A TASK-ORIENTED BRANCH-AND-BOUND METHOD FOR THE ASSEMBLY LINE WORKER ASSIGNMENT AND BALANCING PROBLEM

Leonardo de Miranda Borba
Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
lmborba@inf.ufrgs.br

Marcus Rolf Peter Ritt
Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
mrpritt@inf.ufrgs.br

ABSTRACT

This paper discusses an exact solution branch-and-bound solver for the Assembly Line Worker Assignment and Balancing Problem Type 2 (ALWABP-2). This problem consists of assigning tasks to workers, satisfying precedence relationships between tasks, so that the production rate is maximized. The problem is complicated by the fact that each worker may need a different time to execute a task or even may be unable to execute it. The solution for this problem can improve the production of Sheltered Work centres for Disabled (SWD), and thus create new job opportunities for disabled persons. The proposed task-oriented branch-and-bound method solves it by selecting only task-to-worker assignments that satisfy the problem constraints and can improve the overall solution. Computational results show that the method achieves promising results on the test instances in literature.

KEYWORDS. Branch-and-Bound. Assembly Line Balancing. ALWABP.

Combinatorial optimization

RESUMO

Este artigo discute um método de solução exata por branch-and-bound para o Tipo 2 do Problema de Balanceamento e Atribuição de Trabalhadores em Linhas de Montagem (ALWABP-2). Este problema consiste em atribuir tarefas a trabalhadores, satisfazendo relações de precedência entre as tarefas, de forma que a taxa de produção seja maximizada. O problema é complicado pelo fato de que cada trabalhador pode necessitar de um tempo diferente para executar uma tarefa ou, até mesmo, pode ser incapaz de executá-la. A solução para este problema pode melhorar a produção dos Centros de Trabalho para Deficientes (CTDs) e, então, criar novas oportunidades de trabalho para as pessoas com deficiência. O método de branch-and-bound orientado a tarefas proposto resolve o problema selecionando somente atribuições de tarefa para trabalhador que satisfaçam as restrições e possam melhorar a solução global. Os resultados computacionais mostram que o método alcança resultados promissores nas instâncias de teste da literatura.

PALAVRAS CHAVE. Branch-and-Bound, Balanceamento de Linhas de Produção, ALWABP.

Otimização combinatória

1. Introduction

There are more than 785 million disabled persons around the world, according to the WHO (World Health Organization), 110 million of these with a severe deficiency degree (Organization, 2011). Additionally, because of common stereotypes, this group lacks job opportunities. Since professional activities are both therapeutical treatment and social inclusion mechanisms for disabled persons, Sheltered Work centres for Disabled (SWD) were created as a means to include these people in the professional market.

Miralles et al. (2007) have demonstrated that an assembly line is a production method that, when applied in SWD, by division of work in smaller tasks, make the differences among the disabled workers almost invisible. Furthermore, with correctly assigned tasks, it becomes a good therapeutic treatment. A SWD is a Not-For-Profit organization, so, the optimization of its assembly line can help to create more jobs for disabled people, contributing to their social inclusion.

The optimization of traditional assembly lines is a well known problem. The simplest variant is called the Simple Assembly Line Balancing Problem (SALBP) (Scholl and Becker, 2006). This problem presupposes equal task execution times for all workers. In the SWD case, however, the task execution time varies according to the degree of ability of the worker and some workers may even be unable to execute some tasks. Therefore, a new problem based on SWD assembly lines was defined in Miralles et al. (2008), called the Assembly Line Worker Assignment and Balancing Problem (ALWABP).

The assembly of a product in the SWD requires the execution of all tasks of a set T by workers in a set W. The constant $p_{w,t}$ represents the time for worker $w \in W$ to execute task $t \in T$. Furthermore, the tasks in T are partially ordered. The precedences are modeled by a transitively reduced graph $G(T,E)$, whose nodes are the tasks and an arc represents a dependency between two tasks. In other words, an arc $(u,v) \in E$ means that task u should be executed before task v. Each workstation of the SWD must be controlled by a worker $w \in W$, executing a subset of tasks $ST_w \subseteq T$. Stations are positioned along a conveyor belt, with all the above precedence graph dependencies satisfied. Thus, if a task t depends on a task t', t must be executed at a station after the workstation executing t' or at the same station.

With a worker w and a subset ST_w assigned to a station, its total execution time, called the station time, is given by the sum of $p_{t,w}$ for all $t \in ST_w$. The largest station time of all workstations defines the cycle time C of the assembly line. Stated this, there are four possible objectives for the problem: Minimize the number of stations needed to achieve a given cycle time (ALWABP-1), minimize the cycle time using a fixed number of stations (ALWABP-2), verify the existence of a solution with given cycle time and number of stations (ALWABP-F), or minimize the multiplication of cycle time and number of stations (ALWABP-E). The main objective of a SWD is to increase the number of workers, which is accomplished by optimizing the production rate, i.e. decreasing the cycle time. Thus, the studied problem in this paper will be the ALWABP-2.

In the literature, there are only a few exact methods for the ALWABP-2, since it is a complex problem. Therefore, this work proposes a new task-oriented branch-and-bound method for solving the ALWABP-2.

In the following sections, we will describe solutions for the ALWABP-2, using the notation presented in Table 1.

Section 2 of the paper gives a literature review of SALBP and ALWABP, providing the necessary background for the later sections. The mixed integer programming model for the ALWABP-2 commonly used in literature is presented in Section 3. This model will be compared later to the proposed task-oriented branch-and-bound which is explained in Section 4. Section 5 experimentally

Table 1. Notation

S	set of workstations;
W	set of workers;
T	set of tasks;
$G(T,E)$	precedence graph of tasks;
$p_{w,i} \in \mathbb{N}$	execution time of task i when executed by worker w;
$I_w \subseteq T$	set of tasks unfeasible for worker w;
$A_w = T \setminus I_w$	set of admissible tasks for worker w;
$A_i = \{w \in W \mid i \in A_w\}$	set of admissible workers for task i;
$P_i \subseteq T$	set of immediate predecessors of task i in the precedence graph;
$M \geq \sum_{w \in W} \sum_{i \in T} p_{w,i}$	a large constant;
$x_{s,w,i}$	a binary variable, equal to 1 if task i is assigned to worker w at workstation s, and 0 otherwise;
$y_{s,w}$	a binary variable, equal to 1 if worker w is assigned to station s, and 0 otherwise;
a_t	an integer variable, equal to the index of the worker assigned to task t;
C	a real variable, representing the cycle time of a solution;

compares the two exact solution procedures presented and evaluates their effectiveness according to execution time and the number of nodes in the branch-and-bound search tree. Finally, Section 6 gives a summary of this paper and proposes future research on this subject.

2. Related Work

The ALWABP-2 was proposed in Miralles et al. (2007). Since the ALWABP is an extension of the SALBP, the methods for solving the latter problem are the base for most of the algorithms for the ALWABP. In the SALBP, a group of tasks should be assigned to a group of stations. But, unlike ALWABP, the execution time of a task is the same regardless of the assigned station. In practice, any instance of SALBP can be reduced to an instance of ALWABP by creating as many workers as needed, with equal task execution times.

Baybars (1986) reviews the literature on solving the Simple Assembly Line Balancing Problem. A more recent review is given by Scholl and Becker (2006), which deals with exact and heuristic solutions for SALBP-1 and SALBP-2. The branch-and-bound methods referenced in that paper are classified as station-oriented (Klein and Scholl, 1996), (Scholl and Klein, 1997) and task-oriented (Sprecher, 2003), (Sprecher, 1999) methods.

Being a recently introduced problem, ALWABP has yet few solution procedures, and most of them are based on meta-heuristics. In Chaves et al. (2007) and Chaves et al. (2009), for example, the authors describe two Clustering Search methods for solving this problem. Furthermore, they propose a set of four families of instances named Roszieg, Heskia, Tonge and Wee-Mag. The total of 320 instances represent all combinations of low and high rates for the number of tasks, the number of workers, the order strength, the variability of the execution time, and the number of infeasibilities for worker-task pairs.

Moreira and Costa (2009) describes a Tabu Search method for ALWABP2, with better results than Chaves et al. (2009) for large instances. Moreira et al. (2012) proposes a simple constructive heuristic based on the SALBP heuristic proposed by Scholl and Voß (1997), and uses it inside a hybrid genetic algorithm, improving over previous works.

All these solutions are heuristic algorithms, giving approximate solutions for the problem. The only known branch-and-bound method for ALWABP-2 has been proposed by Miralles et al. (2008). This method executes successive calls to a branch-and-bound procedure for ALWABP-1, for an increasing cycle time. The first cycle time feasible for the specified number of stations is the minimum cycle time for ALWABP-2. In the paper, the authors also propose an integer programming formulation, subsequently implemented in Chaves (2009), which is the fastest exact method in the literature.

3. Mathematical Model

In this section we present a mathematical model for the ALWABP-2 proposed by Miralles et al. (2008).

$$\textbf{minimize} \quad C \tag{1}$$

$$\textbf{subject to} \quad \sum_{w\in A_i}\sum_{s\in S} x_{s,w,i} = 1 \qquad \forall i \in T \tag{2}$$

$$\sum_{s\in S} y_{s,w} \leq 1 \qquad \forall w \in W \tag{3}$$

$$\sum_{w\in W} y_{s,w} \leq 1 \qquad \forall s \in S \tag{4}$$

$$\sum_{w\in A_i}\sum_{s\in S} s\cdot x_{s,w,i} \leq \sum_{w\in A_j}\sum_{s\in S} s\cdot x_{s,w,j} \qquad \forall j \in T, i \in P_j \tag{5}$$

$$\sum_{i\in A_w} p_{w,i}\cdot x_{s,w,i} \leq C \qquad \forall w \in W, s \in S \tag{6}$$

$$\sum_{i\in A_w} x_{s,w,i} \leq M y_{s,w} \qquad \forall w \in W, s \in S \tag{7}$$

$$x_{s,w,i} = 0 \qquad \forall w \in W, s \in S, i \in I_w \tag{8}$$

$$x_{s,w,i} \in \{0,1\} \qquad \forall s \in S, w \in W, i \in A_w \tag{9}$$

$$y_{s,w} \in \{0,1\} \qquad \forall s \in S, w \in W \tag{10}$$

$$C \in \mathbb{R}. \tag{11}$$

The decision variables $x_{s,w,i}$ and $y_{s,w}$, as explained in Table 1, represent an assignment of tasks to workers and stations. As stated in the constraint (2), a task t should be assigned to one and only one worker w at a workstation s. Furthermore, one station will be allocated to at most one worker and one worker will be assigned to at most one station, as defined in (3) and (4). The precedence graph dependencies constraints are guaranteed by (5). If a task t depends on another task u, the stations executing t (SA_t) and u (SA_u) should respect $SA_u \leq SA_t$. The cycle time C is defined in (6), as the station time. Restriction (7) forces consistency of $x_{s,w,i}$ with the assignments $y_{s,w}$. And finally, restriction (8) prohibits infeasible assignments.

4. A Branch-and-Bound Algorithm

A common characteristic of ALWABP and SALBP is the relation between the problem types 1 and 2. The optimal solution of SALBP-2 can be obtained by successive executions of the SALBP-1 instance. A similar relation is valid for ALWABP-1 and ALWABP-2. Miralles et al. (2008) uses this fact to solve ALWABP-2 using subsequent branch-and-bound solutions of ALWABP-1. This solution causes repetitive visits of nodes in the branch-and-bound search tree, slowing down the program.

Therefore, a direct approach to the ALWABP-2 seems more adequate. To this end, there are two main branch-and-bound strategies, analogous to those used for solving the SALBP: a station-oriented and a task-oriented strategy. In a station-oriented solution, the station n should be filled before considering the next station $n+1$. In task-oriented methods, a task is assigned to any worker and if the solution becomes invalid, according to the problem constraints, the node is rejected and the branch-and-bound continues.

The order of the workers, in the task-oriented method, when applied to the ALWABP, is implicit by the assignment of tasks, as will be explained later. Thus, the assignment of workers to stations does not need to be considered when using this strategy. Because of this, this solution tends to be simpler than the station-oriented methods. Therefore, in next subsections we propose a task-oriented branch-and-bound method.

4.1. Branching strategy

The task-oriented method applied to the SALBP assigns in every node a task to a station. For the ALWABP-2, the important choice is the worker which will execute a task. Therefore, we propose a branching strategy which in every node assigns a task to a worker.

The next task to be assigned is selected greedily to be one of the tasks with the fewest possible choices for the worker and if there is a tie, the task t is selected from the assignment of one of the tied tasks to a feasible worker with smallest lower bound. Then, a new branch is created for each worker w that may execute t. The procedure *branch_tasks*, shown in Algorithm 1, stops when all the tasks are correctly assigned.

Algorithm 1: *branch_tasks*(llb)

```
input : A global upper bound gub.
input : A global subset UT ⊆ T of already used tasks.
input : A local lower bound, based on the already defined assignments.

if UT = T then
    if llb < gub then gub ← llb;
    return
end
select t ∈ (T \ UT) according to the proposed tasks priority rule;
V ← sort W according to the proposed workers priority rule ;
foreach w ∈ V do
    if assignment_is_valid(t, w) then
        new_llb ← calculate lower bound with new assignment (t, w);
        if new_llb < gub then
            set_assignment(t, w);
            branch_tasks(new_llb);
            unset_assignment(t, w);
        end
    end
end
```

4.2. Lower and upper bounds

There are two types of lower bounds in this algorithm. The first type is the global lower bound. This bound is calculated before the branch-and-bound algorithm and the result is used as initial lower bound for *branch_tasks*. The second type are local lower bounds. In this second type, according to the assignments already defined in a node, the algorithm updates the lower bounds used

in the branches below the current node. These bounds are useful to define which partial solutions can achieve better results than the current upper bound.

Let us define $p_i^- = \min\{p_{w,i} \mid w \in W\}$. For each task i in the ALWABP instance, create a task with execution time equal to p_i^- in a SALBP instance. An optimal solution for this SALBP, with the same dependencies graph of ALWABP, will give a lower bound for the main problem. Thus, all SALBP lower bounds can be applied to this subproblem to find lower bounds for ALWABP.

A first SALBP lower bound $LB_1 = \lceil \sum_{i \in T} p_i^- / |S| \rceil$ is obtained by dividing the tasks equally among the stations. Furthermore, since all tasks are executed non-preemptively, the maximum task execution time for SALBP is also a lower bound $LB_2 = \max_{i \in T} p_i^-$ for this and, so, for ALWABP.

We also use an ALWABP-specific lower bound in this paper given by the linear relaxation of the integer programming model above. This solution is not used as a local lower bound, since solving the linear relaxation is too costly during the branch-and-bound procedure. Nevertheless, the relaxation gives the best known global lower bound (Moreira et al., 2012).

During the branch-and-bound procedure, the local lower bounds can be calculated by updating the previous values. For each new assignment of a task t to a worker w, three local lower bounds are considered. First of all, each assignment made is a new restriction for the problem. Thus, the solution for the new partial problem will be greater or equal to the previous partial solution. Therefore, the lower bounds in the parent node are considered as lower bounds for the current node. Since no further solution in the branch will eliminate assignments of tasks to workers, the accumulated execution times of all tasks already assigned to any worker is also a local lower bound LB_3 for a node. Moreover, the lower bound LB_1 can be improved during the branch-and-bound procedure, by replacing the minimum task time p_i^- by the task time of the worker already assigned to task i. All these local lower bounds can be calculated in time $O(1)$, by simply updating the previous results.

A simple global upper bound for the problem is presented in Moreira et al. (2012). The authors propose a constructive heuristic to ALWABP, generating good feasible solutions for the problem in a short time, so a simplification of this method is applied to provide an initial upper bound for our branch-and-bound method.

In this upper bound procedure, a value v is heuristically tested as a solution for the problem and is increased until the test succeeds. The test consists in filling all stations based on greedy task assignments. For each worker w, the valid tasks are sorted according to the execution time $p_{w,t}$ and, so, are assigned to w until the limit v is reached. The worker which executes the greatest number of tasks assigned is selected and assigned to the current station. The algorithm proceeds, until the last station. If all tasks are assigned to a station during this procedure, the solution is valid and hence, yields an upper bound to the problem.

4.3. Valid partial solutions

To validate a solution, an important data structure is the precedence graph of workers, defined by the precedences of the tasks which have been already assigned. For each task t assigned to a worker w, a new set of arcs is added to this precedence graph. If task t depends on a task u, already assigned to worker a_u, a new arc between w and a_u is created. Furthermore, this is a transitive graph. In other words, if another worker v has an arc to a_u, v should have an arc to w too. In Figure 1, an example of an update of the precedence graph of workers is shown. The graph at the top shows the dependencies between the tasks and the bottom part of the figure shows how the assignment of task 1 to worker 1 affects the partial worker dependency graph. When assigning a new task to some worker, this graph can be updated in worst case time $O(|T||W|)$.

The worker precedence graph simplifies the implementation of the function

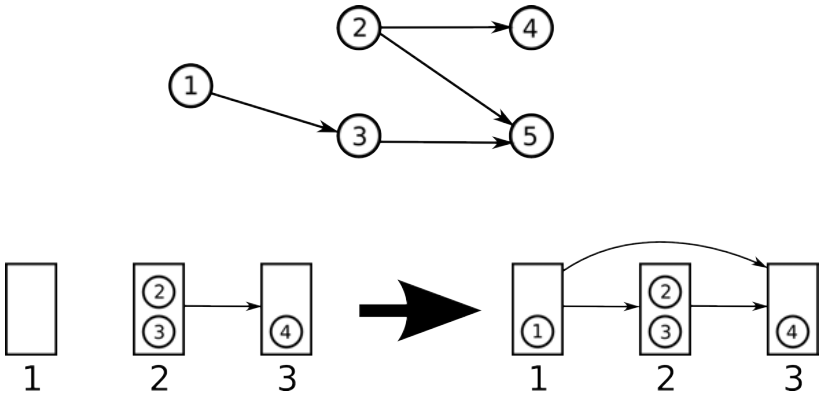

Figure 1. Worker dependencies after assignment of task 1 to worker 1

assignment_is_valid(t,w), used in *branch_tasks*, which verifies if a task t can be assigned to worker w in the current partial solution. If a loop in the precedence graph of workers is created by the insertion of this new assignment, the solution is invalid. In other words, if there is an arc from task t to task u in the dependencies graph of tasks, the precedence graph of workers must not have an arc from a_u to a_t. Also, if a worker $v \neq w$ is assigned to both a follower and a predecessor of t, the assignment is invalid. This two verifications are performed in time $O(|T|)$ in the worst case.

4.4. Priority rules

Selecting a task is an important part of this algorithm. If the tasks are selected in a correct order, the branch-and-bound tree will be tightened, decreasing the number of nodes and, consequently, the execution time of the procedure.

Suppose we select a task t, and $V_t = \{w \in W \mid assignment\_is\_valid(t,w)\}$ is the set of feasible workers for t. Since the workers for which the assignment of t generates a local lower bound greater than the upper bound can be pruned after branching, they are removed from V_t, in the search tree. In this way, $|V_t|$ defines the maximum number of children of the current node if task t is selected. Thus, a local heuristic to decide which task will generate the smallest branch-and-bound tree is to select the task with lowest $|V_t|$. As a consequence, tasks with only one valid worker will be assigned imediately, forcing dependencies among them to be satisfied earlier in the tree.

The branch-and-bound procedure sorts the workers that can be assigned to the chosen task t according to another priority rule and visits the branches based on this order. The solution applied in this paper uses the value of the local lower bound with the new assignment of task t to a worker w, i.e, we use a best bound strategy for choosing the next branch. Thus, the algorithm will visit first the branches that can lead to better results, trying to find the optimal solution early during the search.

5. Computational Results

Two methods will be compared in this section: the mixed integer programming model from Miralles et al. (2008) solved by a standard solver and the proposed branch-and-bound algorithm. The branch-and-bound method also proposed by (Miralles et al., 2008) is not analysed because no time results for comparation were presented in that paper.

The main quantity of interest when comparing exact algorithms is their execution time. Thus, these two methods were implemented and executed on identical machines, to make their execution time comparable.

The two methods were implemented using C++. The MIP model of Section 3 were executed using the CPLEX 12.4 library. The same library solved the linear relaxation of the model, used as lower bound in the branch-and-bound algorithm. The experiments were conducted on a PC with a 2.8 GHz Core i7 930 processor and 3GB of 1333 MHz DDR3 memory, running a 64 bit Ubuntu Linux.

In the literature, 320 instances of ALWABP are mainly used as a benchmark. 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. Internally, each of the instance types is also divided in eight groups of ten instances, based on three parameters: the number of workers, the variability of the task execution times [2] (Var) and the percentage of infeasibilities (Inf). For each of these parameters, a low level and a high level are defined, as shown in Table 2.

Table 2. Parameter levels for ALWABP

Parameter	Low Level	High Level
$\|T\|$	25-29	70-75
$\|W\|$	$\|T\|/7$	$\|T\|/4$
OS	20% - 25%	55% - 75%
Inf	10%	20%
Var	L1	H3

Roszieg and Heskia are the sets of instances with a low number of tasks, and order strength equal to $71,67\%$ and $22,49\%$ respectively. The other two groups have a high number of tasks 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.

Two metrics were collected from the analysed methods. The first is the runtime of the procedure, which is the main comparison metric. The second is the number of branch-and-bound nodes visited during the search. This metric has an almost direct relationship with the runtime, but is independent of machine specifications and can be reproduced more easily than the runtime metric.

For each group of ten instances of Roszieg and Heskia, the average result for the two metrics is calculated and shown in Table 3.

In other hand, neither of the two methods could solve the larger instances, Tonge and Wee-Mag, in acceptable time. The ten first test cases of these sets were executed by the two methods, but no instance could be solved in less than one hour.

Seven branch-and-bound executions of Tonge instances achieved the optimal solution in less than four hours, but no Wee-Mag instance was solved by this method in this amount of time. The MIP model solution by CPLEX had even worse results and solved only one Wee-Mag instance in less than four hours. This corroborates the findings of Chaves (2009).

The proposed branch-and-bound method outperforms the MIP model in runtime for every set of parameters tested, as presented in the Table 3. The total execution time of all Roszieg and Heskia instances is 13.3 seconds in the branch-and-bound method against more than 37 minutes in the MIP model solution, using a standard solver.

The number of nodes, on the other hand, is smaller in the CPLEX search tree, with excep-

[1] The percentage of precedence relations of all possible relations $|T|\cdot(|T|-1)/2$ present in the instance.

[2] Based on a reference time t_i for each task, the task times are randomically selected in $[1,t_i]$ (L1) or $[1,3t_i]$ (H3). This interval defines the variability of task times.

Table 3. Results for Roszieg and Heskia

				Branch-and-Bound		MIP Model	
Instance	$\|W\|$	Var	Inf	Runtime (s)	Nodes	Runtime (s)	Nodes
Roszieg	4	L1	10%	0.0660	681.1	3.4194	56.9
			20%	0.0500	161.6	2.8039	1.1
		H3	10%	0.0776	1150.0	3.6433	156.6
			20%	0.0676	647.4	3.9991	82.9
	6	L1	10%	0.2247	6852.0	24.9134	2715.0
			20%	0.1142	1723.6	28.5800	2601.3
		H3	10%	0.2156	6392.7	29.9071	3467.0
			20%	0.1567	3903.8	23.0671	2785.0
Heskia	4	L1	10%	0.0332	551.0	2.0114	0.0
			20%	0.0334	546.1	2.0602	25.0
		H3	10%	0.0505	1354.7	2.5472	65.0
			20%	0.0548	1364.2	2.0483	24.3
	7	L1	10%	0.0490	210.4	26.7348	1535.9
			20%	0.0377	202.7	20.1038	1174.1
		H3	10%	0.0457	192.2	25.8845	1677.8
			20%	0.0543	277.0	25.6675	1344.1

tion of five parameter sets, four of them in the Heskia instances. In these, the low order strength of Heskia strenghtens the lower bounds, because they do not consider the dependencies among the tasks. This, together with the better distribution of tasks on more workers, decreases the values of valid solutions. Thus, with better lower and upper bounds, the algorithm will produce more cuts, tightening the branch-and-bound tree for this set of instances. Besides that, the longer execution time of CPLEX is explained by the preprocessing techniques and by the intra-node processing applied by the solver.

6. Conclusion and future work

In this paper we have proposed a new task-oriented branch-and-bound method for solving the assembly line worker assignment and balancing problem. This algorithm has been found to outperform a standard solver on the mixed integer programming model defined in the literature in a series of computational tests, although the larger instances of the existing set of benchmarks are not yet solved in acceptable execution time.

Further research is needed for exactly solving these larger instances. The MIP model can be improved, decreasing the number of variables and creating new constraints, accelerating the CPLEX solution of the model. Another possibility of improvement is to use better lower and upper bounds in the branch-and-bound, which will prune the branch-and-bound search tree. Also, a strategy similar to the algorithm described in Klein and Scholl (1996) applied to the ALWABP may produce good results for larger instances. This is a station-oriented branch-and-bound method with stations filled according to the current lower bound, i.e. the runtime of the algorithm depends more on the optimal cycle time and less on the number of tasks. Thus, large instances, with small optimal cycle times may take advantage of such a procedure.

References

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

Chaves, A., Lorena, L., and Miralles, C. (2009), Hybrid metaheuristic for the assembly line worker assignment and balancing problem, In *Hybrid Metaheuristics*, volume 5818 of *Lecture Notes in Computer Science*, pages 1–14, Springer Berlin / Heidelberg.

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

Chaves, A. A., Miralles, C., and Lorena, L. A. N. (2007), Clustering search approach for the assembly line worker assignment and balancing problem, *Proceedings of ICC&IE*, pages 1469–1478.

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

Miralles, C., García-Sabater, J. P., Andrés, C., and Cardos, M. (2007), Advantages of assembly lines in sheltered work centres for disabled. A case study, *International Journal of Production Economics*, 110(1-2):187–197.

Miralles, C., García-Sabater, J. P., Andrés, C., and Cardós, M. (2008), Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled, *Discrete Applied Mathematics*, 156(3):352–367.

Moreira, M., Ritt, M., Costa, A., and Chaves, A. (2012), Simple heuristics for the assembly line worker assignment and balancing problem, *Journal of Heuristics*, 18(3):505–524.

Moreira, M. C. O. and Costa, A. M. (2009), A minimalist yet efficient tabu search for balancing assembly lines with disabled workers, *Anais Do XLI Simpósio Brasileiro de Pesquisa Operacional, Porto Seguro*.

Organization, W. H. (2011), World report on disability, *Geneva: WHO*.

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

Scholl, A. and Klein, R. (1997), SALOME: A bidirectional branch-and-bound procedure for assembly line balancing, *INFORMS Journal on Computing*, 9(4):319–334.

Scholl, A. and Voß, S. (1997), Simple assembly line balancing - Heuristic approaches, *Journal of Heuristics*, 2(3):217–244.

Sprecher, A. (1999), A competitive branch-and-bound algorithm for the simple assembly line balancing problem, *International Journal of Production Research*, 37(8):1787–1816.

Sprecher, A. (2003), Dynamic search tree decomposition for balancing assembly lines by parallel search, *International Journal of Production Research*, 41(7):1413–1430.