics*, Springer Netherlands, v. 18, n. 3, p. 505–524, 2012.

MOREIRA, M. C. d. O.; COSTA, A. M. A minimalist yet effective tabu search algorithm for balancing assembly lines with disabled workers. In: *Proceedings of the XLI SBPO 2009*. [S.l.: s.n.], 2009. p. 660–671.

MORRISON, D. R. et al. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, v. 19, n. Supplement C, p. 79 – 102, 2016. ISSN 1572-5286. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1572528616000062>.

MORRISON, D. R.; SEWELL, E. C.; JACOBSON, S. H. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. *European Journal of Operational Research*, v. 236, n. 2, p. 403–409, jul 2014.

Mukund Nilakantan, J. et al. Bio-inspired search algorithms to solve robotic assembly line balancing problems. *Neural Computing and Applications*, v. 26, n. 6, p. 1379–1393, 2015. ISSN 1433-3058. Available from Internet: <http://dx.doi.org/10.1007/s00521-014-1811-x>.

MUTLU, Ö.; POLAT, O.; SUPCILLER, A. A. An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. *Computers & Operations Research*, v. 40, n. 1, p. 418–426, 2013.

NEARCHOU, A. Balancing large assembly lines by a new heuristic based on differential evolution method. *The International Journal of Advanced Manufacturing Technology*, Springer-Verlag, v. 34, n. 9-10, p. 1016–1029, 2007.

NICOSIA, G.; PACCIARELLI, D.; PACIFICI, A. Optimally balancing assembly lines with different workstations. *Discrete Applied Mathematics*, v. 118, n. 1-2, p. 99–113, apr 2002.

NILAKANTAN, J. M.; PONNAMBALAM, S. An efficient PSO for type II robotic assembly line balancing problem. In: *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. [S.l.]: IEEE, 2012. p. 600–605. ISBN 978-1-4673-0430-6. ISSN 21618070.

NOURIE, F. J.; VENTA, E. R. Finding optimal line balances with OptPack. *Operations Research Letters*, v. 10, n. 3, p. 165–171, 1991.

OLIVEIRA, R. C. M.; LORENA, L. A. N. Detecting Promising Areas by Evolutionary Clustering Search. *Advances in Artificial Intelligence. Springer Lecture Notes in Artificial Intelligence Series*, v. 3171, p. 385–384, 2004.

Organisation for Economic Co-Operation and Development. *Transforming Disability into Ability*: *Policies to Promote Work and Income Security for Disabled People*. Paris: OECD Publishing, 2003. 220 p.

OTTO, A.; OTTO, C.; SCHOLL, A. Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research*, v. 228, n. 1, p. 33–45, jul 2013. ISSN 03772217. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0377221713000039>.

OW, P. S.; MORTON, T. Filtered beam search in scheduling. *International Journal of Production Research*, v. 26, n. 1, p. 35–62, 1988.

PARK, K.; PARK, S.; KIM, W. A heuristic for an assembly line balancing problem with incompatibility, range, and partial precedence constraints. *Computers & Industrial Engineering*, v. 32, n. 2, p. 321–332, apr 1997.

Passmark Software Pty. Ltd. *Passmark benchmarks*. 2017. Available from Internet: <http://www.cpubenchmark.net>.

PEETERS, M.; DEGRAEVE, Z. An linear programming based lower bound for the simple assembly line balancing problem. *European Journal of Operational Research*, v. 168, n. 3, p. 716–731, feb 2006.

PEREIRA, J. Empirical evaluation of lower bounding methods for the simple assembly line balancing problem. *International Journal of Production Research*, Taylor & Francis, v. 53, n. 11, p. 3327–3340, nov 2014.

PIERSMA, N.; DIJK, W. van. A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search. *Mathematical and Computer Modelling*, v. 24, n. 9, p. 11–19, nov 1996.

RITT, M.; COSTA, A. M. Improved integer programming models for simple assembly line balancing and related problems. *International Transactions in Operational Research*, 2015.

RUBINOVITZ, J.; BUKCHIN, J.; LENZ, E. RALB - A Heuristic Algorithm for Design and Balancing of Robotic Assembly Lines. *CIRP Annals - Manufacturing Technology*, v. 42, n. 1, p. 497–500, jan 1993.

RUBINOVITZ, J.; BUKCHIN, J.; of Manufacturing Engineers, S. Design and Balancing of Robotic Assembly Lines. In: *Proceedings of the fourth world conference on on robotics research*. Pittsburgh (PA): Society of Manufacturing Engineers, 1991. (Creative manufacturing engineering program).

SABUNCUOGLU, I.; EREL, E.; TANYER, M. Assembly line balancing using genetic algorithms. *Journal of Intelligent Manufacturing*, Kluwer Academic Publishers, v. 11, n. 3, p. 295–310, 2000.

SALVESON, M. E. The assembly line balancing problem. *Journal of Industrial Engineering*, v. 77, p. 393–948, 1955.

SCHOLL, A. Ein B-&-B-Verfahren Zur Abstimmung von Einprodukt-Fließbändern Bei Gegebener Stationsanzahl. *Publications of Darmstadt Technical University, Institute for Business Studies (BWL)*, Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL), 1993.

SCHOLL, A.; BECKER, C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, v. 168, n. 3, p. 666–693, feb 2006.

SCHOLL, A.; KLEIN, R. SALOME: A Bidirectional Branch-and-Bound Procedure for Assembly Line Balancing. *INFORMS Journal on Computing*, v. 9, n. 4, p. 319–334, 1997.

SCHOLL, A.; KLEIN, R. Balancing assembly lines efectively - A computational comparison. *European Journal of Operational Research*, v. 114, p. 50–58, 1999.

SCHOLL, A.; VOSS, S. Simple assembly line balancing - Heuristic approaches. *Journal of Heuristics*, Springer Netherlands, v. 2, n. 3, p. 217–244, 1997.

SCHRAGE, L.; BAKER, K. R. Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, 1978.

SELS, V. et al. Hybrid tabu search and a truncated branch-and-bound for the unrelated parallel machine scheduling problem. *Computers & Operations Research*, v. 53, n. 0, p. 107–117, 2015.

SEWELL, E. C.; JACOBSON, S. H. A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem. *INFORMS Journal on Computing*, INFORMS, v. 24, n. 3, p. 433–442, jun 2011. ISSN 1091-9856.

SEWELL, E. C.; JACOBSON, S. H. A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem. *INFORMS Journal on Computing*, v. 24, n. 3, p. 433–442, 2012.

SOURD, F. Scheduling Tasks on Unrelated Machines: Large Neighborhood Improvement Procedures. *Journal of Heuristics*, Kluwer Academic Publishers, v. 7, n. 6, p. 519–531, 2001.

SPRECHER, A. Dynamic search tree decomposition for balancing assembly lines by parallel search. *International Journal of Production Research*, v. 41, n. 7, p. 1413–1430, 2003.

SURESH, G.; SAHU, S. Stochastic assembly line balancing using simulated annealing. *International Journal of Production Research*, v. 32, n. 8, p. 1801–1810, 1994.

UĞURDAĞ, H. F.; RACHAMADUGU, R.; PAPACHRISTOU, C. A. Designing paced assembly lines with fixed number of stations. *European Journal of Operational Research*, v. 102, n. 3, p. 488–501, nov 1997. ISSN 03772217.

VELDE, S. L. van de. Duality-Based Algorithms for Scheduling Unrelated Parallel Machines. *ORSA Journal on Computing*, ORSA, v. 5, n. 2, p. 192–205, may 1993.

VILÀ, M.; PEREIRA, J. An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time. *European Journal of Operational Research*, v. 229, n. 1, p. 106–113, aug 2013.

VILÀ, M.; PEREIRA, J. A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research*, v. 44, p. 105–114, apr 2014.

VILA, M.; PEREIRA, J. A branch-and-bound algorithm for assembly line worker assignment and balancing problems. v. 44, p. 105–114, 2014.

World Health Organization. Book, Online. *World report on disability*. [S.l.]: World Health Organization Geneva, 2011. 325 p.