J Mar Sci Technol (2013) 18:404–417
DOI 10.1007/s00773-013-0217-2

ORIGINAL ARTICLE

Optimization of shipyard space allocation and scheduling using a heuristic algorithm

J.-D. Caprace · C. Petcu · M. G. Velarde · P. Rigo

Received: 29 February 2012 / Accepted: 6 March 2013 / Published online: 18 May 2013
© JASNAOE 2013

Abstract In this article we describe the development of a tool that allows planners to efficiently and effectively plan space within valuable areas of a shipyard. Traditionally, space is considered as resource; however, it is difficult to accurately account for and plan its consumption with the currently available planning software's. The spatial scheduling tool described in this article can be used by planners to manually or automatically reserve space within the shipyard for construction of large blocks over the entire erection period of the ship. The software is coupled with a heuristic optimization solver inspired by an algorithm used for "3D bin-packing problems." The result is the ability to efficiently generate and compare multiple space allocation alternatives in a reduced time with the ultimate goal of maintaining the critical ship erection schedule. A better solution than manual or semi-automatic allocation of blocks can be obtained through the optimization module.

Keywords Space allocation · Optimization · Decision making · Scheduling · Planning · Shipbuilding · 3D bin-packing

J.-D. Caprace (✉) · C. Petcu · P. Rigo
ANAST-University of Liège, 1 Chemin des chevreuils, 4000 Liège, Belgium
e-mail: jd.caprace@ulg.ac.be; jcaprace@espol.edu.ec

C. Petcu
e-mail: cristian.petcu@ulg.ac.be

P. Rigo
e-mail: ph.rigo@ulg.ac.be

J.-D. Caprace · M. G. Velarde
FIMCBOR-Escuela Superior Politecnica del Litoral (ESPOL), Guayaquil, Ecuador

M. G. Velarde
e-mail: mvelarde@espol.edu.ec

1 Introduction

1.1 Why space allocation is an issue for shipyards

The high complexity of ship production, due to the interaction of many different disciplines (hull construction, electricity, fluids, interior fitting, propulsion, etc.) requires an intensive design and a detailed production planning where most of the tasks are carried out in parallel. Hugues [1] highlighted that it is necessary to increase the number of simultaneous tasks in order to obtain the best quality, the lowest price, and the shortest manufacturing lead time during the ship production process.

Today, shipyards change their design method in order to increase the number of simultaneous tasks with the use of more structural blocks (modular construction strategy). Traditionally, the majority of the design decisions were taken based on the experience and opinion of the designers. These decisions have a strong influence on production costs, but subsequently on the ship's performance during its life.

One of the most significant observations in the last decades concerning shipbuilding is the increase in size of the ships, as shown for passenger ships on Fig. 1. In addition, making weld in a workshop is much cheaper than creating the same weld in the dry dock (worse access conditions, welding overhead, slower welding process, etc.). The consequence is an increase of the block size and/or the number of blocks while the working surface is almost equal. Moreover, most of the time, it is not possible to enlarge these working surfaces. It follows that a space allocation problem develops.

The assembly of big elements requires a necessary available area within the fabrication workshop to perform the production. As the blocks become larger and heavier,

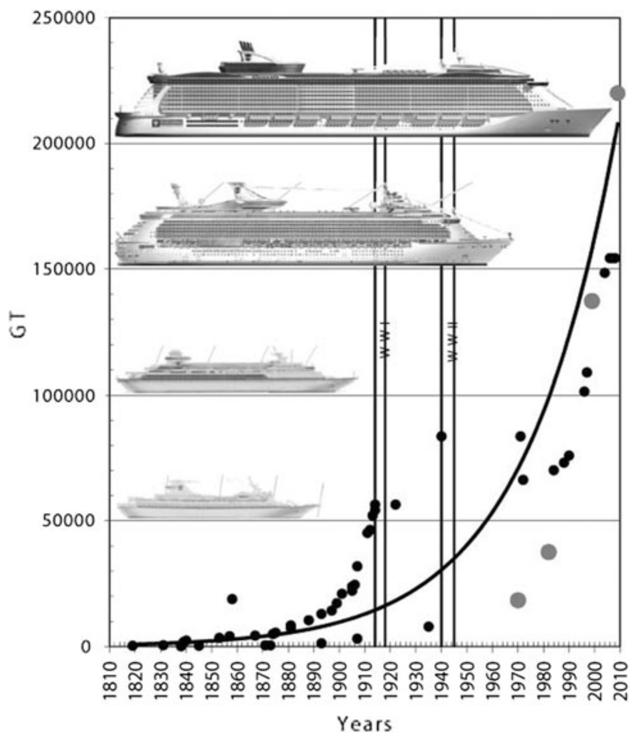


Fig. 1 Largest passenger ship size (GT) throughout the ages [2]

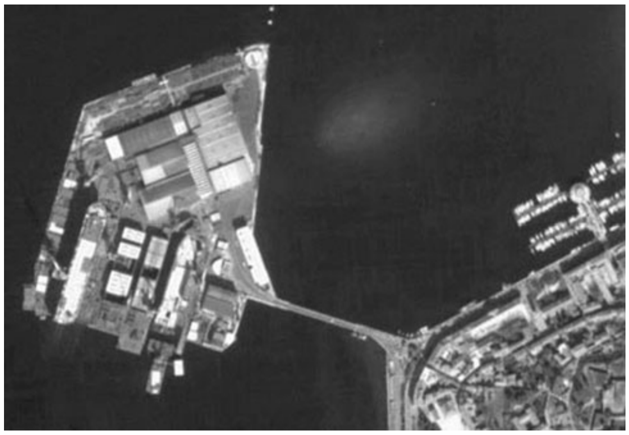


Fig. 2 Limited working space in Uljanik shipyard island (Pula, Croatia)

production space in the shipyard becomes a constraint. There are only limited spaces where the largest blocks can be produced because of the lifting and handling requirements. For this reason, it is important to plan the space accurately in these areas to ensure that blocks are moved only when and where necessary to use the available space efficiently. Unnecessary moves result in non-value-added costs for the block. However, due to production constraints and aggressive construction schedules, maximizing the number of blocks in an area may result in unnecessary moves, while minimizing unnecessary moves results in less efficient use of the space. The limited space available in shipyards, as we can see in Fig. 2 for the Uljanik shipyard in Pula (Croatia), and the growth in the size of blocks and sections force the planners to optimize the use of the available surface within the workshops and storage areas.

1.2 Current practice

Spatial scheduling is still currently being done by small groups of experienced people using tools such as Computer-Aided Design (CAD), PowerPoint, or Excel and schedule information from their planning systems. Although these ad-hoc tools are relatively effective, they are cumbersome and require a significant amount of time to update even for minor schedule changes. In addition, scheduling practices and lessons learned over time are only retained by the experts themselves. Therefore, this knowledge is lost and must be reacquired by yet another generation of new employees. Providing some innovative solution to capture this knowledge and automate the process with a "smarter tool" would provide more efficient allocation of the valuable production space in each of the construction areas facilities.

Research related to optimal block allocation scheduling in shipbuilding is not prevalent, even though it is possible to increase the productivity of shipyards and to decrease the building cost of a ship through efficient use of space resources. However, some recent research shows a growing interest in improving shipyard space utilization inside workshops.

In Korea, simulation-based production scheduling is growing, which can contribute to improving production scheduling, planning work and evaluating various production scenarios [3]. To make the most use of the simulation, coupling optimization with simulation is expected to be far more effective to improve the planning quality as well as to reduce the effort involved in production planning and control [4, 5].

To solve the specific problem, various heuristic-based algorithms have been developed to optimize the block assignment and space allocation. Park et al. [6] presented a scheduling algorithm using partial enumeration and decomposition to generate a spatial allocation plan. Okumoto and Iseki [7] proposed an optimization of block allocation in the assembly area using a simulated annealing method [8, 9], block allocation optimization in the assembly area based on the constraint satisfaction technique (CST) and optimized block division planning using a genetic algorithm and product model [10]. Similarly, Finke et al. [11, 12] and Cho et al. [13] proposed a semi-automated scheduler to increase the utilization of work area space. Utilizing the similarity of the two-dimensional packing problem, Shin et al. [14] recently presented a bottom-left-fill heuristic method for spatial planning of block assemblies.

As presented before, a large spectrum of research has been conducted to investigate various algorithms for optimal configuration and develop decision support systems for spatial scheduling of dynamic block assembly. However, all the studies deal with a limited number of production constraints, which hardly reflect the realistic production situation. For instance, it can be desirable to keep some blocks together during the assembly stage or to place some blocks only in one type of assembly shop, at the exit gate or near the ship that is currently being erected, etc.

The article is organized as follows. After a literature review on relevant research work, a systematic framework for a look-ahead scheduling mechanism is presented in which a heuristic-based algorithm for optimizing the spatial layout of block assemblies is developed. A case study with a computational experiment is then presented to demonstrate the proposed approaches.

2 Space allocation issue

2.1 Similarities with other theories

The dynamic allocation of space in the shipyard is an immensely difficult and time-consuming effort. The difficulty in scheduling floor space, or spatial scheduling, arises because the space allocation for one block significantly affects the availability of floor space for every other block. Scheduling production space to satisfy an erection schedule becomes even more complex when unexpected changes in the schedule occur (e.g., upstream process delays, weather-related delays or subcontractor timeliness) [11, 12]. This illustrates the need for a tool that can assist planners in not only generating efficient spatial layouts, but also modifying these plans accordingly with minimal additional effort. Not only is the practice of scheduling space a difficult problem, but also the automatic or semi-automatic scheduling of the space is even more difficult.

The space allocation issue looks like a cutting stock problem. The cutting stock problem is a well understood problem in the shipbuilding industry. Steel processing facilities in almost every shipyard use nesting software to determine the best allocation of steel plate area for cutting out profiles. Having this technology, the allocation of steel plate space is much more efficient and results in reduced steel waste. Solution procedures to the two-dimensional cutting stock problem have to and continue to be developed to improve the efficiency and computation time of the plate layout.

This issue can also be considered with the conventional three-dimensional bin-packing problem (3D-BPP) where cubes or solid boxes are "packed" into a larger empty container in an effort to maximize the number of boxes in the container; see Fig. 3 and in [15], for example. In the shipbuilding context, the working area, platen, or shop floor length, width and height are considered the X, Y and Z dimensions of the container, and the t dimension is the time schedule horizon. The problem is thus more complex than a simple 3D bin-packing problem: there are three geometric dimensions and in addition the time dimension. In order to simplify the problem, only two geometrical dimensions (floor length and width) are generally considered in addition to the time dimensions. The objective, as defined by the shipyard managers, is thus to maximize the number of building blocks produced in a given surface over a certain time horizon. In the sequel, we refer to this problem as the space and time allocation (STA) problem.

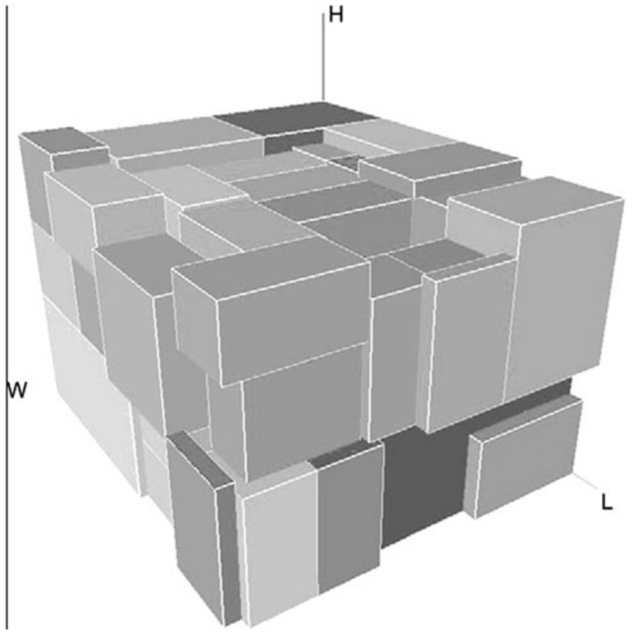


Fig. 3 Three-dimensional bin-packing problem in a container [16]

Only a few solution procedures have been developed for these types of problems [8, 17–19], and optimal solutions procedures have proved to be NP-complete because of the exponential explosion of the solution space [20]. In other words, an "optimal solution" for a large application cannot be found within reasonable computing times. Therefore, the user should accept obtaining a "nearly optimum" solution. An efficient tool should make use of modern heuristics to find such results within short computing times.

There is one key difference between shipyard spatial scheduling and the conventional bin-packing problem. In the bin-packing problem, it is generally assumed that the blocks to be packed are all available at time 0. In the shipbuilding industry, the blocks become available for placement at different times. Also, the general case of the academic problem is not relevant in the practical sense due to the fact that the real-world system has significantly more complex constraints than those in the general case. Some of these constraints include preferred locations, spacing between the units, schedule requirements and so forth.

The studies of Martello et al. [15], Brunetta and Gregoire [21], Foroe et al. [22] and Martello et al. [23] are recent contributions that provide brief surveys of the literature on 3D-BPP. Since the problem is difficult, most efficient approaches rely on local search metaheuristics for the solution of large-scale situations. In particular, Faroe et al. [22] have proposed a guided local search (GLS) heuristic for 3D-BPP. In their computational experiments, this approach appears to outperform the best available heuristics for 3D-BPP. It also offers a high degree of flexibility in its implementation, so that it can be easily adapted to variants of the problem involving different objective functions and/or additional constraints. Therefore, the algorithm that we have developed for STA explicitly builds with the aid of their work.

2.2 Challenges of space allocation issue

The dynamic allocation of blocks in shipyards is a huge, difficult and time-consuming effort. The difficulty in space allocation arises because:

- The allocation of space to one block significantly affects the availability of floor space for the other blocks [11]. Scheduling production space to satisfy an erection schedule becomes even more complex when unexpected changes in the schedule occur (e.g., upstream process delays, weather-related delays or subcontractor timeliness).
- The allocation of space in an industrial environment is an issue with different complex production constraints:
  - Block height might be important because sometimes blocks have to be moved by a crane bridge above others blocks.
  - Spacing between blocks might be required for safety and accessibility reasons.
  - Spacing below blocks might be required for transportation with skid platforms.
  - Space above blocks might be required for the movement of other blocks.
  - The preferred location for some blocks might require placing blocks close to specific tools or equipment, etc.

This illustrates the need for a flexible tool that can assist planners in not only generating optimal spatial layouts, but also modifying these plans day after day according to the variation of the initial schedule (delays, unplanned maintenance, etc.). The next section describes an approach that has been developed to help in the allocation and planning of floor space within the shipyard.

3 Approach

The objective of the tool described in this article is to increase the utilization of the working area, while maintaining production schedules. An innovative approach has been developed in order to include the following features:

- The automatic allocation of activities (blocks, sections, panels, etc.) in the workshops;
- The minimization of wasted surfaces;
- Long-term and day-to-day simulations in order to determine how a delay impacts the global planning;
- The post-processing of the result in order to allow fast decision-making (floor plan printing, display of working load and working force charts, display of surface utilization charts, etc.).

This tool should thus provide planning proposals, i.e., a location and a starting day for each block. Unfortunately, it may be that the available surface in the assembly hall is not sufficient to produce the entire set of blocks. The tool should then try to help the user to make the most efficient decision.

For simplification reasons, no details will be taken into account regarding the production processes. It is also assumed that blocks have their final shape during the assembling process. We do not take the successive assembly stages into account. In addition, a block is considered to have a parallelepiped shape. Many blocks are indeed almost parallelepipeds, and other shapes can be considered using the same optimization technique.

Dealing with simple data is more convenient, and we know that a decision tool is only efficient if it keeps things easy to use, even if complex methods are used to solve the problem. Indeed, the software would lose part of its power and efficiency if the time needed to prepare the data becomes excessive. In addition, the ability to make changes quickly and to view the impact of those changes in real time provides a tool that will significantly reduce the cost of planning and replanning.

The first phase of the tool's development is creating a graphical user interface (GUI) to assist the planner in his tasks. The second phase of the tool development focuses on capturing the planner's knowledge and using it to implement an automated optimization module. The following sections provide further details on these two phases of tool development: the GUI and the automated optimization scheduling procedure. Finally, we present an industrial case study and a set of conclusions.

4 Required data for optimization

Both the data related to the shipyard' facilities and to production activities (ship blocks, sections, etc.) are required in order to define the problem.

4.1 Shipyard facilities

The assembly surfaces in a shipyard often comprise more than one working area A_k, for $k = 1, 2, \ldots, m$ of rectangular shape, such as section assembly halls, block assembly areas, painting halls, outfitting areas, etc. Different activities on the block are processed in these areas. Each working area could contain different preferential zones in order to perform the activity in a specific place in the workshop rather than another. Subsequently, three different levels of information should be considered: the workshop, the working area and the preferential zone.

The main information required about the shipyard facilities are:

- The available space in the working areas (length L, width W, height H) of the workshop; see Fig. 4;
- The crane capacities (maximum load, height under the hook C);
- The definition of preferential zones q inside the workshop (length, breadth, height, type of work, etc.);
- The position of the gates;
- The industrial calendar (working days for each ship);
- The personnel availability over time.

It is imperative to know the location of the gates in the assembly hall and the crane bridge height. Indeed, it may happen that a particular block cannot be taken out because other high blocks are in its way to the gate, and the height of the crane bridge may not be sufficient to pass over them (crane hook constraint). If blocks are too heavy for the crane bridge, they need to be driven out on a skid platform. In this case, no block at all should remain in the way, and supports for blocks have to be elevated in order to allow the skid platform to get under the block.

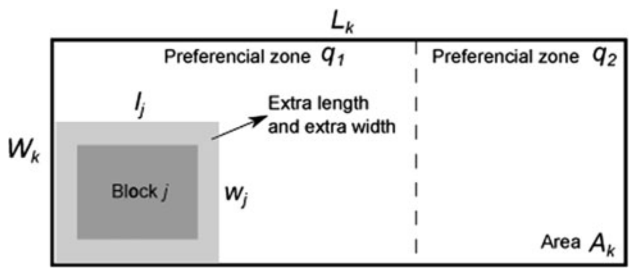


Fig. 4 Working areas and blocks

4.2 Production activities

Basically, the input data of the software can be summarized as a list of n "activities." Each activity represents a certain work to be done on a particular block $j = 1, 2, \ldots, n$. Hence, the following information that can be provided by the enterprise resource planning (ERP) system of the shipyard is required:

- Description of the block: block identification, ship identification, comments, etc.;
- Prismatic dimensions of each block: length l_j, width w_j, height h_j, for $j = 1, 2, \ldots, n$ and related spaces allocated to movements around the blocks. Blocks are considered to be parallelepipeds. The major reason for this assumption is that these data are very easily available; it is easier to deal with basic shapes, and their representations on a surface are more easily interpretable. This concept does not drastically affect the results since most blocks indeed have an (almost) parallelepiped shape. For accessibility and security reasons, a certain distance may be required between nearby blocks in the assembly hall. Therefore, an extra length, an extra width and an extra height can be considered;
- Position of the block x_j and y_j: these parameters are coordinates representing the position of the upper left corner of block j in the selected area a_j;
- Processing time t_j: processing time interacts with two aspects: the total amount of workforce needed for each block and the duration of work. At this stage of the planning, a precise processing time cannot be assessed; therefore, the processing time has to be estimated. An estimation of the total amount of man-time needed is available; thus, the processing time is computed by dividing this man-time by the available number of workers. The workload assessments become more precise over time. In addition to the processing time of an activity, some time may be required to prepare the appropriate surface and build up supports for blocks or to dismantle them. This work has no effect on the start

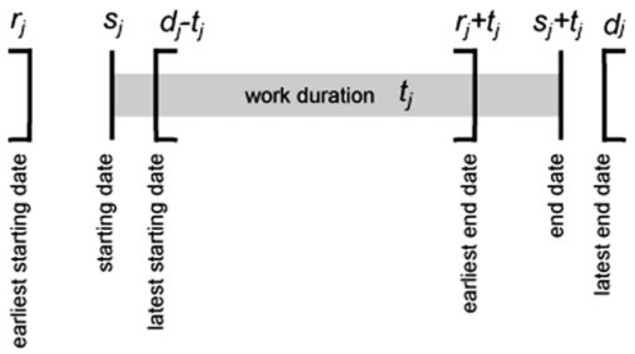


Fig. 5 Date and duration of an activity

and the end date. Therefore, it has to be taken into account separately;

- Date of production of each block: in this case, the earliest starting date, also called the release date, r_j, is used (earliest date at which production can start because the required parts are available for assembly), and the latest end date, also called the due date, d_j, is the date at which the activity of the block has to be delivered. See Fig. 5;
- Starting date $s_j \in \{r_j, \ldots, d_j - t_j\}$: this parameter indicates the starting date of the assembly of block j;
- Area $a_j \in \{1, 2, \ldots, m\}$: this parameter indicates the working area where the block j will be produced. In some cases, the values may be restricted to a subset of the sections, depending on block characteristics.
- Orientation $o_j \in \{0, 1, 2, 3\}$: this parameter indicates the orientation of block j in selected area a. Blocks can have four orientations, turning by 90°.
- Subcontractor possibility $b_j \in \{0, 1\}$: this parameter indicates whether activity j will be produced inside the shipyard ($b_j = 0$) or whether it will be subcontracted ($b_j = 1$). During optimization, these blocks will preferentially be selected to be produced in other workshops if the assembly area is overloaded;

The following optional additional information can be defined by the user to improve the quality of the scheduling solutions:

- Fictitious block $c_j \in \{0, 1\}$: this Boolean parameter indicates that a block is a dummy. This option allows the possibility to introduce zones temporarily reserved for activities different from block-mounting operations (e.g., storing the ship engines in the assembly shop, temporary space required for cranes, etc.). The fictitious blocks are only used to reduce the available space during a definite time window.
- Target date f_j: this option allows the user to give a preferential start date for the optimization module. If this date cannot be reached by the optimizer, the trend will be to approach it as best as possible. On one hand, if we put the target date on the early start date, we can perform the space allocation with the "as soon as possible" rule, and on the other hand, if we put the target date on the latest start date, we perform the space allocation with the "as late as possible" rule.
- Group of blocks g_j: several blocks can be grouped so that the optimizer will find a position for them as if they are a single unit. The advantage is that we can simulate the impact of the production of blocks nearby similar ones. Thus, the optimization module takes into account a group of blocks as a huge block. A snap tool was implemented to link several blocks together.
- Preferential zone q_j: this field indicates the preferred zone to produce the blocks;
- Ship zone p_j: this field indicates the zone of the ship to which the blocks belong. During the optimization, we are trying to group the block from the same ship zone together to decrease the movements of the gantry crane;

5 The graphical user interface

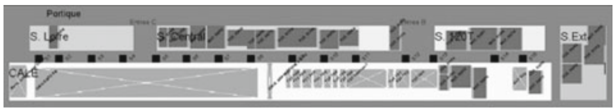
The first part of the tool is an interface for the user (usually a planner or construction manager) to interact with the block attributes, schedule information and actual placement of the units within a production working area. A color code is used to show the different statuses of the blocks.

The main frame of the GUI is divided into two windows. One is the spatial view of the workshop (top view of the workshop on a given date); the other is the timeline view (top view of the workshop with a dimension in space and a dimension in time). These two frames interact in order to display the situation of the workshop at different dates by the dragging of the daily line in the timeline view.

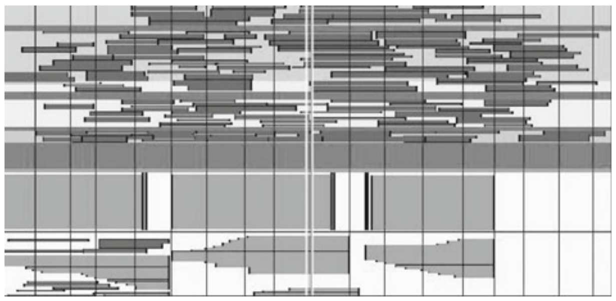
5.1 Spatial view of the workshop

This frame (see Fig. 6a) simply shows a top overview of the workshop at a selected date. It is possible that certain blocks will appear or disappear depending on when those blocks were placed and when they are scheduled to be complete.

The user can move blocks (drag and drop) inside space (*X* and *Y*) for the day selected. The main block attributes, such as length, width, height, weight and schedule information, such as scheduled start date, planned duration,



(a) Spatial view at a given date



(b) Time-line view – where the vertical axis represents spatial X dimension and the horizontal axis the time-line

Fig. 6 Main frame of the space allocation optimization tool

earliest starting date, latest ending date and actual start, can be edited in a properties windows.

5.2 Timeline view of the workshop

The timeline frame (see Fig. 6b) shows an overview of each working area with an axis for the time (horizontal axis) and another one for a dimension (X or Y-vertical axis). The user can move blocks along the temporal and spatial (X or Y) dimension by a simple drag and drop. However, the displacement of the blocks is limited between the earliest start date and the latest end date; see Fig. 5. The vertical line can be placed on a precise day of the timeline and shows the state of all areas at this date.

5.3 Detection of overlaps

The user is also notified of any collisions between overlapping blocks. The tool detects all the collisions and overlaps between the blocks not only occurring at the present time, but also for the entire planning period.

5.4 Towards the automated planning

While the spatial scheduling tool (GUI) provides a planner with several features to generate efficient spatial plans more rapidly, the actual method of allocating space is not much different than current shipyard practices, where block placement decisions are based on expert-user knowledge. The following section describes a method to automatically allocate and optimize space according to a heuristic algorithm. While it is nearly impossible to capture the entire set of rules, constraints and preferences used to generate a near-optimal spatial layout, the automated scheduler can be used to generate a valid baseline layout, and the end-user can make modifications to this layout using the spatial scheduling tool.

6 Optimization of space allocation

6.1 Optimization variable and objective function

The STA problem consists of orthogonally ordering the n blocks into the m rectangular areas, without overlapping, so as to respect the time constraints, with the objective to produce the largest possible number of building blocks.

To achieve this, we defined the following decision variables for each block $j = 1, 2, \ldots, n$:

- Position of the block x_j and y_j;
- Starting date of the activity $s_j \in \{r_j, \ldots, d_j - t_j\}$;
- The block orientation $o_j \in \{0, 1, 2, 3\}$;
- The working area $a_j \in \{1, 2, \ldots, m\}$;
- Subcontractor possibility $b_j \in \{0, 1\}$.

In addition, each variable can be fixed so that the optimization algorithm does not have the opportunity to modify the value. For example, this feature is used to define the daily production situation of the workshop (real block position inside the workshop).

A solution, that is to say, an assignment of values to the above variables, is feasible if the individual and the collective constraints are met. We call individual constraints those that bear on one block only, regardless of the other blocks. The individual constraints can be modeled as follows:

1. each block must fit within the width of an area a_j: $x_j \geq 0$ and $x_j + [o_j w_j + (1 - o_j) l_j] \leq W$;
2. each block must fit within the length of an area a_j: $y_j \geq 0$ and $y_j + [o_j l_j + (1 - o_j) w_j] \leq L$;
3. each block must fit in its time window: $s_j \geq r_j$ and $s_j + t_j \leq d_j$

On the other hand, collective constraints deal with the interaction between the positions of different blocks. Unless we say otherwise, the only collective constraint is that the blocks may not overlap.

6.2 Algorithm

As previously mentioned, our developments are based on the solutions presented by Forae et al. [22] for the 3D-BPP problem.

6.2.1 General approach

Let X be any solution of the STA problem, that is, any assignment of values to the variables a_j, x_j, y_j, o_j and s_j for $j = 1, 2, \ldots, n$. We implicitly assume that $b_j = 1$ for all j. While trying to find a feasible schedule, our local heuristic search strictly enforces the individual block constraints, meaning that X always satisfies the constraints (1)–(3). On the other hand, we do not enforce the collective constraints, but we measure the extent of their violation, and these measures are summed in an auxiliary objective function to be minimized. Without additional real-life collective constraints, the extent of the violations can be measured by the total "volume" $[m^2 \times days]$ of pairwise overlaps between the n blocks. Thus, if we denote by $overlaps_{ij}(X)$ the volume of the overlap between blocks i and j, then the auxiliary objective function can be formulated as shown in Eq. 1.

$$f(X) = \sum_{1 \leq i < j \leq n} overlap_{ij}(X) \tag{1}$$

Starting from an arbitrary infeasible solution where blocks can overlap, searching for a feasible solution can be

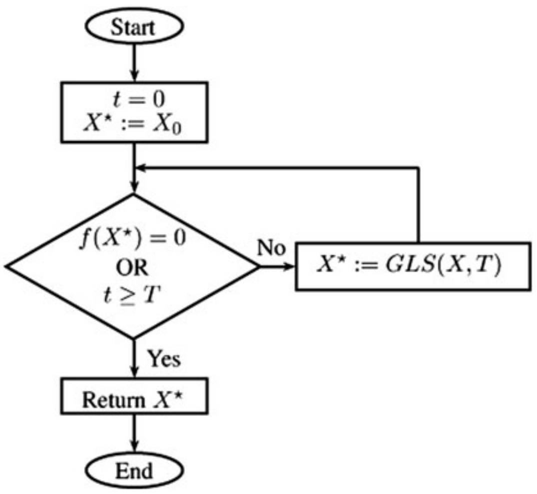


Fig. 7 Optimization flow of the STA problem —where $X^\star$ is the best available solution, X_0 is the initial solution, $f(X^\star) = 0$ represents a solution without overlaps, t measures the runtime, T is the runtime limit, and GLS is the Guided Local Search

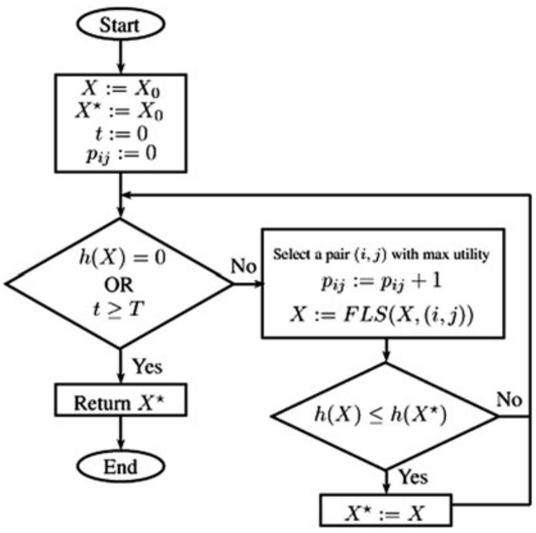


Fig. 8 Optimization flow of the *GLS* (X, T)—where X is the current solution, $X^\star$ is the best available solution, X_0 is the initial solution, t measures the runtime, T is the runtime limit, FLS is the fast local search, p_{ij} is the penalty for all pairs of blocks, (i, j) is a pair of blocks, and $h(X)$ is defined in Eq. 2

achieved by minimizing the function f, since an objective value of zero indicates that all the collective constraints are satisfied; see Fig. 7.

A typical local search procedure starts by moving from the current solution X to another X' in a neighborhood $v(X)$ whenever this move improves the value of the objective function. Slightly adapting the framework of Faroe et al. [22], who do not allow rotating the boxes, we define the neighborhood $v(X)$ as the set of all solutions that can be obtained by translating any single block along the coordinate axes or along the timeline, or by a move of the same relative position in another area of the working surface, or by a $\pm 90°$ rotation of a block around one of its four corners. A neighbor of X is therefore constructed by assigning a new value to exactly one of the variables x_j, y_j, s_j, a_j or o_j. It is clear that this definition allows to move from any solution to any other solution through a sequence of neighbors.

It is well known that local search procedures may easily get stuck in a local minimum of poor quality. Another difficulty with local search procedures is that the neighborhood of any given solution may be quite large, and therefore, exploring the neighborhood to find an improving move can be very costly in computation time. To deal with the above issues, we rely on the GLS heuristic and its accompanying neighborhood reduction scheme called fast local search (FLS).

6.2.2 Guided local search

Generally speaking, GLS augments the objective function f of a problem to include a set of penalty terms associated with "undesirable features" of a solution, and it considers the new function h, instead of the original one, for minimization by a local search procedure. This procedure is confined by the penalty terms and focuses attention on promising regions of the search space. Each time the local search procedure gets caught in a local minimum, penalties are modified and the local search procedure is called again to minimize the modified objective function. This general scheme has been adapted to 3D-BPP by Faroe et al. [22]. In their precedure, the features of a solution X are the Boolean variables $I_{ij}(X) \in 0, 1$, which indicate whether blocks i and j overlap $I_{ij}(X) = 1$ or not $I_{ij}(X) = 0$. The value of the $overlap_{ij}$ (X) measures the impact of the corresponding feature on the solution X. The number of times an "active" feature has been penalized is denoted by p_{ij}, which is initially zero. Thus, the augmented objective function takes the form shown in Eq. 2, where λ is a parameter—the only one in this method—that has to be chosen experimentally.

$$\begin{aligned} h(X) &= f(X) + \lambda \sum_{1 \leq i < j \leq n} p_{ij} I_{ij}(X) \\ &= \sum_{1 \leq i < j \leq n} overlap_{ij}(X) + \lambda \sum_{1 \leq i < j \leq n} p_{ij} I_{ij}(X) \end{aligned} \tag{2}$$

Intuitively speaking, GLS attempts to penalize the features associated with a large overlap, but that have not been penalized very often in the past. More formally, we define a utility function $\mu_{ij}(X) = overlap_{ij}(X) \,/\, (1 + p_{ij})$ for each pair of blocks (i, j). At each iteration, the procedure adds one unit to the penalty p_{ij}, corresponding to the pair of blocks with maximum utility, then calls the local search procedure; see Fig. 8. In a sense, the search procedure forced setting a higher priority on these features. Since features with maximum utility keep changing all the time, this guiding principle prevents GLS from getting stuck in local minima.

The adaptation of the algorithm has been numerically validated and tested for several standard test cases and published in Langer et al. [24] and Bay et al. [25].

6.2.3 Fast local search

The main objective of FLS is to reduce the size of the neighborhoods explored in the local search phase by an appropriate selection of moves that are likely to reduce the overlaps with maximum utility.

To describe the FLS, consider any solution X and any variable m among the variables x_j, y_j, s_j, a_j, o_j with $j \in [1, \ldots, n]$. Informally, FLS selects at random a variable m within a list of activate variables; as long as this list is not empty, active variables are those that are most likely to lead to an improvement of the current solution. The FLS searches within the domain of m for an improvement of the objective function. If no improvement is found, then the variable m becomes inactive and is removed from the list until the end of the current call of FLS.

More formally, we define $v_m(X)$ as the set of all solutions that differ from X only by the value of variable m. The neighborhood $v(X)$ is thus divided into a number of smaller sub-neighborhoods, as shown in Eq. 3.

$$v(X) = \bigcup_m v_m(X) \tag{3}$$

Each of the sub-neighborhoods $v_m(X)$ can be either active or inactive. Initially, only some sub-neighborhoods are active. FLS now continuously visits the active sub-neighborhoods in random order; see Fig. 9. If a solution X_m exists within the sub-neighborhood $v_m(X)$ such that $h(X_m) < h(X)$, then X 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 no active sub-neighborhoods are left, the FLS procedure is terminated, and the best solution found is returned to GLS.

The size of the sub-neighborhoods related to the a_j and the o_j variables are relatively small; therefore, FLS is set to test all the neighbors of these sets. On the other hand, using an enumerative method for testing the translations along the x, y and s axes would be very time consuming, especially when areas and/or time windows are large. We may observe, however, that only certain coordinates of such neighborhoods need to be investigated. Indeed, as pointed out by Faroe et al. [22], all $overlap_{ij}(x)$ functions (respectively $overlap_{ij}(y)$, $overlap_{ij}(s)$ are piecewise linear functions and will for that reason always reach their minimum in one of their breakpoints or at the limits of their domains.

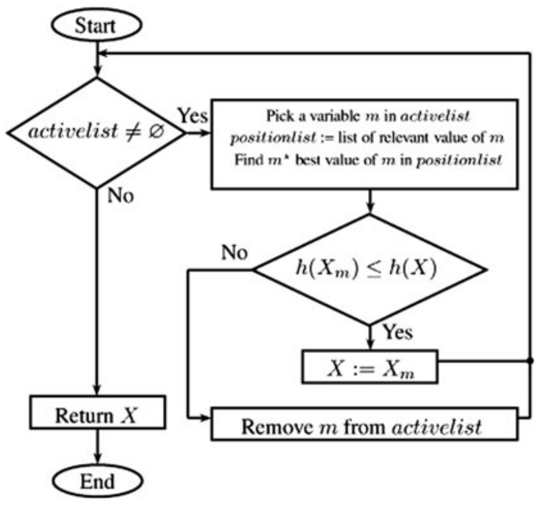

Start
$activelist \neq \varnothing$
Yes
No
Pick a variable m in $activelist$
$positionlist$:= list of relevant value of m
Find m^* best value of m in $positionlist$
$h(X_m) \leq h(X)$
No
Yes
$X := X_m$
Return X
Remove m from $activelist$
End

Fig. 9 Optimization flow of the FLS (X, (i, j))—where X is the current solution, (i, j) is a pair of blocks, *activelist* is the list of the variables associated with the moves applicable to blocks i and j, and to the blocks overlapping either i or j, $m^\star$ is the best value of m, i.e., the best move reducing the objective function more, X_m is the solution obtained by setting $m := m^\star$ in X, and $h(X)$ is defined in Eq. 2

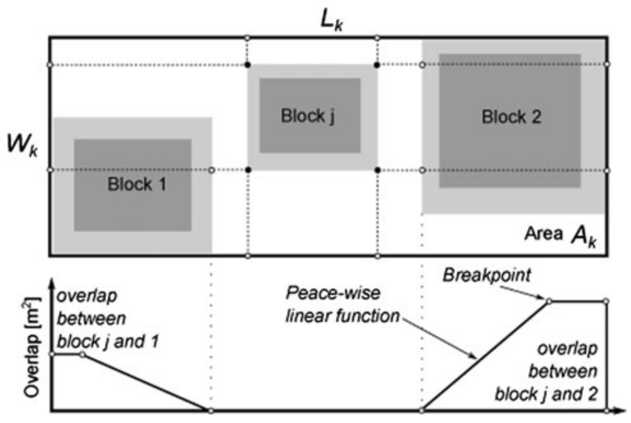

L_k
W_k
Block j
Block 2
Block 1
Area A_k
Breakpoint
Overlap [m²]
overlap between block j and 1
Peace-wise linear function
overlap between block j and 2

Fig. 10 Illustration of FLS neighborhood size reduction

Thinking of the geometry of the 3D-BPP, we can easily understand that a best packing arises either when the boxes touch each other along their faces or when they touch the sides of the bins; see Fig. 10. As results, FLS only needs to compute the values of $f(x)$ (respectively, $f(y)$, $f(s)$) for x (respectively, y, s) at breakpoints or extreme values. In fact, there are at most four breakpoints for each overlap function, and only the first and the last are evaluated.

Once all active moves have been deactivated by the FLS, two possibilities remain. If the total objective is null, the process is finished. In the other case all the moves will become inactive, because no more improving moves exist. A local minimum is reached, and the FLS iteration is finished. The process starts again at the first step, and as those overlaps cannot be solved, they will sometimes be selected as the pair having the maximum utility. When this occurs, moves of these blocks are reactivated, and their penalties are increased. So, if we add the penalties to the objective function, the objective function value of the actual solution will enlarge, and moves improving the objective function will appear, even if the result is worse in terms of overlap. But now we have left the local optimum, and a better solution can be found after several iterations.

6.2.4 Selecting the blocks

In the previous sections, we described a GLS heuristic to find a feasible solution to the STA problem. If GLS works as expected, then it should return a space and time allocation with zero overlap, i.e., a feasible solution, when there is one. In general, however, no such feasible solution may exist for the set of blocks initially included in the instance, and we face the problem of selecting a maximum subset of blocks to be scheduled for assembly. In order to solve this problem, we rely on the following assumption, describing that if GLS cannot find a feasible solution of STA within a predetermined amount of computation time T, then the heuristic assumption is that the instance is probably infeasible.

A procedure has been developed allowing to add and remove blocks from the current set; see Fig. 11. Thus, assume that, at any iteration of the procedure, X is a solution (feasible or not) involving some subset of blocks. If the solution GLS(X, T) returned by GLS is feasible, then this solution is a candidate to be the final optimal solution. So, we record it if it is better than the best incumbent solution X^*, and we try to include an additional block in the set. On the other hand, if GLS(X, T) is not feasible, then a fast post-processing step is performed to produce a feasible solution X': this is achieved by simply removing blocks in a greedy fashion until all overlaps are cancelled. The solution X' is recorded if it is better than the incumbent X^*; then, we remove an overlapping block from GLS(X, T), and the process is repeated. The procedure is stopped after a predetermined amount of computation time, or by any more sophisticated stopping criterion, and returns the feasible solution X^* involving the largest collection of blocks.

6.2.5 Additional constraints

The general case of the academic problem is not relevant in the practical sense due to the fact that the shipbuilding industry has significantly more complex constraints than those of the general case. Various side constraints have to be considered in order to increase the practical relevance of the STA model. Fortunately, the GLS framework proved

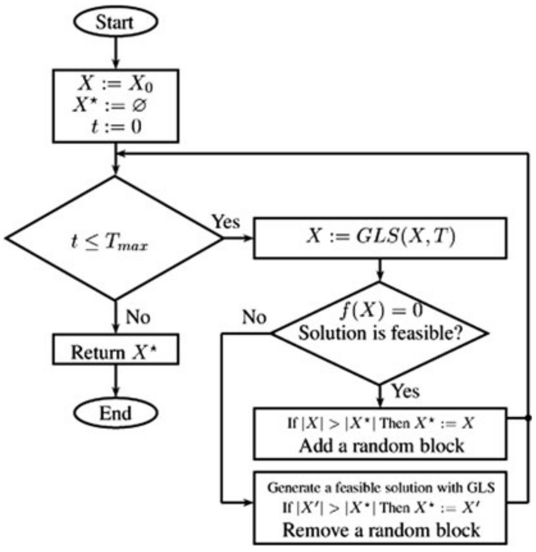


Fig. 11 Optimization flow of the block selection—where X is the current solution, $X^\star$ is the best available solution, X' is a feasible solution generated by GLS, X_0 is the initial solution, t measures the runtime, T is the runtime limit for GLS, and $T_{\max}$ is the global runtime limit of the optimization

flexible enough to incorporate most of these constraints without too much additional effort.

For example, in practice, it may be necessary to restrict or to impose the position of certain blocks, e.g., because these blocks are already in process when the planning process is launched, or because some required handling or production equipment is only available in a particular area, etc. Such individual constraints on blocks are easily handled by the GLS algorithm: forbidden positions and infeasible neighbors are simply not generated during the search. Thus, in practice, the end-user may fix the value or reduce the domain of any variable when using the software. For instance, he may prohibit the rotation of some blocks and/or their translations and/or the working area.

Another industrial constraint that can occur is to allocate the preferential zones q_j defined in Sect. 4.2 for some specific blocks. In other terms, the GLS will allocate the block taking into account a certain priority instead of using a random selection of the blocks. Other priorities can easily be defined taking into account other parameters such as the block weight, block size or block complexity.

More complex collective constraints also appeared in the real-life situation. In particular, for the assembly halls, each working area has a single door, and the crane bridge can only carry the blocks up to a certain height C; see Fig. 12. As a result, it may happen that a tall block obstructs the door or stands otherwise in the way, and some blocks may not be delivered in time because there is no feasible passageway to carry them out of the hall. Here again, the GLS approach proved “generic” enough to deal

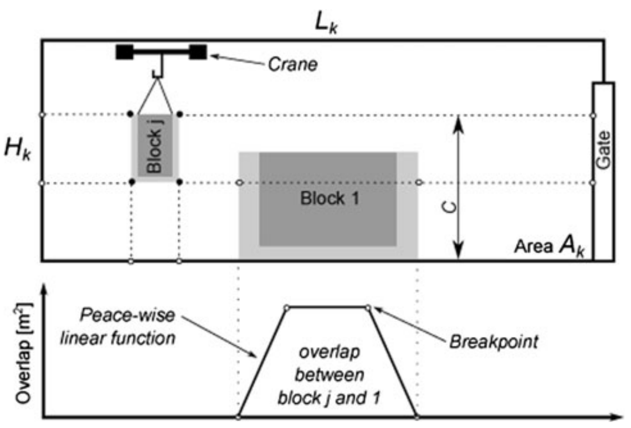


Fig. 12 Illustration of overlaps due to crane movements

with this issue. For each generated solution X, we added to the objective function $h(X)$ a new penalty term that accounts for exit difficulties as shown in Eq. 4, where $exit_{ij}(X)$ measures the overlap between block i and the "exit path" for block j.

$$\begin{aligned} g(X) &= h(X) + e(X) \\ &= \sum_{i<j} overlap_{ij}(X) + \lambda \sum_{i<j} p_{ij} I_{ij}(X) + \sum_{i<j} exit_{ij}(X) \end{aligned} \tag{4}$$

The exit path for j is restricted by security constraints, which impose the use of a straight path, and thus it is determined by:

- the longitudinal interval $[x_j, x_j + o_j w_j + (1 - o_j) l_j]$;
- the transversal interval $[0, y_j + (1 - o_j) w_j + o_j l_j]$, as the doors are at position $y = 0$;
- the vertical interval $[C - h_j, C]$, since each block can be carried up to the height of the crane;
- the completion date $s_j + t_j$ of block j;
- the area a_j where block j is produced.

Note that the value of the exit terms could somehow be scaled in relation to the $h(X)$ values, but this did not appear to be useful in our procedure, as the new penalty terms proved sufficient to drive the objective function to zero.

Additional collective constraints arise when a family of related blocks have to be produced for a ship. For example, all the blocks that include emergency boats require similar production equipment, and it is convenient to allocate them to the same zone of the working area. In a similar way, two blocks that are adjacent in the ship structure may need to be produced next to each other in the assembly area so as to allow fine positioning of connecting elements such as structural members or piping tracks. An easy way to cope with the latter requirement is to define a super block that includes the two (or more) adjacent blocks and to replace the individual blocks by this super block in the data of the problem. With the variable g_j described in Sect. 4.2, several blocks can be grouped so that the GLS will find a position for them as if they are a single unit. This parameter does not directly affect the GLS algorithm, but it increases the end-user flexibility considerably. For the first situation, however, this method is too restrictive, and we preferred instead to define a "distance" constraint. With the variable p_j described in Sect. 4.2, GLS will minimize the relative distance between two blocks in order to reduce the undesired movement of the gantry crane.

Other collective constraints could certainly be included in the model by taking full advantage of the flexibility of the GLS framework.

6.2.6 *Robustness*

The GLS procedure always starts from an initial solution X_0. A drawback of this approach is that the structure of X_0 can confine the GLS to an area of the solution space that can be difficult to escape (especially for small values of λ); therefore, the search process may not reach the very best solution.

However, in a dynamic industrial setting, this apparent drawback turns out to be an advantage. Indeed, it may be very costly or practically impossible for the company to readjust the schedules and the allocation of blocks to the working areas frequently. By generating new solutions from previous ones, the GLS procedure actually ensures that the structure of previous solutions can be preserved when the production plans are updated. This is to be contrasted with various methods proposed for rectangle packing problems, which typically rely on construction strategies and for which a slight modification of the data may lead to major perturbations of the solution. As a consequence, it may prove rewarding to run GLS with a relatively small value of λ in the industrial context.

7 Case study

7.1 Presentation

This case study focuses on the assembly shop of a European shipyard where relatively small sections (60–120 tons) are joined together to form huge blocks (550–750 tons). Typically, the ship is then erected in the dry dock block-by-block until the ship is finished using a gantry crane.

The modeling of this workshop contains four working areas (see Fig. 13) as well as the dry dock. The "bin" is an additional area where blocks are temporary placed if they are not allocated.

A data set of 268 blocks has been considered where 61 blocks are fictitious and mainly represent part of the ships

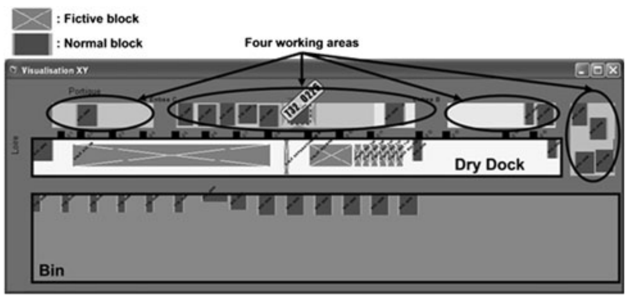


Fig. 13 Layout of the assembly shop considered for the case study

in construction in the dry dock, 51 blocks are fixed and represent the actual situation of the working areas, and finally 156 blocks should be allocated to solve the STA problem. The surface of the blocks to be allocated varyies between 200 m^2 and 1,800 m^2, with an average of 750 m^2. The dimensions of the blocks to be allocated (width and length) vary between 8.5 and 50 m with an average of 28 m. The time period of the data set is about 1.5 year. The working duration of the blocks varies between 15 and 68 days with an average of 40 days. This data set corresponded to a highly constrained shipyard STA problem where the space available inside the workshop is quite similar or superior to the sum of the block surface.

7.2 Manual and automatic scheduling

7.2.1 Manual allocation of activities

Manually, the user can drag and drop a non-allocated block from the "bin" area into an empty working area. Even if an area seems to be void, another block may have been allocated to this place some days later. The software automatically detects such conflicts: critical blocks will directly change color. A double click on any block shows all block attributes directly. Multi-selection is also available to change the value of any parameters for several blocks.

7.2.2 Allocation of activities with the optimization module

In practice, allocating blocks with the optimization tool ensures that there is no collision. The optimizer tool always gives a feasible solution.

While it is nearly impossible to capture the entire set of rules, constraints, and preferences used by the planner, the optimization module can be used to generate a valid baseline layout, and the end-user can make modifications to this layout using the GUI. The tool of overlapping detection is very powerful in this case of manual adjustment of the final optimized solution.

If no feasible solution is found by the optimization algorithm, the user can directly see which blocks lead to problems and then choose an adapted solution. The user can identify the production surface utilization problems that may happen for the actual data (basically the fact that not all blocks can be produced in time). He may for example raise the workforce availability or subcontract some blocks.

7.2.3 Day-to-day optimization

The difficulty in space allocation, or spatial scheduling, arises because the allocation of space to one block significantly affects the availability of floor space to the other blocks. Scheduling production space to satisfy an erection schedule becomes even more complex when unexpected changes to the schedule occur (modification of block duration, production delays, etc.).

In order to take this constraint into account, we implemented the possibility to fix the attribute of some blocks so that they cannot be moved by the optimizer. This functionality is useful to define the starting state of the workshop: blocks already in the workshop cannot be moved during the optimization!

7.2.4 Data connection

A link between the current enterprise resource planning (ERP) system of the shipyard and the software was also implemented to update all attributes of blocks before an optimization. When a modification occurs inside the planning (activities duration, block dimensions, production delays, etc.), the latest information is always available for the optimization.

7.3 Results and achievements

A comparison between the manual allocation of activities and the optimization algorithm was done by a planner in the shipyard. The planner used a simplistic allocation rule such as the allocation of the first block in one corner and processing by rows or by columns. Nevertheless, we considered some additional constraints such as the safe distance between the blocks or the possibility to rotate the blocks in order to improve the solution.

The main problem during the allocation of the blocks is that the planner does not foresee the scheduling of the blocks along time. For an optimal allocation of the activities, the optimization algorithm is really helpful.

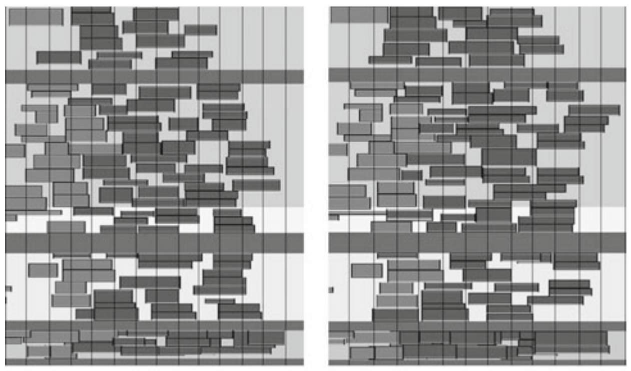
Table 1 as well as Fig. 14 show the relative gains between the manual and the automatic allocation of the blocks (optimization algorithm).

We observe in Table 1 that the working surfaces are better used (gain of 12.7 %) because more blocks have been allocated during the same considered period (137

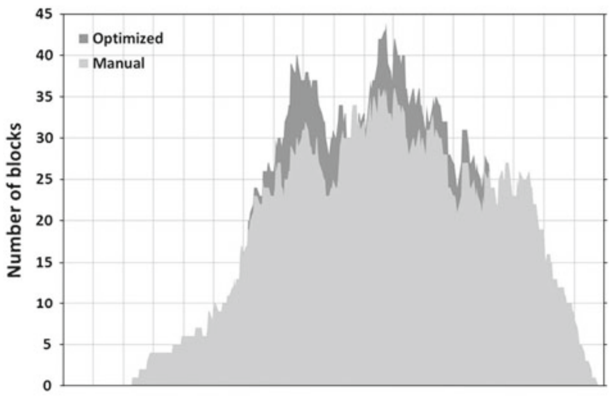
Table 1 Gain between the manual and the automatic allocation

Description	Unit	Manual	Optimized	Gain (%)
Surface used	$m^2 \times$ days	3593324	4115958	12.7
Block placed	#	118	137	13.8
Block not placed	#	38	19	50

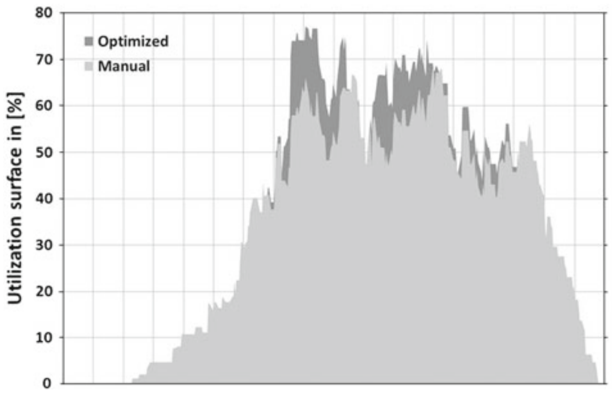
# Represents the cardinality of a set



(a) Time-line – Manual **(b)** Time-line – Optimized



(c) Number of blocks vs. time



(d) Surface utilization in % vs. time

Fig. 14 Comparison of the timeline results between the manual and automatic allocation, where **a** and **b** present the timeline view where the vertical axis is the spatial *X* dimension and the horizontal axis the timeline, and **c** and **d** present respectively the number of blocks and the surface utilization ratio in the function of time

blocks instead of 118 blocks). In the case of the optimized schedule, only 19 blocks have not been allocated, while for the manual solution, 38 blocks have not been placed. However, the planner should find a solution to allocate the remaining 19 blocks that cannot be produced in time. He may for example raise the workforce availability to reduce the working duration or subcontract some blocks. Furthermore, the average utilization of the working areas has been improved, as shown in Fig. 14d.

Both of the solutions, i.e., the manual scheduling and the automatic scheduling, were considered feasible by the shipyard planner. Nevertheless, he corrected 3 % of the position of the blocks for the optimized solution taking into account additional constraints.

In addition, the scheduling time is drastically reduced (hours instead of days). This is probably the most interesting advantage of the developed tool. The planner considered that the tool can contribute to improving production scheduling works and appreciate particularly that he is now able to evaluate various production scenarios in a short time.

For small problems, when the space available inside the workshop is largely superior to the sum of the block surfaces, computation times are between 30 and 150 s for ~250 blocks. For more constrained problems, when the space available inside the workshop is quite similar to the sum of the block surfaces, the time required to find an optimized solution increases from 5 min to 1 h.

Without the tool, such planning could take several days. Consequently, another advantage of the tool is that different schedules can be tested. For instance, if any production parameter is changed, such as the block splitting or the number of ships to produce simultaneously, the impact on the total production time can easily be studied. These kinds of studies were not possible manually.

8 Conclusions

In this article, we have presented a space and time allocation problem arising in large shipyards, and we have modeled it as a three-dimensional bin-packing problem.

We have demonstrated the main advantages of the GLS to solve the STA problem:

- Good correspondence between results obtained and industrial constraints;
- Low computation time (some minutes);
- Four-dimensional problems are solved (three spatial dimensions and one temporal dimension);
- The solution obtained is always feasible;
- If a schedule modification arises and causes an overlap between blocks, the algorithm will only be able to solve the issue locally without modifying the global solution.

The proposed innovative approach allows more efficient scheduling for the shipyard. The planner can test more alternatives and rapidly modify the scheduling to find the best one. But the preparation and verification of data for the simulation remain a major stage to ensure the reliability of the results.

Gains obtained for the shipyard are substantial. By using the new concept, the workshop productivity is increased as less time is needed for scheduling and better space utilization is achieved.

This generic approach allows incorporating various real-life constraints and leads to the successful implementation of a flexible and robust application for the shipbuilding industry, but potentially also for other industries.

9 Future work

The work outlined in this article presents a new promising method for shipyard spatial scheduling. Nevertheless, several improvements or integrations of new constraints could be performed:

- An extension of the rectangular shapes for workshops and/or blocks to any shapes. However, this would require a complete overhaul of the software and its optimization method.
- An implementation of a tool to fit and smooth the workload of the workshop to the workforce available. It could be done during the optimization phase as a multi-objective optimization.
- Development of a tool to consider predecessors and successors for the different activities to be allocated.

References

1. Hugues O (1994) Applied computer aided design. Technical report, International Ship and Offshore Structure Congress—ISSC, inst. marine dynamics, Newfoundland
2. List of the world's largest cruise ships. http://en.wikipedia.org/wiki/
3. Kim Y, Lee D (2007) A study on the construction of detail integrated scheduling system of shipbuilding process. J Soc Naval Archit Korea 44:48–54
4. Bair F, Langer Y, Caprace J, Rigo P (2005) Modelling, simulation and optimization of a shipbuilding workshop. COMPIT'05—the 4th international conference on computer applications and information technology in the maritime industries, May, vol 1, Hamburg, pp 283–293
5. Nedeb C, Friedewald A, Wagner L, Hubler M (2006) Simulation of material flow processes in the planning of production spaces in shipbuilding. COMPIT'06—the 5th international conference on computer applications and information technology in the maritime industries, May, vol 1, Leiden, pp 186–198
6. Park K, Lee K, Park S, Kim S (1996) Modeling and solving the spatial block scheduling problem in a shipbuilding company. Comput Ind Eng 30:357–364
7. Okumoto Y, Iseki R (2005) Optimization of block allocation in assembly area using simulated annealing method. J Jpn Soc Naval Archit Ocean Eng 1:71–76 (in Japanese)
8. Lee K, Lee J, Choi S (1996) A spatial scheduling system and its application to shipbuilding: DAS-CURVE. Exp Syst Appl 10:311–324
9. Lee D, Kang Y, Kim H (2005) Block assignment planning for shipbuilding considering preference shop and load balancing. Proc 12th ICCAS 2005 1:55–68
10. Wibisono M, Hamada K, Kitamura M, Kesavadey V (2007) Optimization system of block division using genetic algorithm and product model. J Jpn Soc Naval Archit Ocean Eng 6:109–118
11. Finke DA, Ligetti CB, Traband MT, Roy A (2007) Shipyard space allocation and scheduling. J Ship Prod 23:197–201
12. Finke DA, Ligetti CB, Traband MT, Roy A (2008) Activity-based spatial scheduling. J Ship Prod 24:12–16
13. Cho K, Chung K, Park C, Park J, Kim H (2001) A spatial scheduling system for block painting process in shipbuilding. CIRP Ann Manuf Technol 50:339–342
14. Shin JG, Kwon OH, Ryu C (2008) Heuristic and metaheuristic spatial planning of assembly blocks with process schedules in an assembly shop using differential evolution. Prod Plan Control 19:605–615
15. Martello S, Pisinger D, Vigo D (2000) The three dimensional bin packing problem. Oper Res 48:256–267
16. Wu Y, Li W, Goh M, de Souza R (2010) Three-dimensional bin packing problem with variable bin height. Eur J Oper Res 202:347–355
17. Lee JK, Lee KJ, Park HK, Hong JS, Lee JS (1997) Developing scheduling systems for Daewoo Shipbuilding: DAS project. Eur J Oper Res 97(2):380–395
18. Lim A, Rodrigues B, Yang Y (2005) 3-D container packing heuristics. Appl Intell 22:125–134
19. Lodi A, Martello S, Vigo D (2002) Heuristic algorithms for the three-dimensional bin packing problem. Eur J Oper Res 141:410–420
20. Garey MR, Johnson DS (1990) Computers and intractability; a guide to the theory of NP-completeness. W. H. Freeman & Co, LinkedIn
21. Brunetta L, Gregoire P (2005) A general purpose algorithm for three-dimensional packing. INFORMS J Comput 17:328–338
22. Faroe O, Pisinger D, Zachariasen M (2003) Guided local search for the three-dimensional bin-packing problem. INFORMS J Comput 15:267–283
23. Martello S, Pisinger D, Vigo D, Boef ED, Korst J (2007) Algorithm 864: general and robot package variants of the three-dimensional bin packing problem. ACM Trans Math Softw 33:1–12
24. Langer Y, Bay M, Crama Y, Bair F, Caprace J, Rigo P (2005) Optimization of surface utilization using heuristic approaches. COMPIT'05—the 4th international conference on computer applications and information technology in the maritime industries, May, vol 1, hamburg, pp 419–425.
25. Bay M, Crama Y, Langer Y, Rigo P (2010) Space and time allocation in a shipyard assembly hall. Ann Oper Res 179:57–76