BIN PACKING PROBLEM WITH GENERAL PRECEDENCE CONSTRAINTS

Raphael Kramer, Mauro Dell'Amico, Manuel Iori
Department of Sciences and Methods for Engineering
University of Modena and Reggio Emilia, Italy
{raphael.kramer, mauro.dellamico, manuel.iori}@unimore.it

ABSTRACT

In this paper we study the bin packing problem with general precedence constraints, in which a set of weighted items has to be packed in the minimal number of capacitated bins, while satisfying precedence relationships among pair of items. The problem generalizes the well-known simple assembly line balancing problem type-1, and models relevant real-world situations. To solve the problem we propose a series of lower and upper bounding techniques, including a large neighborhood search algorithm. Preliminary computational results show the efficiency of the proposed approach in solving complex instances.

KEY WORDS. Bin Packing Problem, Simple Assembly Line Balancing problem, General Precedence Constraints, Lower Bounds, Large Neighborhood Search.

Main areas: Combinatorial Optimization. Metaheuristics. Mathematical Programming.

RESUMO

Neste artigo estudamos o problema de empacotamento em caixas com restrições gerais de precedência, o qual um conjunto ponderado de itens devem ser empacotados em um número mínimo de caixas, satisfazendo as relações de precedências entre pares de itens. Tal problema generaliza o conhecido problema de balanceamento de linhas de montagem tipo-1, e também problemas práticos. Para resolver o problem, um conjunto de técnicas para geração de limites inferiores e superiores são propostos, bem como um algoritmo de busca em vizinhança ampliada. Resultados computacionais preliminares mostram a eficiência da abordagem proposta na resolução de problemas complexos.

PALAVRAS CHAVE. Problema de Empacotamento em Caixas, Problema de Balanceamento de Linhas de Montagem, Restrições Gerais de Precedência, Limites Inferiores, Busca em Vizinhança Ampliada.

Áreas Principais: Otimização Combinatória. Metaheurísticas. Programação Matemática.

1 Introduction and Related Problems

This paper presents an extension to the *Bin Packing Problem* (BPP), named as BPP with general precedence constraints (BPP-GP). The BPP is one of the most studied problems in combinatorial optimization. It requires to pack a set N of one-dimensional items, each of them having width w_j, into the minimum number of identical bins of capacity C. The *cutting stock problem* (CSP) is the BPP version in which all items having the same width are grouped together. Both problems have been largely studied, especially starting from the early 1960's (see, e.g., Kantorovich, 1960; Gilmore and Gomory, 1961, 1963).

In addition to its theoretical importance, the BPP is also interesting from a practical standpoint, commonly appearing in real-life situations in several different areas, such as logistics, computer science, industry and others (see, e.g., Waescher *et al.*, 2007). For recent surveys on the BPP and the CSP, the reader is referred to Valério de Carvalho (2002) (linear programming methods), Coffman *et al.* (1999) (upper bounds), Clautiaux *et al.* (2010) (lower bounds) and Delorme *et al.* (2015) (exact algorithms and computational results).

A *Generalized BPP* (G-BPP), was introduced by Garey *et al.* (1976). In the G-BPP items and bins have $s \geq 1$ dimensions, i.e., each item j has weights $w_j^1, w_j^2, \ldots, w_j^s$ and each bin has capacities $C^1, C^2, \ldots, C^s$. Moreover a set of *precedence constraints* is imposed among the items. A precedence constraint between items j and k states that the index i_k of the bin containing item k should be strictly greater than the index i_j of the bin containing item j, i.e., or $i_k - i_j \geq 1$. The G-BPP also extends the *BPP with precedence constraints* (BPP-P), which is the G-BPP case in which $s = 1$.

Some works dealing with the BPP-P can be found in Schaus (2009), Augustine *et al.* (2009) and Dell'Amico *et al.* (2012). Schaus (2009) handled the problem by means of constraint programming, Augustine *et al.* (2009) studied the BPP-P as a special case of the *strip packing problem with precedence constraints*, while Dell'Amico *et al.* (2012) proposed a branch-and-bound algorithm (together with some reduction criteria, lower bounds and upper bounds) to address the problem with exact methods for the first time in the literature.

A very similar problem to the BPP-P is the *Simple Assembly Line Balancing Problem Type 1* (SALBP-1)[1]. The SALBP-1 aims to assign tasks to workstations such that all precedence constraints are fulfilled, the station time does not exceed the cycle time, and the number of workstations is minimized. In the BPP notation, the workstations correspond to the bins, the jobs to the items and the cycle time to the capacity of the bins. In terms of mathematical formulation, the only difference to the BPP-P is related to the precedence constraints: in the SALBP-1 an item/task can be placed in the same bin/station of its precedent item/task, i.e., $i_k - i_j \geq 0$. For a literature review on problems and methods for assembly line balance problems we refer the interested reader to the surveys by Becker and Scholl (2006) and Scholl and Becker (2006).

Finally, the *BPP with general precedence constraints* (BPP-GP), as well as the previous presented problems aims to minimize the number of bins used to pack a set of items, but extends the BPP-P and the SALBP-1 in such way that the "distance" between items can assume different positive integer values, i.e., $i_k - i_j \geq t_{ij}$, with $t_{jk} \in \mathbb{Z}_0^+$. In particular, the BPP-GP model those situations where the distance between two operations must be large enough to guarantee structural properties of the products. This is a common requirement arising in many industrial problems (see, e.g., Tirpak, 2008). A graphical illustration of some generalization problems for the BPP is presented in Figure 1. Figure 2 shows an example of a precedence graph for the BPP-GP. Two possible solutions respecting the precedence relationships of Figure 2 can be visualized in Figure 6.

Given that the mathematical formulation for the BPP-GP (presented in Section 2) cannot

[1]An assembly line consists of a set of workstations disposed along a conveyor belt where (sub)items are transported. Each workstation performs the same activities and has a limited cycle time. In a simple assembly line only one particular homogeneous product is assembled.

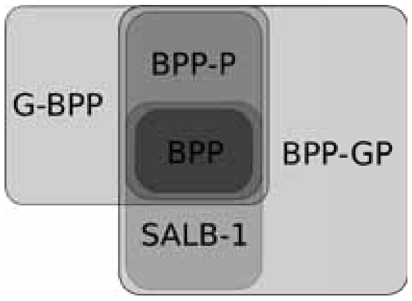

Figure 1: Illustration of generalization problems for the BPP

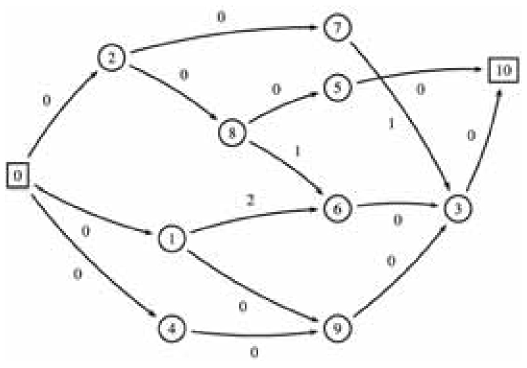

Figure 2: Example of a precedence graph for the BPP-GP

handle with large size instances (for most of the largest tested instances, any solution is generated in 600 seconds), we propose a hybrid algorithm based on local search and large neighborhood search, in order to generate good solutions in a reasonable computational time. Moreover, some lower bound procedures are used to verify whether such solutions are optimal (stopping the algorithm as soon as an optimal solution is found). Computational experiments show that the proposed algorithm is able to generate very good solutions, of which 70.19% of 2100 tested instances are proved to be optimal.

The remainder of this paper is organized as follows. Section 2 formally defines the BPP-GP. Section 3 presents the algorithms used to generate good lower bounds and upper bounds. Computational results are provided in Section 4, while Section 5 concludes.

2 Problem Description

We study the *Bin Packing Problem With General Precedence Constraints* (BPP-GP), which is a generalization of the well-known *Bin Packing Problem* (BPP). In the BPP-GP we are given a set N of n items, each of them with weight w_j, a set M of m identical bins, each of them having capacity C, and a set A of precedence relationships, $j \prec k$, with values $t_{jk} \geq 0$, giving the minimum distance required between the bin allocating item j and the bin allocating item k. The BPP-GP asks for minimizing the number of bins used to allocate all items respecting the precedence relationships.

Defining $head_j$ as the minimum number of bins that precede j, and $tail_j$ as the minimum number of bins that succeed j, it is possible to compute the minimum and the maximum indices of a bin where j can be placed[2]. The computation of such values is very useful for the development of the lower bounds presented in Section 3. Let the bins be numbered from 1 to m. By introducing

[2]These values can be computed by solving a longest path problem (e.g. using Dijkstra's algorithm (Dijkstra, 1959)).

binary variable y_i that takes value 1 if bin i is used and 0 otherwise, and a binary variable x_{ij} that takes value 1 if item j is allocated to bin i, 0 otherwise, the BPP-GP can be modeled as the following *Integer Program* (IP).

$$\min \quad \sum_{i \in M} y_i \tag{1}$$

$$s.t. \quad \sum_{j \in N} w_j x_{ij} \leq C y_i \qquad i \in M \tag{2}$$

$$\sum_{i \in M} x_{ij} = 1 \qquad j \in N \tag{3}$$

$$\sum_{i \in M} i x_{ik} \geq \sum_{i \in M} i x_{ij} + t_{jk} \qquad (j,k) \in A \tag{4}$$

$$x_{ij} = 0 \qquad i \in M, j \in N : i < head_j \tag{5}$$

$$y_i \geq y_{i+1} \qquad i = 1, \ldots, m-1 \tag{6}$$

$$y_i \in \{0,1\} \qquad i \in M \tag{7}$$

$$x_{ij} \in \{0,1\} \qquad i \in M, j \in N. \tag{8}$$

The objective function (1) seeks to minimize the number of used bins. Constraints (2) guarantee that if a bin is used, then the sum of the weights of the items allocated to that bin does not exceed the bin capacity. Equation (3) states that all items must be assigned to a bin. Constraints (4) ensure that, if there is a precedence between items j and k, then the index of the bin where k is packed must be greater than or equal to the index of the bin receiving item j plus the value of the precedence relationship. Constraints (5) prohibit that an item k for which there exists a precedence relationship $(j,k) \in A$ of value $t_{jk} > 0$ is allocated to one of the first t_{jk} bins. Constraints (6) impose that a bin cannot be used if its predecessor is not also used. Finally, constraints (7) and (8) impose variables to be binary.

3 Solution Algorithms

The mathematical formulation (1)-(8) introduced in the previous section is not able to solve large size instances. Thus, to solve the BPP-GP we developed a set of lower and upper bounding procedures. Subsection 3.1 details five procedures for generating good lower bounds (LB), while Subsection 3.2 presents an algorithm based on two local search operators, and a large neighborhood search that iteratively solves restricted mathematical models to generate improved solutions.

3.1 Lower Bounds

Concerning lower bounds, we generalized those proposed by Dell'Amico *et al.* (2012) for the BPP-P, by taking into account the new precedence constraints. These lower bounds are either based on classical techniques originally developed for the BPP, including dual feasible functions, or on computation of the *longest path* on the acyclic graph imposed by the precedences. They also make use of combinations of these two topics (BPP and longest path) that are tailored for the BPP-GP. In total, five lower bound (LB) procedures were implemented, whose descriptions are presented below:

- **LB 1**: By removing the precedence constraints, a trivial relaxation can be obtained by rounding up the optimal solution of the linear relaxation, i.e., $LB1 = \lceil \sum_{j=1}^{n} w_j / C \rceil$.
- **LB 2**: By adding two dummy nodes, 0 and $n+1$, to the precedence graph (with precedence values equal to 0), such that 0 precedes all nodes and $n+1$ has all nodes as predecessors, a better bound is obtained by computing the longest path from 0 to $n+1$.

- **LB 3**: An immediate bound is the maximum of the previous two, i.e., $LB3 = max\{L1, L2\}$.
- **LB 4**: First define P as the subset of items in the longest path from 0 to $n+1$, $Q = N \setminus P$, and B^P as the set of bins used by the items in P. Compute S' as the maximum (fractional) weight of items $j \in Q$ that can be packed in the bins in set B^P, by respecting the precedence and capacity constraints (the value of S' can be obtained by solving a standard *max-flow procedure* from Q to B^P). Then our fourth lower bound is computed as follows:
$$LB4 = LB2 + \left\lceil \frac{\sum_{j \in Q} w_j - S'}{C} \right\rceil \tag{9}$$
- **LB 5**: From the definition of $head_j$ and $tail_j$, this lower bound is computed as follows. Let $S_{r,q}$ be a subset of items $j \in N$ such that $head_j \geq r$ and $tail_j \geq q$, an improved bound can be obtained by computing a lower bound for the subset $S_{r,q}$ adding r and q, as expressed in Equation (10), where L_α is any valid BPP-GP lower bound.
$$LB5(L_\alpha) = \max_{\substack{r,q = 1,2,\ldots,c_{0,n+1}-1 \\ r+q \leq c_{0,n+1}}} \{L_\alpha(S_{r,q}) + r + q\} \tag{10}$$

For more details about the presented lower bounds and their worst-case analysis the reader is referred to the paper of Dell'Amico *et al*. (2012).

3.2 Upper Bounds

To generate good upper bounds, an algorithm based on Local Search (LS) and *Large Neighborhood Search* (LNS) is proposed. In LNS, an initial solution is gradually improved by means of ruin-and-recreate moves, destroying and repairing the solution (Pisinger and Ropke, 2010).

3.2.1 Constructive Procedures

An initial solution is constructed according to two independent procedures: the first one invokes a *first fit algorithm* (FFA), while the second one consists of solving a series of subset sum (SS) problems. FFA assigns one item at a time to the first feasible bin, if any, or opens a new bin otherwise. This process stops when all items are allocated. On the other hand, the SS problem looks for maximizing the load of each bin. That is, of all possible assignments of items (respecting the capacity constraint and the precedence constraints) it chooses the one for which the resulting bin load is the maximum one. The max subset sum heuristic iteratively adds new bins to the partial solution until all items have been assigned.

3.2.2 Local Search

Once the initial solution is generated, it is optimized by a local search procedure composed by three neighborhood operators, described below:

- *Relocate* attempts to move an item from its current bin to another;
- *Swap* attempts to exchange the position of two items;
- *Push* can be viewed as a two-step *Relocate* operator. When the relocation of an item i from a bin k to a bin l violates the capacity of l, it tries to relocate all items in bin l to another bin, in order to enable the move.

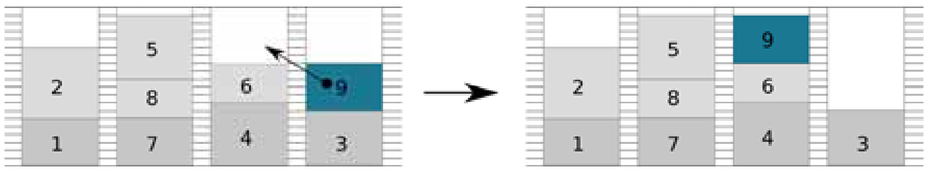

Figure 3: Example of a *Relocate* move

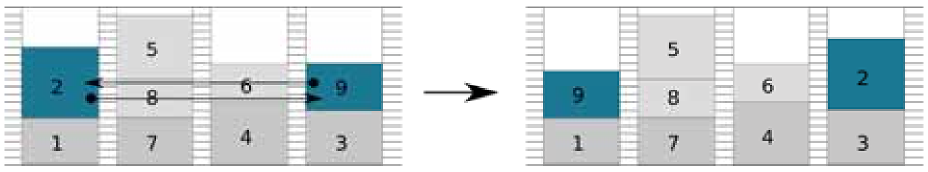

Figure 4: Example of a *Swap* move

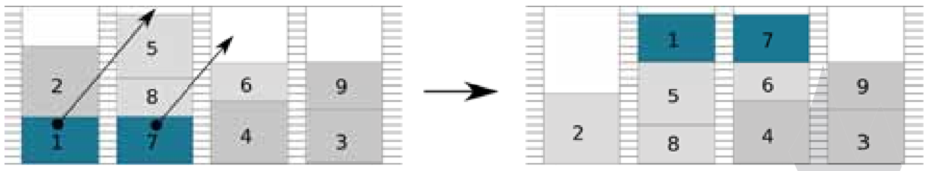

Figure 5: Example of a *Push* move

Both operators try to minimize the minimum load among the bins, with the aim of producing the most unbalanced situation. The move evaluation is computed according to Equation (11), where l_i and l'_i refer to the load of a bin i before and after the evaluated move, respectively, and M' refers to the set of modified bins. If Δ is lower than 0, such move is stored in a pool of moves. After evaluating all possible moves in the neighborhoods *Relocate*, *Swap*, and *Push* the pool are sorted in a non-decreasing order based on the value of Δ.

$$\Delta = (\sum_i l_i^2) - (\sum_i {l'}_i^2), \forall i \in M' \tag{11}$$

Once the pool is sorted, the current solution is modified iteratively starting from the best move. Note that after the first modification, the value Δ of some moves becomes inaccurate. To avoid a worsening of the solution, such moves needs to be checked if they still lead to improvements; in negative case, they are ignored. If the minimum load of the last bin becomes equal to zero, then the best solution is eventually updated.

3.2.3 Large Neighborhood Search

If the solution found after the local search is not proven to be optimal (by comparison with the lower bound), then the LNS procedure is started. This algorithm iteratively destroys the current solution by (i) emptying completely a bin, and (ii) removing some items from the residual subset of bins. Both the bin and the items are selected randomly. The solution is then re-completed by re-inserting the removed items solving a mathematical model based on that presented in Section 2. A graphical illustration of this strategy is presented in Figure 6.

Let n_δ be a parameter that defines the number of residual items to be unchanged in the new solution, $b(j)$ be the index of the bin containing item j, r be the index of the bin to be removed, and z_{cur} be value of the current solution. Then, an IP model is formulated as (12)-(20) and solved to optimality (if a feasible solution exists). This formulation is basically the same as that presented in Section 2 with two modifications: Constraint (3) is replaced by (14) in such way that at most z_{cur-1} bins are used; and Constraint (16) is added to impose a fixed number of items to be unchanged, also shifting the items placed after the removed bin r to one bin before.

$$\min \quad \sum_{i\in M} y_i \tag{12}$$

$$s.t. \quad \sum_{j\in N} w_j x_{ij} \le C y_i \qquad i \in M \tag{13}$$

$$\sum_{i=1}^{z_{cur-1}} x_{ij} = 1 \qquad j \in N \tag{14}$$

$$\sum_{i\in M} i x_{ik} \ge \sum_{i\in M} i x_{ij} + t_{jk} \qquad (j,k) \in A \tag{15}$$

$$\sum_{j:b(j)<r} x_{b(j),j} + \sum_{j:b(j)>r} x_{b(j)-1,j} \ge n_\delta \tag{16}$$

$$x_{ij} = 0 \qquad i \in M, j \in N : i < head_j \tag{17}$$

$$y_i \ge y_{i+1} \qquad i = 1,\dots,m-1 \tag{18}$$

$$y_i \in \{0,1\} \qquad i \in M \tag{19}$$

$$x_{ij} \in \{0,1\} \qquad i \in M, j \in N. \tag{20}$$

This procedure of destruction-and-repair of solutions is iteratively repeated until a proven optimal solution for the BPP-GP is found or a maximum time limit is reached.

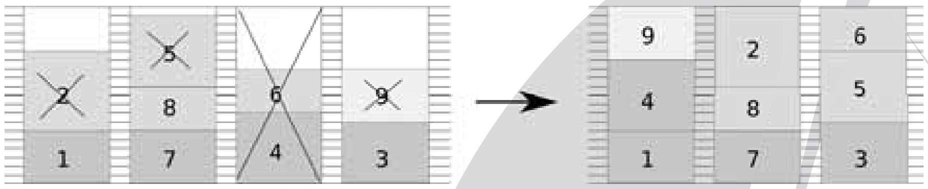

Figure 6: Solution updating with LNS

3.3 General overview of the proposed algorithm

The algorithms described in Sections 3.1 and 3.2 were integrated into an iterated local search (ILS) metaheuristic, named as *FFSS-LS-LNS*, as showed in Algorithm 1. Let $MaxIter$ be a parameter that defines the number of trials to escape from a local optimal solution.

Algoritmo 1 FFSS-LS-LNS

```
1: $Iter \leftarrow 0$
2: $lb \leftarrow computeLowerBounds()$
3: $sol \leftarrow generateInitialSolution()$
4: $bestSol \leftarrow sol$
5: enquanto $Iter < MaxIter$ faça
6:     $sol' \leftarrow localSearch(sol)$
7:     se $sol' < bestSol$ então
8:         $bestSol \leftarrow sol'$
9:     senão
10:        $sol' \leftarrow bestSol$
11:    $sol \leftarrow perturbation(sol')$
12:    $Iter \leftarrow Iter + 1$
13: $sol \leftarrow largeNeighborhoodSearch(bestSol)$
14: retorne $sol$
```

FFSS-LS-LNS firstly computes the maximum lower bound from procedures presented in Sec. 3.1 and stores it in lb. Procedure `generateInitialSolution` builds an initial solution executing, independently, the First Fit Algorithm and the Subset Sum procedure presented in Sec 3.2.1, keeping the best one solution sol. Then, the ILS procedure starts performing $MaxIter$ iterations of local searches and perturbations.

Procedure `localSearch(sol)` computes the cost of all neighborhood solutions (defined by the operators presented in Sec. 3.2.2) of sol storing that ones which leads to an improvement on the current best solution $bestSol$ in a list of candidate moves. Such list is sorted in decrease order of improvement, and each move is executed sequentially, starting from the best one, always verifying if the candidate move still leads to an improvement. This lists of moves is rebuild iteratively, until no more improvement be possible. Once an local optimal solution sol' is found, the procedure `perturbation(sol')` is invoked. It consists of random Relocate and Swap moves, that generates a new (feasible) solution but without seeking for improvements. When $Iter$ reaches $MaxIter$, the ILS procedure finishes and, then, a Large Neighborhood Search (Sec. 3.2.3) is performed on $bestSol$. After that, the algorithm *FFSS-LS-LNS* returns the best found solution sol.

4 Preliminary Computational Results

To test our algorithms, we generate a new set of instances (2100 in total) based on those of Otto *et al.* (2011): items weight, bins capacity and precedence graph are the same. However, the precedence values were modified: instead of cost 0 for every arc, now they varies between 0 and 1. Such values were randomly generated with uniform probability. All algorithms were implemented in C++ and run with a time limit of 5 minutes on an Intel Xeon E5530 2.4 GHz with 23.5 GB of memory, under Linux Ubuntu 12.04.1 LTS 64-bit. The IP models were solved by CPLEX 12.

Table 1 reports, for each class of instance size (label n), the total number of problems in the set (label #inst.), the bins capacity (label C), and the number of unsolved instances (label *uns.*), the percentage gap of the unsolved instances (label *gap%*), computed as (UB-LB)/LB*100, and the average running time in CPU seconds, for each method. The label "n.a." indicates that no solution was obtained by IP within the given time limit. The last line gives the total number of unsolved instances by FFSS-LS-LNS and by IP. In order to verify the possible optimality of FFSS-LS-LNS solutions, they were compared with the best known lower bound from the procedures presented in Section 3 (when computing $LB5$ we used $L_\alpha = LB2$) and from the IP solver at the end of optimization.

FFSS-LS-LNS and IP can easily solve the small ($n = 20$) instances. Yet, FFSS-LS-LNS failed in two cases and needed more time to find their best solutions. However, for larger instances, FFSS-LS-LNS presents better results. For medium instances ($n = 50$), FFSS-LS-LNS solves nine

Table 1: Computational results on BPPGP instances with precedences values $\in \{0, 1\}$.

			IP model (1)-(8)			FFSS-LS-LNS		
n	*#inst.*	C	*uns.*	*gap%*	*sec.avg.*	*uns.*	*gap%*	*sec.avg.*
20	525	1000	0	0.00	0.21	2	7.74	53.28
50	525	1000	94	4.15	120.80	85	4.65	92.78
100	525	1000	230	4.61	269.17	209	5.10	143.15
1000	525	1000	525	n.a	602.17	330	4.28%	285.34
Tot	**2100**		**849**			**626**		

CPU time limits: IP-600 s; FFSS-LS-LNS-300 s.

more instances than IP with a lower computational time. FFSS-LS-LNS obtains a certificate of optimality for 440 solutions, and the gap of the unsolved instances is around 4% for both methods.

For large instances ($n = 100$), FFSS-LS-LNS and IP solved to optimality more than half of the instances, and the gap stills around 5%. The computational time of FFSS-LS-LNS is two orders of magnitude smaller, as well as for the very large instances ($n = 1000$). When solving the largest set of instances, it is verified that CPLEX cannot even generate a feasible solution.

Summarizing, FFSS-LS-LNS was able to solve 1474 instances, within computational times lower than 300 seconds. When solving small instances, CPLEX is more indicated, but when the size of instances increases FFSS-LS-LNS becomes more attractive.

5 Conclusions

We presented a generalization of the Bin Packing Problem with precedence constraints in which the precedence values between items can assume different integer-positive values. We first presented some new lower bounding procedures. Then, to generate good feasible solutions in a reduced computational time, an algorithm based on local search and on a Large Neighborhood Search procedure was implemented. Computational experiments showed that the proposed algorithms are capable of generating good results in terms of computational time and quality of solutions.

Acknowledgements

This research was partially supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES/Brazil), grant PVE n. A007/2013, and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq/Brazil), grant GDE n. 201222/2014-0.

References

Augustine, J., Banerjee, S. and Irani, S. (2009), Strip packing with precedence constraints and strip packing with release times. *Theoretical Computer Science*, v. 410, p. 3792–3802.

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

Clautiaux, F., Alves, C. and Valério de Carvalho, J. (2010), A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, v. 179, n. 1, p. 317–342.

Coffman, E. G., Galambos, G., Martello, S. and Vigo, D. Bin packing approximation algorithms: Combinatorial analysis. Du, D.-Z. and Pardalos, P. (Eds.), *Handbook of Combinatorial Optimization*, p. 151–207. Springer US, 1999.

Dell'Amico, M., Diaz-Diaz, J. and Iori, M. (2012), The bin packing problem with precedence constraints. *Operations Research*, v. 60, n. 6, p. 1491–1504.

Delorme, M., Iori, M. and Martello, S. Bin packing and cutting stock problems: Mathematical models and exact algorithms. Technical Report OR-15-1, University of Bologna, available on line at `http://or.dei.unibo.it/library/bpplib`, 2015.

Dijkstra, E. (1959), A note on two problems in connexion with graphs. *Numerische Mathematik*, v. 1, n. 1, p. 269–271.

Garey, M., Graham, R., Johnson, D. and Yao, A. C.-C. (1976), Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, v. 21, n. 3, p. 257 – 298.

Gilmore, P. C. and Gomory, R. E. (1961), A linear programming approach to the cutting-stock problem. *Operations Research*, v. 9, n. 6, p. 849–859.

Gilmore, P. C. and Gomory, R. E. (1963), A linear programming approach to the cutting stock problem–part ii. *Operations Research*, v. 11, n. 6, p. 863–888.

Kantorovich, L. V. (1960), Mathematical methods of organizing and planning production. *Management Science*, v. 6, n. 4, p. 366–422.

Otto, A., Otto, C. and Scholl, A. A systematic data generator for (simple) assembly line balancing. Relatório técnico, School of Economics and Business Administration, 2011.

Pisinger, D. and Ropke, S. Large neighborhood search. Gendreau, M. and Potvin, J.-Y. (Eds.), *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, p. 399–419. Springer US, 2010.

Schaus, P. *Solving Balancing and Bin-Packing problems with Constraint Programming*. Tese de doutorado, Université Catholique de Louvain, Département d'Ingénierie Informatique, Louvain-la-Neuve, Belgium, 2009.

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*, v. 168, p. 666–693.

Tirpak, T. (2008), Developing and deploying electronics assembly line optimization tools: A motorola case study. *Decision Making in Manufacturing and Services*, v. 2, p. 63–78.

Valério de Carvalho, J. M. (2002), LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, v. 141, n. 2, p. 253 – 273.

Waescher, G., Haussner, H. and Schumann, H. (2007), An improved typology of cutting and packing problems. *European Journal of Operational Research*, v. 183, p. 1109–1130.