Optimization of surface utilization using heuristic approaches

Y. Langer[1], M. Bay[1], Y. Crama[1], F. Bair[2], J. D. Caprace[2] & Ph. Rigo[2]
[1]Department of Operations Research and Production Management, HEC-Business School of the University of Liège, Belgium
[2]Department of Naval Architecture, University of Liège, Belgium

Abstract

In this paper, we present a scheduling problem that arises in factories producing large building blocks (in our case, a shipyard workshop producing prefabricated keel elements). The factory is divided into several equally size areas. The blocks produced in the factory are very large, and, once a building block is placed in the factory, it cannot be moved until all processes on the building block are finished. The blocks cannot overlap. The objective is to maximize the number of building blocks produced in the factory during a certain time window. To solve this problem, we propose heuristics inspired by techniques initially developed for the three-dimensional bin packing problem, since constraints for both problems are quite similar. Starting from an unfeasible solution, where blocks can overlap, a Guided Local Search (GLS) heuristic is used to minimize the sum of total overlap. If a solution with zero overlap is found, then it is a feasible solution; otherwise the block with the biggest overlap is removed and the procedure is restarted. The GLS algorithm has been improved by Fast Local Search (FST) techniques in order to speed up convergence to a local minimum. Additionally, neighborhoods are restricted to their smallest size so as to allow their evaluation in polynomial-time. In a last step, we explain the additional real-life issues arising in the industrial application and how firm-specific constraints can be conveniently considered by the model.
Keywords: scheduling, simulation, surface utilization, Guided Local Search, three-dimensional bin-packing, building blocks, shipyard.

1 Introduction

The aim of this paper is to present a new method to solve a scheduling problem that arises in factories producing large building blocks (in our case, a shipyard workshop producing prefabricated keel elements). The factory is divided in equal size rectangular areas. The blocks produced in the factory are very large, and, once a building block is placed into the factory, it cannot be moved until all processes on the building block are finished. The blocks cannot overlap. The objective is to maximize the number of building blocks produced in the factory during a certain time window.

More precisely, we are given a set of n rectangular-shaped blocks. Each block is characterized by its geometric dimensions (width w_j, length l_j and height h_j) but also by processing information such as its processing time t_j, its ready time r_j and its due date d_j (j in $\{1, \quad , n\}$). We are also given a number A of identical two-dimensional areas (consequences of limited height are discussed in section 5), having width W and length L. Time is considered as a third dimension. The areas are fully dedicated to the production of the blocks.

The problem we are facing consists of orthogonally ordering the blocks into the areas, while respecting the time constraints, and with the objective to produce the largest number of building blocks. In practical terms, this means that we have to assign six variables for each block j:

- $p_j = \{0,1\}$ indicating whether the block j is produced or not;
- the name $a_j = \{1,\dots, A\}$ of the area where block j is to be produced;
- x_j and y_j coordinates, representing the position of the upper left corner of the block j in the area;
- an orientation $o_j = \{0,1\}$ (either horizontally or vertically) for block j;
- a starting date s_j.

A solution will be considered as feasible if the individual and the collective constraints are met. We call individual constraints those which are focusing on one block only, regardless of the other blocks. Major individual constraints represent the fact that:

- blocks must fit within the width of an area
  ($x_j \geq 0$ and $x_j + [\ o_j.w_j + (1-o_j).l_j\ ] \leq W$)
- blocks must fit within the length of an area
  ($y_j \geq 0$ and $y_j + [\ o_j.l_j + (1-w_j).l_j\ ] \leq L$)
- blocks must fit in their time windows
  ($s_j \geq r_j$ and $s_j + t_j \leq d_j$)

We will show in a further section how additional individual constraints are easily integrated.

Collective constraints focus on the interaction between the positions of different blocks. In a first step, the only collective constraint considered is that we need to prevent the blocks from overlapping.

We will also assume in the first sections, that there exists at least one feasible solution for the set of blocks initially given. In other words, this means that all the constraints can be satisfied when $p_j = 1$ for all j in $\{1,..., n\}$.

To explain the algorithm developed for the entire problem, we will show in section 3 the method that leads to one of these feasible solutions. Our technique, as explained in section 2, has been largely inspired from techniques initially developed for a very similar problem called the Three-Dimensional Bin Packing Problem. Section 4 considers the problem without the feasibility assumption, using the first method to assess whether a subset of blocks is feasible or not. Then, section 5 describes how additional real-life issues can be easily integrated.

2 Analogy to the 3D-BPP

In the Three Dimensional Bin Packing Problem (3D-BPP), we are given a set of n rectangular-shaped items, each characterized by width w_j, height h_j, and depth d_j (j in $\{1,\ldots, n\}$) and an unlimited number of identical three-dimensional containers (bins) having width W, height H, and depth D. The three-dimensional bin packing problem (3D-BPP) consists of orthogonally packing all the items into the minimum number of bins.

The major difference between 3D-BPP and our initial problem is that, in the former, items/blocks must fit into the container height ($z_j \geq 0$ and $z_j + h_j \leq H$), whereas they must fit into their time window in the latter ($s_j \geq r_j$ and $s_j + t_j \leq d_j$). (One can compare a bin in the 3D-BPP to a timeline of the two-dimensional representation of an area, which gives us a three-dimensional representation of the problem.) The differences between minimizing the number of bins and maximizing the number of blocks will not complicate the formulation, since we will assume, at least for the first part of this paper, that there exists at least one feasible solution for a fixed number of block and of areas/bins.

The 3D-BPP is strongly NP-hard. Indeed, it is a generalization of the well-known one-dimensional bin packing problem (1D-BPP), in which a set of n positive values w_j has to be partitioned into the minimum number of subsets so that the total value in each subset does not exceed a given bin capacity W. It is clear that 1D-BPP is the special case of 3D-BPP arising when $h_j = H$ and $d_j = D$ for all j in $\{1,\ldots,n\}$, and it has been proven (Coffman et al. [2] that the 1D-BPP is NP-Hard. For such difficult problems, one way to contain combinatorial explosion is to allow algorithms to reach fairly good solutions, without guaranteeing that the best possible solution is reached. Some of the best known methods which use this strategy are *local search* heuristics.

Several methods for solving the 3D-BPP have been compared. Faroe et al. proposed in 2003 a *"New Heuristic for 3D-BPP*" [3]. Their method offers a huge degree of flexibility so that it can be adapted to various additional constraints. Therefore it fits perfectly to the wording of our problem, and to most of the additional real-life issues described in section 5.

3 Finding feasible solutions

3.1 General approach

The local search heuristic proposed to find a feasible schedule strictly enforces the individual constraints only; in other words, all the solutions generated by the heuristic respect the constraints associated with each individual block. Then, penalties linked to the collective constraints are summed up in an objective function that is minimized. With no additional real-life collective constraints (see section 5), the objective function value of a given solution is the total pairwise overlap between the blocks. Therefore, with a randomly generated unfeasible solution where blocks can overlap, searching for a feasible solution is equivalent to minimizing the objective function, since an objective value of zero indicates that also the collective constraints are met. For any solution X, let $overlap_{ij}(X)$ be the overlap (in square meters days) between blocks i and j. The objective function can now be formulated as

$$f(X) = \sum_{i<j} overlap_{ij}(X)$$

Given a solution X, we can redefine the neighborhood υ(X) proposed by Faroe et al. [3] as *the set of all solutions that can be obtained by translating any single block along the coordinates axes and the timeline, or by a move to the same position in another area, or by a +/-° 90 degree rotation of a block around one of its four corners*. A neighbor of X is therefore constructed by assigning a new value to one of the variables x_j, y_j, s_j, a_j, o_j. It is clear that this definition of a solution space includes all feasible schedules and that there is a path of moves between every pair of solution.

A typical local search procedure proceeds by moving from the current solution X_p to a neighboring solution X_{p+1} in $\upsilon(X_p)$ whenever this move improves the value of the objective function. This may lead to two types of difficulties. First, the solution may settle in a local minimum (states which are better than all the neighbors but not necessarily the best possible). Several standard methods, such as the Simulated Annealing (Aarts and Korst [1]) or the Tabu Search (Glover [4]), exist to avoid this well-known shortcoming of local search procedures. Secondly, the neighborhood of any given solution may be quite large (even if continuous, variables like x_j, y_j or s_j can be discretized for practical purposes). Therefore, exploring the neighborhood to find an improving move can be very costly in computing time. To deal with these issues, we present in this paper an application of the *Guided Local Search* (GLS) heuristic, and its accompanying neighborhood reduction scheme called *Fast Local Search* (FLS).

3.2 Guided local search

The Guided Local Search Heuristic (GLS) has its root in a Neural Network architecture named GENET, developed by Wang and Tsang (1991), which is applicable to a class of problems known as *Constraint Satisfaction Problems*. The actual GLS version, with its accompanying FLS, has been first showed by Voudouris [7] and Voudouris and Tsang [8], and finally applied to the 3D-BPP by Faroe et al. [3].

Basically, GLS augments the objective function of a problem to include a set of penalty terms and considers this function, instead of the original one, for minimization by the local search procedure. Local search is confined by the penalty terms and focuses attention on promising regions of the search space

(Voudouris and Tsang, [8]). Iterative calls are made to a local search procedure, denoted as *LocalOpt(X)*. Each time *LocalOpt(X)* gets caught in a local minimum, the penalties are modified and local search is called again to minimize the modified objective function. In a certain measure, the heuristic may be classified as a Tabu Search heuristic; it uses memory to control the search in a manner similar to Tabu Search.

GLS is based on the concept of *features*, a set of attributes that characterizes a solution to the problem in a natural way. In our adaptation of the model, features are the overlaps between the blocks, and we denote by $I_{ij}(X) = \{0,1\}$ the indicator whether blocks i and j overlap or not. In a particular solution, a feature with a high overlap is not attractive and may be penalized. As a result, the value of $overlap_{ij}(X)$ can measure the impact of a feature on a solution X (referred as cost function in Faroe et al. [3]).

The number of times a feature has been penalized is denoted by p_{ij}, which is initially zero. Loosely speaking, we want to penalize the features with the maximum overlap that have not been penalized too often in the past. The source of information that determines which features will be penalized should thus be the overlap and the amount of previous penalties assigned to the features. For this purpose, we define a utility function $u(X) = overlap_{ij}(X)/(1+p_{ij})$. After each *LocalOpt(X)* iteration, the procedure adds one to the penalty of the pairs with maximum utility.

After incrementing the penalties of the selected features, they are incorporated in the search with an augmented objective function

$$h(X) = f(X) + \lambda \sum_{i,j} p_{ij} I_{ij}(X) = \sum_{i<j} overlap_{ij}(X) + \lambda \sum_{i,j} p_{ij} I_{ij}(X)$$

where λ is a parameter – the only one in this method – that has to be chosen experimentally. Thus, when local search has found a solution $X^*= LocalOpt(X)$, overlaps with maximum utility are penalized and become undesirable. In a sense, the search procedure is commanded to set a priority on these features and, for this reason, it jumps out of the local minimum.

3.3 Fast local search

Let us now describe the so-called *fast local search* procedure (see Voudouris and Tsang, [8] and Faroe et al. [3]). FLS is used to transform a current solution X_{cur} into a local minimum $X^* = LocalOpt(X_{cur})$. It will help us to reduce the size of the neighborhood with a selection of the moves that are likely to reduce the maximum utility overlaps.

We define the sets $\upsilon_m(X)$ as subsets of the neighbourhood $\upsilon(X)$ where all solutions in $\upsilon_m(X)$ only differ from X by the value of the variable m ($m = \{x_j, y_j, t_j, a_j, o_j\}$ with j in $\{1,...,n\}$) (in the case of $m = \{o_1,..., o_n\}$, υ_m also includes the particular change in x_j and in y_j that considers a rotation around the four corners of a block. To simplify the explanation, this technical issue is not detailed). The neighborhood $\upsilon(X)$ is thus divided into a number of smaller sub-neighborhoods that can be either *active* or *inactive*. Initially, only some sub-neighborhoods are active (we show at the end of this section how the selection is made to focus on the maximum utility overlaps). FLS now continuously visits the active sub-neighborhoods in a random order. If there exists a solution X_m within the sub-neighborhood $\upsilon_m(X_{cur})$ such that $f(X_m) < f(X_{cur})$, then X_{cur} becomes X_m; otherwise we suppose that the selected sub-neighborhood will provide no more significant improvements at this step, and thus it becomes inactive. When there is no active sub-neighborhoods left, the FLS procedure is stopped and X_{cur}, the best solution found, is returned to GLS. From a less formal point of view, FLS selects at random a variable m within a list of active variables, as long as this list is not empty. Then, it searches within the domain of m any improvement of the objective function. If it does not exist, the variable m becomes inactive and is removed from the list. By doing so, we focus specially on variables open for improvement.

The size of the sub-neighborhoods related to the a_j and the o_j variables is relatively small (A in the first case, 5 in the second (initial + four corners)), therefore FLS is set to test all the neighbors of these sets. But, on the other hand, using an enumerative method for the translations along the x, y and t axis would become very expensive in terms of computing time, if areas and/or time windows are large. We may, however, show that only certain coordinates of such neighborhoods need to be investigated. If m represents x_j, changes in the overlap function only depend on x_j ($h(X) = h(x_j)$). Most of the terms of this function are constant, thus, since we want to compare values, only the few terms dependent on x_j should be computed (furthermore, an overlap is the product of four partial overlaps (three for the overlaps on each of the x, y and z axis, and the fourth equals one if a_i=a_j; zero otherwise). Since we know that only the partial overlaps for the x axis depend on x_j, computing efforts can be reduced to their smallest size). Also, it is obvious that $overlap_{ij}(X)$=$overlap_{ji}(X)$, so that the computing time of one solution is linear (n) instead of quadratic (n^2). Additionally, all functions $overlap_{ij}(x_j)$ are piecewise linear functions, and therefore the functions will attain their minimum in one of their breakpoints (or at the limits of their domains). As a result, FLS only needs to compute the values of $f(x_j)$ with x_j at breakpoints or at extreme values. In fact, there are at most four breakpoints for each function, and only the first and the last one are evaluated. Indeed, in regard to the analogy with the 3D-BPP, a good packing intuitively supposes that the boxes touch each other.

We have shown that FLS represents a relatively fast procedure that leads to a local minimum, if the amount of active sub-neighborhoods is relatively small. Let us remember that *LocalOpt(X)* is called iteratively by GLS, and that penalties are changed with an objective of escaping local minima. Activation of sub-neighborhoods should therefore allow moves on penalized features. The following reactivation scheme is used (Faroe et al. [3]): first, moves on the two blocks i and j, corresponding to the penalized features, are reactivated. Secondly, we reactivate the moves on all blocks that overlap with blocks i and j. The latter reactivation is added to allow FLS to pay attention not only to the two overlapping blocks but also to the whole area around the penalized feature.

4 Selecting the blocks

In the previous chapter, we described a method that minimizes the collective constraints under restriction of the individual constraints and we supposed that there exists at least one feasible solution for the set of blocks initially given. Let us denote this procedure by *GlobalOpt(X)*. If *GlobalOpt(X)* is efficient, it should find a solution with an objective function of zero after a certain time and this solution would be one of the feasible solutions. However, in the initial formulation of the problem, we do not know whether a set of blocks is feasible or not. The combination of GLS and FLS can be used anyway if we rely on the following heuristic assumption: there exists no feasible solution if none is found within a certain amount of computing time T. Consequently, the search heuristic *GlobalOpt(X,T)* is utilized as a test of feasibility and gives the correspondent schedules if a feasible solution is identified within T.

Several methods have been tested using this concept. The objective was to remain as close as possible to the working methods and habits used in the factory under study. From this point of view, an efficient approach for the industrial application is to start GLS with a randomly generated solution X_0 that includes the entire set of blocks ($p_j = 1$ for all $j = \{1,..., n\}$). After a search of T seconds, the algorithm is stopped and returns $X_1 = GlobalOpt(X_0, T)$, the best solution found (in terms of overlap). One of the blocks with the highest overlap is removed from the set ($X_1 \rightarrow X'_1$) and the heuristic $GlobalOpt(X'_1, T)$ is restarted. The entire procedure ends if a solution X_n with zero overlap is found.

A variant procedure is to start with an empty set X_0 ($p_j = 0$ for all $j = \{1,..., n\}$). At each iteration, if the solution $X_{n+1} = GlobalOpt(X_n, T)$ is feasible, then an additional block is inserted in the set; otherwise an overlapping block is removed. This procedure is stopped after a certain amount of computing time, or by any more sophisticated stopping criterion, and returns the solution with the largest collection of blocks. Figure 1 shows the iterative processes of this procedure.

Both approaches suffer from one major default: they are likely to have aversion for the largest blocks. Indeed, we do not have an appropriate weighting scheme to evaluate the preferences between blocks, and, since small blocks generally provide smaller overlaps, they are preferred to larger ones. In the real-life situation, when the entire set of block cannot be produced, the person in charge of scheduling can either subcontract specific blocks in other factories, or change some temporal parameters (e.g. intensify the workforce to reduce processing times or postpone due dates). No formal information can describe all the aspects of these choices. For this reason, the operator should be able to change manually the collection of blocks to be produced. Starting from our "fairly good" feasible solution X_n, iterative $X_{n+1} = GlobalOpt(X'_n)$ calls are ordered manually after deliberate changes ($X_n \rightarrow X'_n$) in the assignment. In addition, a last procedure provides a list with each block that is not assigned even though a feasible solution that includes the block can be found.

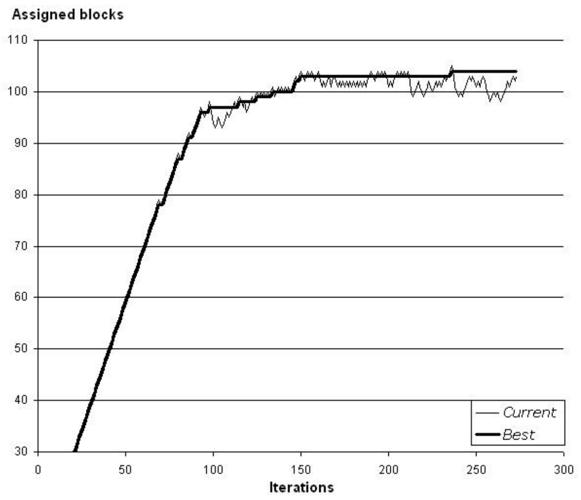

Figure 1: GlobalOpt test instance (T = 1s, TimeLimit = 600s).

By not regenerating solutions on a random basis, some of the information from previous solutions is preserved. Of course a drawback to this approach is that the structure of a previous solution can confine GLS to an area of the solution space that can be difficult to escape (Faroe et al. [3]) suggest a similar problem in their approach for 3D-BPP). We may therefore not reach the very best solution. However, the modus operandi described in this section is developed for a daily industrial use. In that setting, the above drawback may actually be viewed as an advantage. Indeed, it may be very costly for the company to mix up the schedules over and over again. Traditionally, methods for problems of similar classes utilize a construction algorithm during the search and a slight improvement may disturb the whole solution; with GLS, non-problematic regions are not perturbed.

5 Additional real-life issues

Additional constraints may occur in any firm-specific situation. The tool proposed in this paper is easily customizable to most of them. For example, we may need to restrict or force the position of a block (e.g. a tool is only available in one area or the block is already in process). Those constraints were called individual in section 1 and integrating them is trivial: restricted positions are not generated and unfeasible neighbors simply don't exist. As a result, the end-user may fix the value of any variable (including p_j) or reduce its domain.

Specific collective constraints may also appear in the wording of a problem. In our case, the areas of the factory have one single door, and the crane bridge can only carry blocks up to a certain height. As a result, a large block may, for example, obstruct a door, and some blocks might not be deliverable in time because there is no route to transport them out. We dealt with this issue in the same way as for overlaps. For each generated solution *X*, we add to the objective function a new term *h(X)* accounting for exit difficulties.

We believe that any other “collective” constraints may be included in the algorithm using this approach.

6 Conclusion

We presented in this paper a sophisticated local-search heuristic based on the GLS method. Faroe et al [3] described some computational experiments on standard instances. They showed that their algorithm outperforms other approaches for the 3D-BPP. Solutions for our industrial problem are indeed found within a few seconds.

A more important achievement is that the algorithm offers much flexibility for handling the constraints so that it might be adapted to many real-life cases.

Additionally, it focuses the search on promising parts of the solution space and previous schedules are not fully perturbed at each iteration.

Acknowledgements

We wish to thank the “Chantiers de l’Atlantique” (ALSTOM, Saint-Nazaire, France) for their support in this work and the problem definition. In particular, we thank Frédéric Chevalier for his openness and his numerous worthwhile comments. We also we want to show gratitude to ANAST for detailed and constructive comments related to shipbuilding processes.

References

[1] Aarts E. and Korst J. (1989), Simulated Annealing and Boltzmann Machines – a stochastic approach to combinatorial optimisation and neural computing, Wiley.

[2] Coffman E.G., Garey M.R. and Johnson D.S. (1997) Approximation algorithms for bin packing: A survey, D.S. Hochbaum editor, Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, Boston.

[3] Faroe O., Pisinger D. and Zachariasen M. (2003) Guided Local Search for the Three-Dimensional Bin-Packing Problem, INFORMS Journal on Computing, Vol.15, No. 3, pp. 267-283.

[4] Glover F. (1990) Tabu Search: A Tutorial, Interfaces, Vol.20, No 4, pp.74-94.

[5] Martello S., Pisinger D. and Vigo D. (2000) The three dimensional bin packing problem, INFORMS Operations Research, Vol.48, No. 2, pp. 256-267.

[6] Scholl A., Klein R. and Jürgens C. (1997) BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem, Computers & Operations Research 24, pp. 627-645.

[7] Voudouris C. (1997) Guided local search for combinatorial optimization problems, Ph.D. Thesis, Dept. of Computer Science, University of Essex, Colchester, UK

[8] Voudouris C. and Tsang E. (1999) Guided local search and its application to the traveling salesman problem, European Journal of Operational Research, No. 113, pp.469-499.

[9] Wang C.J. and Tsang E. (1991) Solving constraint satisfaction problems using neural-networks, Proceedings of IEE Second International Conference on Artificial Neural Network, pp.295-299