# A look-ahead strategy-based method for scheduling multiprocessor tasks on two dedicated processors

Meziane Aider, Fatma Zohra Baatout, Mhand Hifi

Version of Record: https://www.sciencedirect.com/science/article/pii/S0360835221002928
Manuscript_6143859bbd7ab839c51397509c22f140

# A Look-Ahead Strategy-Based Method for Scheduling Multiprocessor Tasks on Two Dedicated Processors

Méziane Aider[†], Fatma Zohra Baatout[†] and Mhand Hifi[‡,⋆].

**All authors are listed in alphabetical order.**

[†]**Méziane Aider**

LaROMaD, USTHB BP 32 El Alia 16111 Bab Ezzouar, Algiers, Algeria

E-mail: meziane.aider@gmail.com

[†]**Fatma Zohra Baatout**

LaROMaD, USTHB BP 32 El Alia 16111 Bab Ezzouar, Algiers, Algeria

E-mail: baatout.fz@gmail.com

[‡]**Mhand Hifi**

[⋆]Corresponding author.

EPROAD EA 4669, UPJV, 7 rue du Moulin Neuf, 80000 Amiens, France

E-mail: hifi@u-picardie.fr

## Declarations of interest.

The authors do not have any possible conflicts of interest.

# A Look-Ahead Strategy-Based Method for Scheduling Multiprocessor Tasks on Two Dedicated Processors

**Abstract**

In this paper, we investigate the use of a look-ahead strategy combined with path-relinking for tackling the problem of scheduling multiprocessor tasks on two dedicated processors. An instance of the problem is composed of a set of tasks divided into three subsets and two processors, where some tasks can be executed either on one processor or two processors. The goal of the problem is to schedule all tasks such that the execution of the last assigned task is minimized. First, the proposed method starts with a greedy feasible solution built by applying a knapsack rule related to a predefined sequence. Second, a series of local operators are added in order to drive the search process around a series of neighborhoods. Third, a first diversification strategy based upon drop and re-build operators is employed. The second diversification/intensification strategy is used for highlighting the performance of the method; it incorporates a look-ahead strategy combined with the path-relinking. Finally, the performance of the proposed method is experimentally analyzed on a set of benchmark instances of the literature, where its provided results are compared to those achieved by more recent methods available in the literature. Its behavior is also evaluated by providing a statistical analysis using both the *Sign test* and the *Wilcoxon signed-rank test*.

**Keywords.** Heuristics, Look-Ahead, Optimization, Scheduling.

## 1 Introduction

The problem of Scheduling multiprocessor Tasks on Two dedicated Processors (noted ST2P) is an NP-hard combinatorial optimization problem (Hoogeveen, van de Velde, & Veltman, 1994), where its goal is to affect all available tasks to either a single processor or two different processors. Generally, for scheduling problems, the measures of performance are often categorized as follows (Brucker, 2007): (i) criteria based upon completion times,

(ii) criteria based on due dates and, (iii) criteria based on inventory cost and use.

In this paper, the studied problem can be viewed as a special case belonging to the family of scheduling problems (Drozdowski, 1996; Priya & Sahana, 2019; Bukchin, Raviv, & Zaides, 2020), where the set of tasks is divided into three groups:

1. the first group contains the tasks that need to be performed on the first processor,
2. the second group includes those executed on the second processor and,
3. the third group contains the tasks which must be performed simultaneously on both processors.

One can observe that, on the one hand, several scheduling problems may consider different objective functions: (a) minimizing the makespan that corresponds to minimize the completion time of the last executed task, (b) to minimize the summation of the delays of all tasks, (c) to minimize both delays and makespan, etc. On the other hand, several versions of the scheduling problem can be accessed (a) on the number of available processors, (b) how tasks are assigned on certain processors, etc.

In this paper, ST2P is tackled where its goal is to minimize the completion time of the last executed task (makespan). Such a problem can be encountered in several real-world applications, like production and data transfer (Manna & Chu, 2010). More precisely, an instance of ST2P is defined as follows: let $N$ denote the set containing $n$ tasks to schedule on two dedicated processors (namely $P_1$ and $P_2$) such that a task $j$ is released at time $r_j$ and has to be processed without preemption during its processing time $p_j$ and, $C_j$ is the completion time of task $j$ while $C_{\max}$ denotes the makespan of the schedule to minimize. As described in Graham et al. (1979), such a problem is defined as $P2|fix_j, r_j|C_{\max}$, where:

$P2$**:** represents two processors on which all tasks must be executed.

$fix_j$**:** means that each task $j$ is assigned and its assignment is fixed (either to a single processor or to both processors).

$r_j$**:** denotes the release date of the task $j$.

$p_j$**:** is the processing time of the task $j$ when executed on the processors.

$C_{\max}$**:** denotes the makespan (completion time) of the last assigned/executed task.

The remainder of the paper is organized as follows. Section 2 reviews the relevant literature related to some scheduling problems. Section 3 discusses a formal description of ST2P (Section 3.1) and its tight lower bound (Section 3.2), proposed by Manna and Chu (2010). The aforementioned lower bound is used for analyzing the quality of the results provided by all methods. Section 4 describes the look-ahead strategy-based algorithm for approximately solving ST2P. A starting solution, using a knapsack greedy rule, is presented in Section 4.1. The intensification operators, combined with a tabu list, are described in Section 4.2. The diversification strategy, using the drop and rebuild operator, is discussed in Section 4.3. The look-ahead strategy incorporated into the core of the algorithm is detailed in section 4.4. Section 5 exposes the performance of the proposed method that is evaluated on a set of benchmark instances of the literature. Its achieved results are compared to those provided by the best and more available methods in the literature. Finally, Section 6 concludes the paper.

Because the proposed method can be viewed as an augmented version of the method proposed in Aïder, Baatout, and Hifi, (2020) and, in order to make the paper self-containing, some parts of the aforementioned paper are repeated in what follows.

## 2 Literature review

The scheduling can be viewed as one of the old problems belonging to the combinatorial optimization family that can be encountered in both real-world applications and academic studies. Such a problem can model several real-world situations, where its formalism fits well with the most complex problems while the academic studies considered practical situations as references. However, the problems belonging to the scheduling family (Drozdowski, 1996) are often NP-hard in the strong sense, where searching for optimal solutions (single objective function) or Pareto optimal points (more than one objective function) remains intractable. In the aforementioned paper, the author described the first principal variants related to both scheduling multiprocessor tasks on parallel processors and scheduling multiprocessor tasks on dedicated processors, where their complexity analysis was underlined.

Because of the NP-hardness of the majority of problems belonging to the scheduling family, any exact method may be used for tackling some small-sized instances and so, the availability of effective heuristics and meta-heuristics are of paramount importance. These methods include an evolutionary technique-based approach, as described in Priya

and Sahana (2019), where some effective strategies employed into multi-population meta-heuristics were underlined. In the same paper, an extended version using useful strategies was designed for efficiently tackling job scheduling in a multiprocessor environment. In addition to the previous work, Lei and Cai (2020) provided an extensive review on solving production scheduling problems that are based upon multi-population meta-heuristics while designing a classification for linking several versions. We note also that in Priya and Sahana (2019), the authors presented a systematic literature review for the directed acyclic graph scheduling; that is the scheduling problem modeled through a directed acyclic graph. Based on the survey of more than fifty references, the authors provided an overview of the last three decades of research on the scheduling domain.

Other studies have tackled several versions of the problems belonging to the scheduling family, like Bianco, Blazewicz, Dell'Olmo, and Drozdowski (1997) who studied the problem of scheduling tasks on two dedicated processors with preemptive constraints (noted $P2|fix_j, r_j, pmtn|C_{\max}$), where tasks can be interrupted and finished later. The authors designed an optimal solution procedure that is based on two steps polynomial time complexity.

Blazewicz, Dell'Olmo, and Drozdowski (2002) tackled a special case of scheduling multiprocessor tasks on two identical parallel processors. The authors discussed the complexity analysis for special cases, like considering (i) the scheduling with unit execution time, (ii) the preemptable tasks with ready times and, due-dates and, (iii) precedence constraints. For these cases, the goal of the problem is to minimize the schedule length and the maximum lateness.

Thesen (1998) designed a tabu search-based algorithm for approximately solving multiprocessor scheduling problems. The proposed algorithm combines tabu strategy, local search operator, and how to manage the lists used. Several strategies have been considered, like random blocking related to the size of the tabu list, frequency-based penalties for diversifying the search process, and the hashing operator for stocking high solutions. The experimental part showed that some combinations have better behavior than others; in this case, the achieved results, on benchmark instances, are better than those reached by several algorithms of the literature.

Blazewicz, Dell'Olmo, Drozdowski, and Speranza (1992) tackled the problem of scheduling multiprocessor tasks on three dedicated processors. The authors made a complexity analysis of the problem and studied different cases related to this problem for which they

proposed optimal solutions in polynomial time complexity.

Buffet, Cucu, Idoumghar, and Schott (2010), developed two tabu search-based algorithms for solving the multiprocessor scheduling problem using $m$ processors. The authors followed the standard principle of the tabu search, where a starting solution is built by respecting a legal schedule, the intensification strategy that checks possible permutations between tasks for improving the quality of the solutions, the diversification strategy using a local search for exploring unvisited subspaces. The resulting algorithm was evaluated on thirty randomly generated instances and showed that the method was able to outperform one of the best methods of the literature.

Zhang, Li, and Wang (2016) solved the single batch-processing machine scheduling problem, where different job sizes and arbitrary job arrivals are considered. Their proposed method is a hybrid one, where the local search is combined with path-relinking. In fact, three kinds of local searches were implemented in order to enhance their method for providing efficient diversified solutions. Further, the path-relinking was incorporated to explore solutions with high quality through links between the highest solution at hand and target initial solutions belonging to the elite set. In their experimental part, the authors underlined the high effect of the path-relinking when incorporated into the local search-based method.

Concerning the problem studied in this paper, scheduling tasks on two dedicated processors, Manaa and Chu (2010) proposed an exact algorithm to solve it. The algorithm is based upon the classical branch and bound procedure, where the internal nodes are bounded with special lower and upper bounds. The experimental part showed the performance of such a method, where it was able to solve instances up to thirty tasks within fifty minutes.

Kacem and Dammak (2014) tailored an effective genetic algorithm for approximately solving the scheduling of a set of tasks on two dedicated processors. The principle of the algorithm is based upon the classical genetic principle reinforced with a constructive procedure able to provide feasible solutions for the problem. The resulting algorithm was evaluated on random instances generated following Manaa and Chu (2010)'s generator, where the experimental part showed that the method was able to achieve solution values closest to those provided by the tight lower bound proposed by Manaa and Chu (2010).

Aïder, Baatout, and Hifi (2020) designed a reactive search-based method for solving the scheduling of a set of tasks on two dedicated processors. Their method starts with a greedy solution provided by a knapsack procedure, where the problem is considered as an ordering of a set of items into knapsacks. The method incorporates both intensification

and diversification strategies hoping for the improvement of the final solutions. Indeed, the intensification strategy is based upon the classical 2-opt and 3-opt local searches reinforced with a tabu list for avoiding cycling. The diversification strategy combines the so-called drop and rebuild strategy, for jumping from one search space to another and thus trying to explore more unvisited subspaces. Finally, the computational part was conducted on a set of benchmark instances of the literature, and its provided results were compared to the best bounds achieved by the method available in the literature; the authors underlined the efficiency of their method for all tested instances.

More recently, several case studies have been investigated in the literature, like the problem related to the relay satellite system mission scheduling (Song, Xing, Wang, Yi, Xiang, & Zhang, 2020), a variant of the multiprocessor task scheduling problem, motivated by a real problem arising in the semiconductor industry (Bukchin, Raviv, & Zaides, 2020), the multiprocessor task scheduling problem with dedicated processors such that each task requires a given and fixed set of processors while processors are the vertices of a given initial graph and the required set of processors must induce a connected subgraph of the initial graph (Kononova, Kononovaa, & Gordeev, 2020) and, the special distributed assembly no-idle flow-shop scheduling problem (Zhao, Zhang, Cao, & Tang, 2021) related to the modern supply chains and manufacturing systems.

# 3 Modeling and bounding ST2P

This section is divided into two parts. First, a mathematical model of ST2P is presented in Section 3.1. Second and last, Section 3.2 describes a tight lower bound already proposed by Manna and Chu (2010).

## 3.1 A formal description of the problem

Because ST2P is NP-hard in the strong sense (Hoogeveen, van de Velde, & Veltman, 1994), we first propose a mathematical model to formulate this problem. We recall that an instance of ST2P is characterized by a set of tasks to be processed on two dedicated processors. Each task has its release time and its processing time and, all tasks are categorized according to their processing. Some tasks need to be assigned only to the first processor and some others to the second one while the remaining tasks, however, need both processors simultaneously. The goal of the problem is to find a schedule that assigns, without preemption, each task

according to its category, where the length of the final schedule (makespan) should be minimized.

For the rest of the paper, the following assumptions are considered: (i) all the numerical numbers are integer, (ii) any task cannot be interrupted once a processor starts processing it, (iii) each processor can process only one task at a time, for the tasks being processed by one processor at a time and, (iv) the processors do not breakdown, no maintenance operations are considered between the production operations. Further, according to the previous notations, we use the additional ones given as follows:

- $N$ : set of tasks to be processed.
- $n$ : number of tasks to be processed.
- $P_1$ : the set of tasks requiring the first processor.
- $P_2$ : the set of tasks requiring the second processor.
- $P_{12}$ : the set of tasks requiring both processors simultaneously.
- $p_j$ : processing time of the task $j$, $j \in N$.
- $r_j$ : release date of the task $j$, $j \in N$.
- $M$ : a non-negative constant penalty, greater than any number to which it will be compared.

Finally, the formal description of ST2P can be stated as follows:

$$\text{Minimize } C_{\max} \tag{1}$$

$$C_j \geq C_i + p_j + (x_{ij} - 1).M, \quad \forall\, (i,j) \in (P_1 \cup P_{12})^2 \tag{2}$$

$$C_j \geq C_i + p_j + (x_{ij} - 1).M, \quad \forall\, (i,j) \in (P_2 \cup P_{12})^2 \tag{3}$$

$$x_{ij} + x_{ji} = 1, \quad \forall\, (i,j) \in (P_1 \cup P_{12})^2 \tag{4}$$

$$x_{ij} + x_{ji} = 1, \quad \forall\, (i,j) \in (P_2 \cup P_{12})^2 \tag{5}$$

$$C_{\max} \geq C_i, \quad \forall\, i \in (P_1 \cup P_2 \cup P_{12}) \tag{6}$$

$$C_i \geq r_i + p_i, \quad \forall\, i \in (P_1 \cup P_2 \cup P_{12}) \tag{7}$$

$$x_{ij} \in \{0,1\}, \quad \forall\, (i,j) \in (P_1 \cup P_{12})^2 \cup (P_2 \cup P_{12})^2, \tag{8}$$

where $x_{ij} = 0, (i,j \in N^2)$, if the task $j$ completes before the task $i$, starts, 1 otherwise. The first line (1) of the model refers to the objective function, where the makespan $C_{\max}$ should be minimized. Constraints (2) and (3) express that if the task $j$ is sequenced after the task $i$ is completed, the completion time of the task $j$ is greater than or equal to the sum of

the completion time and the processing time of that task $i$. Constraints (4) and (5) mean that for any pair $\{i, j\}$ of sequenced tasks on the same processor, one has to be completed before the other starts. The constraint (6) ensures that the makespan is greater than or equal to the completion time for each task $j$. The constraints (7) mean that the completion time of task $j$ is greater than or equal to its release date plus its processing time. Finally, constraints (8) are related to the domain of all decision variables.

### 3.2 ST2P's lower bound

Manaa and Chu (2010) proposed a nice lower bound for ST2P that is based on relaxing the original problem into two subproblems to solve. They also proved that the proposed bound provides an optimal solution for the preemptive case of the problem, i.e., $P2|fix_j, r_j, pmtn|C_{\max}$. The calculation of such a bound is explained in what follows.

A task $j \in N$ is called a $P_1 - task$ (resp. $P_2 - task$) if it is allocated to the processor $P_1$ (resp. $P_2$) while it is called $P_{12} - task$ whenever the task $j$ requires simultaneously both processors $P_1$ and $P_2$; that is a biprocessor task. Then, the lower bound can be computed by splitting ST2P into two subproblems, where all bi-processor tasks are divided into two sets of mono-processor tasks each. In this case, the first (resp. second) set, noted $P_{12}^1 - tasks$ (resp. $P_{12}^2 - tasks$) is separately scheduled on each processor. Thus,

- $P_1 - tasks$ and $P_{12}^1 - tasks$ should be scheduled on processor $P_1$.
- $P_2 - tasks$ and $P_{12}^2 - tasks$ should be scheduled on processor $P_2$.

Finally, the optimal solution for each subproblem can be provided by processing tasks in nondecreasing order of their release dates $r_j$ on each processor. Positioning step by step the tasks affected to each processor induces an optimal solution for each subproblem, an optimal solution $C_1^{\text{opt}}$ for the first subproblem with processor $P_1$ and, $C_2^{\text{opt}}$ for the second one with processor $P_2$. Hence, ST2P's lower bound corresponds to

$$\max(C_1^{\text{opt}}, C_2^{\text{opt}}).$$

Note that the solution procedure used for computing the aforementioned bound is a polynomial-time algorithm with an order time complexity of $O(n \log n)$.

# 4 A look-ahead strategy-based method

This section exposes the main principle of the proposed look-ahead strategy-based method for scheduling tasks on two dedicated processors problems. Indeed, the following steps may summarize the key features of the method:

1. Starting the search process with a greedy solution provided by applying the so-called basic knapsack procedure (cf. Section 4.1).

2. Making a series of moves on the current solution for achieving an improved solution (cf. Section 4.2).

3. Perturbing the search process by using both drop and rebuild strategy for reaching a new current solution, according to a new order on tasks (cf. Section 4.3).

4. Using a look-ahead strategy for linking a series of a couple of solutions aiming the enhancement of the solutions at hand (cf. Section 4.4).

5. Steps (1)-(4) are repeated until a satisfactory solution is reached.

## 4.1 A knapsack procedure

One can observe that building any solution for the studied problem is equivalent to provide a sequence of positions related to the tasks on the processors. In this case, an initial greedy solution can be built by applying a standard scheduling's greedy procedure adapted for ST2P. The procedure uses two main steps: (i) reordering the tasks (items) according to a given criterion and, (ii) selecting step by step a non-affected task (item) and assigning it to a processor (knapsack). The second step is iterated until positioning all tasks (items) on their corresponding processor(s) (knapsack).

The following steps describe the main steps of the standard scheduling's greedy procedure. Suppose that $r_j$ denotes the release date of the task $j$, and $p_j$ its processing time. Then,

1. Determine all ratios related to the processing time per release date, i.e., the value $\frac{p_j}{r_j},\ j \in N$.

2. Rank all tasks according to the non-increasing order of ratios; that is $\frac{p_1}{r_1} \geq \ldots \geq \frac{p_j}{r_j} \geq \ldots \geq \frac{p_n}{r_n}$.

By using the above steps to each task, according to the current order, an initial solution may be built for ST2P. This solution represents a sequence of tasks assigned to either the first processor, or the second processor, or both processors.

## 4.2 Intensification strategy

Determining a new sequence (solution) with higher quality is equivalent to force the assignment of some tasks to processors (providing a partial solution) and, to solve the rest of the problem for completing the partial solution. Indeed, introducing some moves between tasks is equivalent to fix some of them and reassigning the rest of the tasks to their corresponding processors(s).

### 4.2.1 A 2-opt Operator

A 2-opt operator is a quick local search procedure that is able to improve solutions even if it is based upon simple local modifications of the current solution. Generally, from a given (current) feasible solution, the 2-opt operator repeatedly makes some swaps/shakes as long as the quality of the induced solution is improved. Whenever the search process stagnates around the same objective value, then the 2-opt operator reaches its limits; that is a situation where the method is trapped into a local optimum. In this paper, we propose a standard 2-opt operator that consists of swapping two randomly chosen positions of the current sequence. The series related to these swaps induced the current neighborhood around the solution at hand.


Figure 1: A swapping operator between positions $\Pi_1$ and $\Pi_2$ related to two assigned tasks

Figure 1 illustrates the swapping operator applied by the intensification search. Of course, making a swap between two tasks (as illustrated in Figure 1) may induce either a feasible solution or an unfeasible one. In the case of an unfeasible solution, the following greedy repairing operator is used for making it feasible. The procedure is composed of two steps as described in what follows.

Let $i$ and $j$, $i \neq j$, denote two assigned tasks (after making a swap), such that $i$ is positioned before $j$. Then the following two-steps procedure is applied to the provided configuration.

***The first-step.*** It can be applied as follows:

1. According to the position of the task $i$, move all tasks from the left to the right till removing all overlapping.
2. According to the position of the task $j$ (with its new position), move all tasks from the left to the right till removing all overlapping.

***The second-step.*** Because the swapping operator produces a new sequence, we then apply the knapsack greedy procedure to that order.

Hence, by applying the above steps to the current solution, one can observe that a series of solutions can be built and so, these solutions form the current 2-opt neighborhood.

### 4.2.2 A 3-opt operator

In this section, we propose a local search based upon the 3-opt operator. As observed above (Section 4.2.1), a current solution may be locally improved by using a simple 2-opt operator that is based on small moves. We then propose to introduce a *neighborhood* operator with higher freedom, which can mix two consecutive neighbors around the current solution. The main idea of such a strategy is to iterate a series of small moves around the current solution. At a certain internal iteration, consider an alternative search operator with higher moves and continue the search by applying small moves. Such a search is iterated until matching predefined stopping criterion.

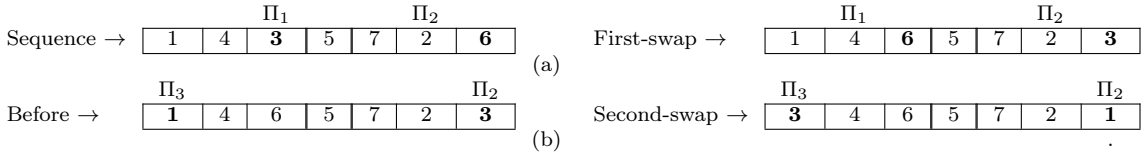


Figure 2: A 3-opt operator related to two couples of swapping: (a) the first couple of positions $(\Pi_1, \Pi_2)$ and, (b) the second couple of positions $(\Pi_2, \Pi_3)$.

Each step of the higher move-based operator can be summarized as follows (in what follows, we consider $\hat{S}$ as the solution at hand):

1. Select two random tasks from the solution $\hat{S}$, permute both tasks for forming a new configuration $\hat{S}'$.

2. Select two new random tasks from $\hat{S}'$ (different from the already swapped tasks), permute these tasks for forming a new configuration $\hat{S}''$.

3. Call the 2-opt operator on $\hat{S}''$ for improving the quality of the solution and, let $S^{\star}$ be the new achieved solution (according to $\hat{S}$).

Figure 2 illustrates the steps used when applying the 3-opt operator that is applied to the current feasible solution.

#### 4.2.3 Introducing a standard tabu list

The goal of both 2-opt and 3-opt operators is to provide a series of solutions iteratively reached throughout searching on several neighborhoods. The principle of the tabu list is to store some moves instead of storing all visited solutions. Indeed, because storing these solutions may induce the saturation of the memory-space, the tabu list is then limited to storing some inverse-moves (inverse-swaps) to avoid cycling and stagnation of solutions. In order to prevent the research process to lead toward the same local optima and so, the method can be trapped in these solutions, the tabu search is introduced. Also, the intensification strategy tries to find solutions throughout a series of neighborhoods provided with both 2-opt and 3-opt operators. Despite the improvements issued from these operators, it is interesting to propose a manner able to drive the search process through other unvisited neighborhoods.

In this study, because both 2-opt and 3-opt use swaps between tasks for iteratively generating a series of neighbors, we then add a tabu list that stores a list of temporarily inverse-swaps; that are the moves trying to avoid return to the solutions already visited.

### 4.3 Diversification strategy

Other operators, like local search using diversifications, can also be applied for improving the quality of the solutions. As used in Hifi and Michrafy (2006), a reactive search was proposed, where two complementary strategies are combined trying to jump from the current search space to a new one. Both (i) dropping strategy and (ii) rebuilding strategy are incorporated in order to favor some fixed object and to optimize the rest of the problem with free variables for completing the solution at hand.

Herein, such a strategy may be described as follows:

1. Construct a partial solution by removing some objects from the current solution. Hence, a partial (feasible) solution is built.

2. Complete the current partial solution reached at the first step by solving the subproblem with the free objects.

In this study, we adapt such a process which consists of dropping a subset of assigned tasks from the current sequence (i.e. the current solution). The dropping strategy tries to diversify the search process by degrading the quality of the current solution with the aim of avoiding stagnation. The partial solution is built and completed by applying the knapsack procedure, according to the new order associated with the free tasks. Indeed, the diversification strategy can be applied by using the Drop and Rebuild Operator (DRO), which can be summarized in what follows. Let $\hat{S}$ be the current solution, the DRO strategy is then applied in order to reduce the problem, by randomly fixing a subset of already assigned tasks of $\hat{S}$, as follows:

**Algorithm 1** - Drop and Rebuild operator (DRO)

**Input.** An instance of ST2P with a solution $\hat{S}$.
**Output.** A perturbed (improved) solution $S^{\star}$, a best solution of ST2P.

1: From the solution $\hat{S}$, drop $\beta\%$, $\beta \in ]0, 100[$, of the tasks according to the current order of the sequence and, let $\hat{S}_1$ be the first partial solution built with the rest of the tasks
2: Solve the reduced problem with the knapsack procedure (cf., Section 4.1) and let $S^{\star}$ be the complete achieved solution
3: Improve the current solution $S^{\star}$ by calling the intensification phase (cf., Section 4.2), including the already removed tasks
4: **return** $S^{\star}$

### 4.4 A look-ahead strategy

Combining path-relinking with a greedy procedure was initially proposed by Laguna and Marti (1999). Its goal is to intensify the search space of each given local solution. On the one hand, Resende, Marti, Gallegoc, and Duarte (2010) discussed different strategies that can be employed in order to build the path that should be created between an initial solution and the target solution. Among these strategies, creating randomness on the generated solutions through the path can also be reached by applying a greedy permutation between an element of the starting solution and another one randomly taken from the target solution. Truncated search and evolutionary strategies can also be used either for making a quick convergence

or for improving solutions. Generally, the path-relinking may start with $x$ and iteratively modifies it to provide the guiding solution $y$. One way toward $y$ is similar to build a series of neighbors that can take different directions. In fact, finding the neighbor solutions leading $y$ can be stated as follows:

1. Set $k = 0$ and let $x = Path_k(x, y)$ be the starting solution and $y = Path_r(x, y)$ be the guiding solution, where $r \geq 1$.

2. Determine an intermediate solution $Path_{k+1}(x, y)$, increment $k$ with one unit and set $x = Path_k(x, y)$.

3. Repeat step (2) till performing $r$ neighbor-solutions, where the latest one is closest to $y$.

**Algorithm 2** - A Look-Ahead (LA)

**Input.** Two solutions $x$ and $y$, where $f(x) \geq f(y)$.
**Output.** An internal solution $y_{Best}$ for ST2P.

1: Set $y_{Best} = \text{argmin}\{x, y\}$.
2: Set $k = 0$, $x = Path_k(x, y)$ and $y = Path_r(x, y)$, where $1 \leq r \leq n$ ($n$ denotes the size of the Hamming distance between both $x$ and $y$).
Let $O_x$ (resp. $O_y$) be a sequence representing tasks assigned to processors according to a given order related to $x$ (resp. $y$).
3: **repeat**
4: Let $O^{(p)} = O_x \cap O_y$ and $O^{(Rest)} = I \setminus O^{(p)}$, where $O^{(Rest)}$ denotes the set of different components between $O_x$ and $O_y$ ($|O^{(Rest)}|$ representing the standard Hamming distance).
*Initialization of the new neighbor* (namely $y'$): $\forall \ell \in O^{(p)}$, set $y'_\ell = x_\ell$.
5: For each task $\ell \in O^{(Rest)}$, do

(a) Set $y'_\ell = x_\ell$ and let $s$ be the $s$-th position of $y$ such that $y(s) = x_\ell$: set $y'_s = x_\ell$.

(b) Apply the knapsack procedure to complete/repair the partial solution $y'$ (containing all ordered tasks of $O^{(p)}$).

(c) Update the best solution, namely $y_{best}$, by comparing the provided solution to the best one related to the current neighborhood, i.e., according to $O^{(p)}$.

6: Let $y$ be the best solution reached in step (5):

(a) Improve the solution quality of $y$ by calling the intensification strategy (cf., Section 4.2) and update the best solution, namely $y_{best}$,

(b) Increment $k$ and set $x = Path_k(x, y)$.

7: **until** either $k = n$ or the stoping criteria is matched
8: **return** $y_{Best}$

On the other hand, another strategy based upon a look-ahead has been discussed in Al-douri, Hifi, and Zissimopoulos (2019), where instead of generating a single solution at

the current level, a set of neighbors is created and so, only one solution among the aforementioned solutions is chosen for continuing the construction of the best path. Indeed, because each neighbor has an objective value, then we choose the one who achieved the best objective value.

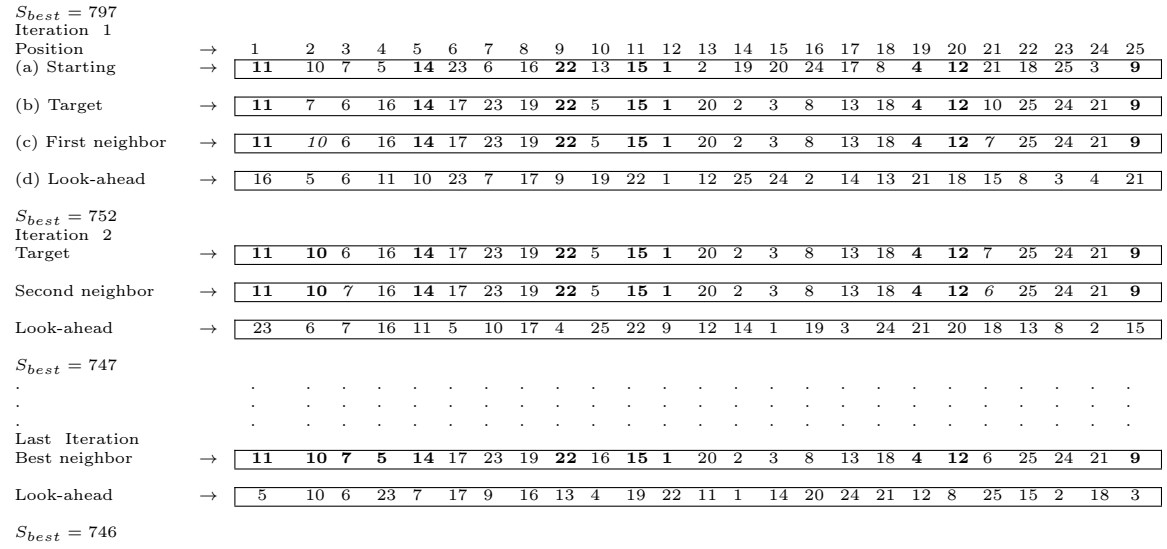


Figure 3: Illustration of the standard look-ahead strategy on an instance of SP2P containing 25 tasks: (a) denotes a starting solution, (b) denotes the (first best) target solution, (c) is the first configuration achieved, (d) the internal solution at one level and, (d) the final best solution.

In this work, we adapt the principle of the look-ahead strategy for simulating the path between the starting solution and the target one. The principle of the method follows: Let $x$ and $y$ be two solutions representing two sequences with their orders $O_x$ and $O_y$, respectively. In order to mimic steps from (1) to (3) above, we propose to build each intermediate solution (at level $k$) by using a look-ahead strategy. The look-ahead-based search investigates the outcomes of potential future levels in order to evaluate the quality of some search directions. The used look-ahead strategy is based upon the best objective function and it works as follows:

One can observe, on the one hand, that the above procedure evaluates a set of paths, where only the one realizing the best bound is chosen (cf., Step (d)); that is the path realizing an internal and intermediate solution at level (iteration) $k$. This manner is called a *look-ahead* strategy because, at each level, several neighbors (complete solutions) are built while only the favored solution with minimum objective value is chosen, in this case. On

the other hand, because the average runtime of the algorithm may grow, we then limit the exploration by introducing a *beam parameter* $\rho$, which serves to reduce the number of neighbors to explore.

Figure 3 shows the iterative search process, when using the look-ahead strategy, on an instance of `Set 1` (a set of small instances belonging to the benchmark instance in the experimental part − Section 5):

***Iteration 1.*** Both (a) starting and (b) target solutions were provided by the first phase of the method. The index of the task is marked in **bold-space** whenever that task has the same order in both solutions, in black (standard color) otherwise. By using the first neighbor with $\rho = 1$ (tasks marked in *italic* represent a part of the global neighborhood: indexes 2 and 21), the look-ahead provides a new solution with an objective value equal to 752; that is better than the starting one (797).

***Iteration 2.*** The solution related to the first neighbor is now considered as the target solution and the second neighborhood (tasks marked in *italic* denote a part of the global neighborhood: indexes 3 and 21) achieves a solution with an objective value equal to 747. It improves the quality of the upper bound provided up to now.

***Internal iteration.*** After several iterations, the search process generates the last neighborhood which reaches a final solution whose objective value is further improved (746), as illustrated in the last iteration.

### 4.5 An overview of the look-ahead-based method

Algorithm 3 illustrates the main steps of the proposed Look-Ahead strategy-Based Method (noted LA-BM). The input of LA-BM is an instance of ST2P and its output is a near-optimal solution $S^{\star}_{ST2P}$. The method starts by generating an initial solution $S_o$ provided following the knapsack order, assigns that solution to the best one ($S^{\star}_{ST2P}$) and improves it by applying both intensification (Section 4.2) and diversification strategies (Section 4.3). The main loop of LA-BM are represented by lines from 5 to 17. It repeats the selection of a couple of solutions $(x, y)$ (line 6) while line 7 applies the path-relinking using a look-ahead (Algorithm 2) for generating an intermediate solution (line 8). Next, a diversification procedure is applied to the current solution (line 14), where the destroying stage is combined with both repairing and exploring stages. The iterated process alternates between improvement,

**Algorithm 3** - Look-Ahead strategy-Based Method (LA-BM)

**Input.** An instance of ST2P.
**Output.** $S^{\star}_{ST2P}$, the best solution of ST2P.

1: Call the greedy knapsack procedure (cf., Section 4.1) for generating the starting solution $S_0 = \{o^{(1)}, \ldots, o^{(s)}, \ldots, o^{(Apha)}\}$ related to the achieved order.
2: Set $S^{\star}_{ST2P} \longleftarrow S_0$
3: Improve $S^{\star}_{ST2P}$ with intensification and diversification strategies (cf. Sections 4.2 and 4.3)
4: Set $S_{ST2P} \longleftarrow S^{\star}_{ST2P}$ and $iter \leftarrow 1$
5: **repeat**
6: Set $y \leftarrow S_{ST2P}$ and $x = S_0$
7: Apply the Look-Ahead$(x, y)$ (cf., Algorithm 2) and let $y'_{Best}$ be the best solution found
8: Set $S_{ST2P} \leftarrow y'_{Best}$ be the new solution
9: **if** $(f(S_{ST2P}) < f(S^{\star}_{ST2P}))$ **then**
10: Set $S^{\star}_{ST2P} \leftarrow S_{ST2P}$
11: $iter \leftarrow 1$
12: **else**
13: Set $iter = iter + 1$
14: Apply the diversification phase (cf., Algorithm 1) to $S_{ST2P}$ and let $y'_{ST2P}$ be the achieved solution
15: Set $S_0 = \text{argmax}\Big\{S^{\star}_{ST2P}, y'_{ST2P}\Big\}$ and $S_{ST2P} = \text{argmin}\Big\{S^{\star}_{ST2P}, y'_{ST2P}\Big\}$
16: **end if**
17: **until** $\big(iter \geq Iter_{\max}$ or the time limit is performed$\big)$
18: **return** $S^{\star}_{ST2P}$.

look-ahead, and diversification strategies, till matching the maximum number of iterations or performing the second stopping criteria related to the runtime limit. Finally (line 18), LA-BM returns $S^{\star}_{ST2P}$, the best solution found so far.

# 5 Computational results

The objective of the computational investigation is to assess the performance of the Look-Ahead strategy-based Method (noted LA-BM) by comparing its provided upper bounds (objective values: minimization problem) to the best-known upper bounds available in the literature. Indeed, its provided results are compared to those achieved by both Genetic Algorithm (noted GA) proposed in Kacem and Dammak (2014)([1]), the Reactive Search-Based Algorithm (noted RSBA) proposed in Aïder, Baatout, and Hifi (2020) and, the tight Lower Bound (noted LB) designed by Manaa and Chu (2010), as used in Kacem and Dammak (2014).

In this part, two sets of instances are considered: each set is composed of five groups,

[1]The code was provided by the first author for generating and testing the behavior of all methods on the same instances (using the same computer).

where each group is related to the type of instances considered (as suggested in Manaa and Chu (2010)). We note that the proposed method was coded in `C++` and run on an `Intel Pentium Core i7-8550U` with 1.99 GHz and 16 Gb of RAM (all methods were tested on the same computer).

| Type of task | Type 1 | Type 2 | Type 3 | Type 4 | Type 5 |
|---|---|---|---|---|---|
| $n1$ | $n$ | $n$ | $n$ | $n$ | $[n/2]$ |
| $n2$ | $[n/2]$ | $n$ | $[n/2]$ | $n$ | $[n/2]$ |
| $n12$ | $[n/2]$ | $[n/2]$ | $n$ | $n$ | $n$ |

Table 1: Characteristics of the used instances

As mentioned above, the used instances were generated by using Manaa and Chu's (2010) generator, where five types of instances are considered([2]), according to the number of tasks $n$ to use and the tasks assigned to both processors $P_1$ and $P_2$ and, the bi-processor (tasks simultaneously assigned to both $P_1$ and $P_2$):

- For small instances, the number of tasks $n$ is fixed to 10, to 20 for medium instances and, to 100, 500 and 1000 for large-scale instances: for each value, thirty instances are considered.
- The number $n_1$ (resp. $n_2$ and $n_{12}$) denotes tasks assigned to $P_1$ (resp. $P_2$ and $P_{12}$) that is generated following the values displayed in Table 1 such that $[x]$ denotes the integral value of $x$.
- The processing time related to the duration of the task $j$, noted $p_j$, is randomly generated in the discrete interval $\{1, \ldots, 50\}$.
- The release date $r_j$, related to task $j$, is randomly generated in the discrete interval $\{1, \ldots, k\}$, with $k = \alpha \times \frac{(s_{12}+(s_1+s_2))}{2}$ and $\alpha \in \{0.5, 1, 1.5\}$ (denoting the density of the instance) and, $s_1$ (resp. $s_2$ and $s_{12}$) denotes the overall duration related to tasks assigned to $P_1$ (resp. $P_2$ and $P_{12}$).

### 5.1 Parameter settings

Often, the behavior of a designed heuristic depends on adjustments used on all of its parameters. This means that some adjustments can degrade the quality of the solutions achieved and therefore, in this part, we try to find experimentally the fine values to assign. The

[2]All tested instances are publicly available for other researchers in the domain (`https://www.u-picardie.fr/eproad/`)

proposed LA-BM uses four main parameters/operators: (i) the 2-Opt and 3-Opt operators, (ii) the dropping parameter $\beta$ used by the dropping procedure and, (iii) the size $\rho$ related to the Hamming distance (path) used by the look-ahead strategy.

| Instances | | | | $V_2$ | | $V_3$ | | $V_4$ | |
|---|---|---|---|---|---|---|---|---|---|
| | $n = 20$ | LB | $V_1$ | Best | Avg | Best | Avg | Best | Avg |
| Type 1 | $\alpha$=0.5 | 354.50 | 395.80 | 369.40 | 378.70 | 368.00 | 370.50 | 362.10 | 367.80 |
| | $\alpha$=1 | 411.60 | 491.90 | 459.20 | 478.00 | 454.90 | 464.30 | 436.40 | 455.60 |
| | $\alpha$=1.5 | 536.30 | 600.30 | 569.20 | 588.10 | 563.80 | 569.60 | 551.80 | 567.30 |
| Type 2 | $\alpha$=0.5 | 318.60 | 377.20 | 346.30 | 359.50 | 338.50 | 345.90 | 332.90 | 342.70 |
| | $\alpha$=1 | 471.10 | 554.70 | 509.60 | 535.10 | 516.80 | 524.70 | 494.30 | 512.70 |
| | $\alpha$=1.5 | 703.50 | 776.70 | 743.60 | 762.60 | 731.40 | 739.40 | 721.10 | 740.60 |
| Type 3 | $\alpha$=0.5 | 392.80 | 440.40 | 412.50 | 427.80 | 409.20 | 413.20 | 403.20 | 410.20 |
| | $\alpha$=1 | 491.80 | 585.50 | 555.30 | 574.30 | 535.70 | 548.40 | 538.10 | 551.60 |
| | $\alpha$=1.5 | 670.70 | 775.50 | 740.40 | 760.20 | 717.50 | 734.70 | 716.70 | 733.50 |
| Type 4 | $\alpha$=0.5 | 443.40 | 514.60 | 487.60 | 500.50 | 475.90 | 483.40 | 468.60 | 478.60 |
| | $\alpha$=1 | 568.50 | 681.00 | 635.50 | 663.90 | 628.00 | 637.90 | 608.50 | 635.20 |
| | $\alpha$=1.5 | 843.30 | 971.90 | 918.70 | 951.40 | 898.80 | 911.00 | 893.20 | 917.50 |
| Type 5 | $\alpha$=0.5 | 287.10 | 329.30 | 307.80 | 318.50 | 306.80 | 310.40 | 299.50 | 307.20 |
| | $\alpha$=1 | 395.10 | 476.80 | 455.20 | 466.90 | 439.30 | 446.50 | 436.00 | 449.10 |
| | $\alpha$=1.5 | 643.00 | 731.00 | 686.80 | 711.30 | 681.90 | 692.60 | 661.80 | 682.50 |
| | Average | 502.087 | 580.173 | 546.473 | 565.120 | 537.767 | 546.167 | 528.280 | 543.473 |

Table 2: Behavior of the standard version of LA-BM with and without using 2-opt and/or 3-opt operators, on instances with $n = 20$ (medium instances).

### 5.1.1 Effect of 2-opt and 3-opt operators

In order to analyze the behavior of LA-BM when using the intensification strategy, four versions of the method are considered. These versions are tested on the small and medium instances. The first version (noted $V_1$) denotes the versions without operators, the second version ($V_2$) uses only the 2-opt operator (without a look-ahead), the third version (noted $V_3$) designs the version using the 3-opt operator while the fourth one (noted $V_4$) is the version using both the 2-opt and 3-opt operators. Moreover, because all tested versions (except $V_1$) are stochastic algorithms, then ten trials were considered for each of them. The achieved results, for the four versions, are reported in Table 2, where the runtime of each version is very small (less than 0.005 seconds). Also, in order to make a more complete comparison between the four versions of the algorithm, we fix $\beta$ to 10% (regarding the good behavior of the method with this value as discussed later in Section 5.1.2). Table 2 shows both best and global average upper bounds provided by the four versions $V_1$, $V_2$, $V_3$ and $V_4$ and, Manaa and Chu's tight lower bound. Columns 1 and 2 of the table display

the instance information: the number $n$ of tasks and the value $\alpha$ representing the density of each instance. Column 3 tallies the lower bound and column 4 the best average upper bounds provided by $V_1$. Columns 5 and 6 report the average best bounds and the global average bounds of all instances achieved by $V_2$. Columns 7 and 8 (resp. columns 9 and 10) tally the same values for $V_3$ (resp. $V_4$).

A thorough discussion of the provided results, for the four versions on the small and medium instances, displayed in Table 2, follows:

1. $V_1$ vs $V_2$: $V_2$ outperforms $V_1$, since $V_2$ is able to provide a better global average upper bound. On the one hand, $V_2$'s average best upper bound is equal to 546.473 while $V_1$ realizes an average best upper bound of 580.173. On the other hand, $V_2$'s experimental approximation ratio (EA($A(I)$)=$\frac{A(I)}{\text{LB(I)}}$), where $A(I)$ denotes the objective value (upper bound) provided by algorithm $A$ for an instance $I$ and, LB denotes the lower bound related to the same instance $I$, equal to 1.15 which is no better than that achieved by $V_2$; that is equal to 1.09. This means that the 2-opt operator has a good behavior, especially when the random aspect is integrated in the operator.

2. $V_2$ vs $V_3$: $V_3$ performs better than $V_2$. In this case, $V_3$ dominates $V_1$ since its average best global upper bound is equal to 537.767, which is better than that achieved by $V_2$ (i.e., 546.473). The experimental approximation ratio EA($V_3$) is now equal to 1.07, which becomes better than EA($V_3$) (that is equal to 1.09). In this case, the 3-opt operator becomes more efficient since it is based upon using the double calls of the 2-opt operator.

3. $V_4$ versus $V_3$: $V_4$ remains very competitive when comparing its provided results with those achieved by the best of the three versions ($V_3$ instead of $V_1$ and $V_2$). Indeed, on the one hand, combining both the 2-opt and 3-opt operators is able to enhance all the average (global) bounds (it becomes now equal to 528.280 when compared to $V_3$'s value of 537.767). On the other hand, EA($V_4$) is equal to 1.05 while EA($V_3$) provides an average value of 1.07.

Figure 4 illustrates the variation of the experimental approximation ratios reached by the four versions: $V_1$, $V_2$, $V_3$ and, $V_4$. One can observe that by combining both the 2-opt and 3-opt operators, the experimental approximation ratio decreases: it varies from 1.15 to 1.05.

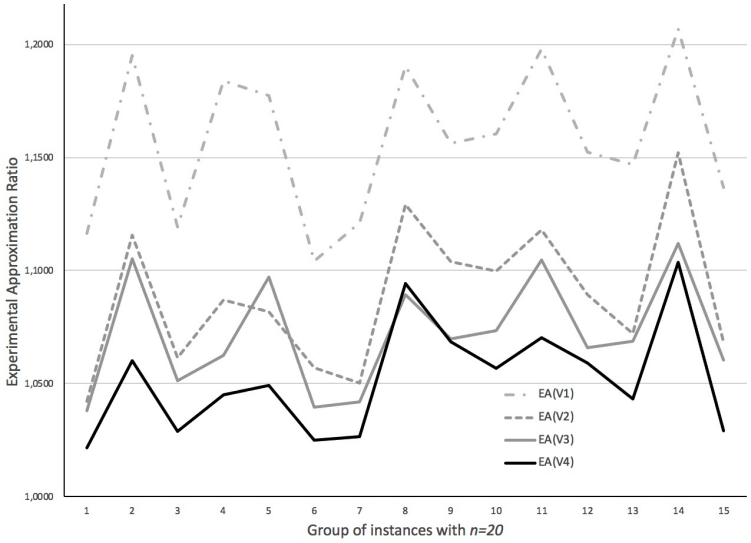


Figure 4: Variation of the experimental approximation ratios for the four versions of LA-BM: EA($V_1$), EA($V_2$), EA($V_3$) and EA($V_4$).

Hence, based on this first analysis, we can expect the good behavior of the proposed LA-BM when combining the specific $k$-opt operators with the look-ahead phase.

#### 5.1.2 Effect of the drop / rebuild operator

In order to evaluate the behavior of LA-BM according to the parameter $\beta$, we introduced a variation on the number of removing items in the discrete interval $\{10, 20, 30, 40, 50\}$, which represents an interval varying from soft dropping to hard dropping. Table 3 reports the average upper bounds achieved by LA-BM when varying the parameter $\beta$ for the same instances tested in Section 5.1.1. Columns 1 and 2 of the table report the instance information: the number $n$ of tasks and the value $\alpha$ related to the instances represented in Table 3. Columns 3 and 4 show the average bounds of all instances of each group and the average runtime consumed when setting $\beta$ to 10%. Columns 5 and 6 display the average bounds and the average runtime for $\beta = 20\%$, while columns 7 and 8 (resp columns 9 and 10 and, columns 11 and 12) report the average bounds and its average runtime when fixing $\beta$ to 30% (resp $\beta$ to 40% and, $\beta$ to 50%).

From Table 3, one can observe what follows:

1. LA-BM achieves better average bounds for $\beta = 10\%$ (last line, column 3 in bold-space).

2. When the value of $\beta$ increases (i.e., $\beta$ varies from 20% to 50%), the used perturbation

is unable to reach better average bounds. We believe that for these largest values of $\beta$, LA-BM may explore a largest space and so, the search process is not able to locate good directions for improving the quality of some visited solutions.

| Instances | | Variation of $\beta$ 10% | | 20% | | 30% | | 40% | | 50% | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n = 20$ | Av | cpu | Av | cpu | Av | cpu | Av | cpu | Av | cpu |
| Type 1 | $\alpha = 0.5$ | 359.10 | 0.0049 | 359.40 | 0.0148 | 359.20 | 0.0192 | 359.10 | 0.0248 | 360.10 | 0.0284 |
| | $\alpha = 1$ | 423.80 | 0.0063 | 433.80 | 0.0225 | 432.30 | 0.028 | 435.00 | 0.0314 | 430.40 | 0.0395 |
| | $\alpha = 1.5$ | 543.30 | 0.0054 | 549.20 | 0.0157 | 550.70 | 0.0187 | 546.60 | 0.0238 | 549.30 | 0.0318 |
| Type 2 | $\alpha = 0.5$ | 325.00 | 0.0055 | 328.70 | 0.0218 | 328.90 | 0.0267 | 326.80 | 0.0329 | 328.00 | 0.0391 |
| | $\alpha = 1$ | 486.50 | 0.0053 | 493.80 | 0.0183 | 491.50 | 0.0244 | 488.10 | 0.0294 | 488.30 | 0.0346 |
| | $\alpha = 1.5$ | 712.90 | 0.0063 | 718.50 | 0.0266 | 720.70 | 0.0368 | 719.30 | 0.0434 | 719.50 | 0.0503 |
| Type 3 | $\alpha = 0.5$ | 397.20 | 0.0082 | 401.50 | 0.0316 | 399.70 | 0.0362 | 402.50 | 0.0439 | 399.50 | 0.0549 |
| | $\alpha = 1$ | 517.90 | 0.0064 | 529.70 | 0.03 | 523.90 | 0.0428 | 527.70 | 0.0482 | 525.70 | 0.0597 |
| | $\alpha = 1.5$ | 690.90 | 0.007 | 700.00 | 0.0298 | 700.60 | 0.0374 | 703.90 | 0.0449 | 700.40 | 0.0554 |
| Type 4 | $\alpha = 0.5$ | 457.10 | 0.0081 | 463.30 | 0.0387 | 461.60 | 0.047 | 463.40 | 0.0608 | 464.20 | 0.0827 |
| | $\alpha = 1$ | 595.80 | 0.0085 | 608.50 | 0.0416 | 601.70 | 0.0545 | 603.70 | 0.0559 | 603.40 | 0.0687 |
| | $\alpha = 1.5$ | 870.00 | 0.0124 | 883.80 | 0.0523 | 884.80 | 0.0638 | 880.10 | 0.0703 | 881.30 | 0.0747 |
| Type 5 | $\alpha = 0.5$ | 295.80 | 0.005 | 297.50 | 0.016 | 297.30 | 0.0208 | 297.60 | 0.025 | 296.40 | 0.0292 |
| | $\alpha = 1$ | 419.00 | 0.003 | 429.60 | 0.0075 | 429.10 | 0.009 | 427.40 | 0.0116 | 428.00 | 0.0141 |
| | $\alpha = 1.5$ | 652.50 | 0.0054 | 657.50 | 0.0146 | 656.00 | 0.0202 | 654.60 | 0.0235 | 657.70 | 0.0276 |
| | Average | **516.45** | 0.0065 | 523.65 | 0.0255 | 522.53 | 0.0324 | 522.39 | 0.0380 | 522.15 | 0.0460 |

Table 3: Effect of the dropping parameter $\beta$.

Moreover, before fixing the final value of $\beta$, for extending the experimental part, we introduce a statistical analysis on the average bounds achieved by the five versions of the algorithm following the variation of $\beta$'s value. In the comparative study, both the *Sign test* and the *Wilcoxon signed-rank test* statistics are considered. Of course, the Wilcoxon signed-rank test is an alternative study employing the p-value for indicating how the version performs better than another one. We then use the following hypothesis: $H_0$: Algo$_{\beta_1}$ - Algo$_{\beta_2} = \mu$, where $\beta_1 \neq \beta_2$, to express that algorithm Algo$_{\beta_1}$ performs better than Algo$_{\beta_2}$ and, the hypothesis $\overline{H}_0$: algorithm Algo$_{\beta_2}$ is better than Algo$_{\beta_1}$ to express the rejection of the hypothesis $H_0$. In this work, the smallest the average bound and the greater the number of better bounds, the better the corresponding version of Algo.

Table 4 reports, by varying $\beta_1$ and $\beta_2$ ($\beta_1 \neq \beta_2$), the statistical study on all instances with $n = 20$ by using the *sign rank test* (the detailed results containing the average bounds are those illustrated in Table 3). The different values related to $\mu$ represent what follows: $\mu_1^{(k)} = \mathrm{V}_{10\%}$ vs $\mathrm{V}^{(k)}$, $k = 1, \ldots, 4$ and $V^{(k)}$ denotes the versions $\mathrm{V}_{20\%}, \ldots \mathrm{V}_{50\%}$ and, $\mu_2 = \mathrm{V}_{20\%}$ vs $\mathrm{V}_{30\%}$, $\mu_3 = \mathrm{V}_{30\%}$ vs $\mathrm{V}_{40\%}$ and, $\mu_4 = \mathrm{V}_{40\%}$ vs $\mathrm{V}_{50\%}$. Columns from 1 to 4 display the

| | $\mu_1^{(1)}$ | $\mu_1^{(2)}$ | $\mu_1^{(3)}$ | $\mu_1^{(4)}$ | $\mu_2$ | $\mu_3$ | $\mu_4$ |
|---|---|---|---|---|---|---|---|
| p-value (Sign test) | <0.0001 | <0.0001 | <0.0001 | <0.0001 | 0.941 | 0.696 | 0.304 |
| $N^+$ | 15 | 15 | 14 | 15 | 5 | 7 | 9 |
| $N^-$ | 0 | 0 | 0 | 0 | 10 | 8 | 6 |
| $N^=$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| p-value (Wilcoxon) | <0.0001 | <0.0001 | <0.00005 | <0.0001 | 0.938 | 0.555 | 0.577 |

Table 4: p-values for both *sign* and *Wilcoxon rank* tests on the instances with $n = 20$ with the significance level $\theta = 0.05$.

statistical results of LA-BM (without using the look-ahead strategy which will be discussed in Section 5.1.3) when varying $\beta$: line 1 displays the p-value corresponding to the sign test, line 2 tallies the number of times that the first version of the algorithm dominates the second version (resp. line 3 reports the number of times that the first version is dominated by the second one and, line 4 shows the number of times that both versions match the same values) and line 5 tallies the p-value related to the rank sign test. From Table 4, we observe what follows:

- For $\mu_1^{(1)}$ the p-value related to the sign test (resp. Wilcoxon rank-test) is smallest to the significance level $\theta = 0.05$, indicating that $V_{10\%}$ performs better than $V_{20\%}$ (accepting the hypothesis $H_0$). Increasing the value of $\beta$ doesn't improve the quality of the average bounds: p-values related to $\mu_1^{(k)}$, $k \geq 2$, varies from 0.00005 to 0.0001, which means that $V_{10\%}$ evolves better than the other versions.

- Regarding the comparative study between the four other versions (i.e., $V_{20\%}$, ..., $V_{50\%}$), $V_{30\%}$ performs better than $V_{20\%}$ according to Wilcoxon's p-value related to $\mu_2$ (equals to 0.938), $V_{40\%}$ performs better than $V_{30\%}$ according to Wilcoxon's p-value related to $\mu_3$ (equals to 0.555) and, $V_{50\%}$ outperforms $V_{40\%}$ regarding $\mu_4$ (the Wilcoxon's p-value is equal to 0.577).

- The number of occasions that $V_{10\%}$, when compared to the rest of the versions of the algorithm, achieves the best average bounds varies from 14 to 15 for $N^+$ representing $\mu_1^{(k)}$, $k \geq 1$, while it is equal to 9 for $\mu_4$. This means that $V_{50\%}$ represents the second best version of the algorithm even if the average runtime becomes more important in this case.

Therefore, because we seek solutions with high quality (smallest values) and with more best average bounds, we then fix $\beta = 10\%$ for the rest of the experimental part.

### 5.1.3 Effect of the look-ahead strategy

In this section, we evaluate the behavior of the proposed method when introducing the look-ahead strategy. We recall that the aforementioned strategy uses a *beam parameter* $\rho$; that is used for limiting the size of the neighborhood to explore.

First, Table 5 reports the average bounds achieved by LA-BM when varying the value of $\rho$ in the interval represented below (obviously, other values have been tested without providing better average bounds). As for the above tables, column 1 refers to the instance's information, columns from 2 to 7, under the value assigned to $\alpha$, display the bounds reached by the algorithm and, the last line labeled "Av" shows the global average value over all tested instances with $n = 20$.

| Instances | | Variation of $\rho$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4% | | 8% | | 12% | | 16% | | 20% | |
| | $n = 20$ | Av | cpu | Av | cpu | Av | cpu | Av | cpu | Av | cpu |
| Type 1 | $\alpha = 0.5$ | 354.50 | 1.659 | 354.50 | 0.595 | 354.50 | 0.142 | 354.70 | 0.120 | 354.60 | 0.094 |
| | $\alpha = 1$ | 411.60 | 1.611 | 411.60 | 0.594 | 413.30 | 0.150 | 411.80 | 0.120 | 412.00 | 0.095 |
| | $\alpha = 1.5$ | 536.30 | 1.707 | 536.30 | 0.628 | 536.60 | 0.158 | 536.30 | 0.113 | 536.30 | 0.103 |
| Type 2 | $\alpha = 0.5$ | 318.70 | 1.273 | 318.70 | 0.684 | 320.70 | 0.186 | 321.20 | 0.148 | 320.50 | 0.130 |
| | $\alpha = 1$ | 471.10 | 1.315 | 471.10 | 0.713 | 474.40 | 0.191 | 475.00 | 0.156 | 475.50 | 0.128 |
| | $\alpha = 1.5$ | 703.50 | 1.392 | 703.50 | 0.730 | 703.90 | 0.194 | 703.50 | 0.153 | 703.50 | 0.136 |
| Type 3 | $\alpha = 0.5$ | 392.90 | 1.311 | 392.90 | 0.689 | 395.00 | 0.188 | 393.80 | 0.161 | 393.60 | 0.138 |
| | $\alpha = 1$ | 491.80 | 1.294 | 491.80 | 0.681 | 496.40 | 0.186 | 493.30 | 0.155 | 493.30 | 0.141 |
| | $\alpha = 1.5$ | 671.00 | 1.373 | 671.00 | 0.711 | 673.10 | 0.199 | 672.50 | 0.159 | 671.50 | 0.137 |
| Type 4 | $\alpha = 0.5$ | 443.60 | 1.830 | 443.60 | 0.940 | 451.10 | 0.262 | 449.50 | 0.216 | 451.20 | 0.172 |
| | $\alpha = 1$ | 568.50 | 1.843 | 568.50 | 0.998 | 573.30 | 0.263 | 573.30 | 0.222 | 573.40 | 0.176 |
| | $\alpha = 1.5$ | 843.30 | 1.956 | 843.30 | 1.032 | 850.30 | 0.267 | 846.80 | 0.220 | 848.40 | 0.172 |
| Type 5 | $\alpha = 0.5$ | 287.60 | 1.505 | 287.60 | 0.528 | 289.80 | 0.142 | 289.70 | 0.100 | 290.20 | 0.081 |
| | $\alpha = 1$ | 395.10 | 1.576 | 395.10 | 0.548 | 401.80 | 0.145 | 401.20 | 0.108 | 402.80 | 0.086 |
| | $\alpha = 1.5$ | 643.10 | 1.630 | 643.10 | 0.602 | 644.60 | 0.148 | 644.80 | 0.113 | 645.60 | 0.087 |
| | Av | **502.17** | 1.552 | **502.17** | **0.712** | 505.25 | 0.188 | 504.49 | 0.151 | 504.83 | 0.125 |

Table 5: Effect of the look-ahead strategy

From Table 5, we observe what follows: LA-BM seems more efficient for both $\rho = 4\%$ and $\rho = 8\%$, especially when comparing its results to the results provided with the other values of $\rho$. In this case, on the one hand, the global average solution value over the tested instances, for the ten trials, is equal to 502.17 (last line, columns 3 and 4 of Table 5); that is, the smallest global average value among all displayed values. On the other hand, the average runtime related to the algorithm with $\rho = 8\%$ is faster than that of the algorithm with $\rho = 4\%$. Therefore, because for large-scale instances, the average runtime can grow

exponentially and so, we fix $\rho$ to 8% for the rest of the experimental part.

Second, before fixing the final values of $\rho$, we also introduce a statistical analysis on the average best solution values achieved by the five versions, where $\rho$ varies from 4% to 20% with a path of four%. Both the *sign test* and the *Wilcoxon signed-rank test* statistics are considered. As described above, we set the hypothesis $H_0$: $\text{Algo}_{\rho_1}$ - $\text{Algo}_{\rho_2} = \nu$, where $\rho_1 \neq \rho_2$, to express that algorithm $\text{Algo}_{\rho_1}$ performs better than $\text{Algo}_{\rho_2}$ and, the hypothesis $\overline{H}_0$: algorithm $\text{Algo}_{\rho_2}$ performs better than $\text{Algo}_{\rho_1}$ to express the rejection of the hypothesis $H_0$.

Table 6 shows, by varying $\rho_1$ and $\rho_2$ ($\rho_1 \neq \rho_2$), the statistical study on instances with $n = 20$ by using both the *sign test* and the *sign rank test* (the detailed results containing the average bounds for the five versions are reported in Table 5). Columns from 1 to 5 display the statistical results of the method when varying $\rho$: line 1 (resp. line 5) tallies the p-value corresponding to the sign test (resp. rank sign test) and, line 2 (resp. line 3) reports the number of times that the first algorithm dominates (resp. is dominated by) the second one.

| | $\nu_1$ | $\nu_2$ | $\nu_3$ | $\nu_4$ |
|---|---|---|---|---|
| p-value (Sign test) | 1 | <0.0001 | <0.00005 | <0.00005 |
| $N^+$ | 0 | 15 | 15 | 15 |
| $N^-$ | 15 | 0 | 0 | 0 |
| p-value (Wilcoxon) | 1 | <0.00005 | 0.001 | 0.001 |

Table 6: Statistical analysis when varying the parameter $\rho$: p-values for both *sign* and *Wilcoxon rank* tests on the instances with $n = 20$ with the significance level $\theta = 0.05$.

From Table 6 one can observe what follows:

- For $\nu_1$ (representing $\text{Algo}_{4\%}$ vs $\text{Algo}_{8\%}$), the p-value related to the sign test (resp. Wilcoxon test) is greatest to the significance level $\theta = 0.05$, indicating that $\text{Algo}_{8\%}$ performs better than $\text{Algo}_{4\%}$ (rejecting the hypothesis $H_0$).

- Increasing the value of $\rho$ doesn't improve the quality of the algorithm's average bounds: p-values related to $\nu_2$ ($\text{Algo}_{8\%}$ vs $\text{Algo}_{12\%}$), $\nu_3$ ($\text{Algo}_{8\%}$ vs $\text{Algo}_{16\%}$) and, $\nu_4$ ($\text{Algo}_{8\%}$ vs $\text{Algo}_{20\%}$) vary from 0.00005 to 0.001, which means that $\text{Algo}_{8\%}$ evolves better than the other versions.

- The number of occasions that $\text{Algo}_{8\%}$, when compared to all versions, matches the best average bounds (the values related to $N^-$ for $\nu_1$ and $N^+$ for $\nu_2$, $\nu_3$ and $\nu_4$) is equal to 15.

Therefore, because we are interested in high-quality solutions, we then fix $\rho = 4\%$ for the rest of the paper.

| | | | GA | | | RSBA | | | LH-BM | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n = 10$ | | LB | UB | Av. UB | $T_{GA}$ | UB | Av. UB | $T_{RSBA}$ | UB | Av. UB | $T_{LA-BM}$ |
| Type 1 | $\alpha$=0.5 | 400.90 | 441.30 | 467.80 | 0.068 | 407.20 | 407.20 | 0.012 | 400.90 | 400.90 | 0.620 |
| | $\alpha$=1 | 478.70 | 540.90 | 575.50 | 0.065 | 496.40 | 496.40 | 0.011 | 478.70 | 478.70 | 0.813 |
| | $\alpha$=1.5 | 789.30 | 845.50 | 907.20 | 0.073 | 797.90 | 797.90 | 0.011 | 789.30 | 789.30 | 0.768 |
| Type 2 | $\alpha$=0.5 | 402.40 | 519.10 | 556.00 | 0.094 | 476.60 | 476.60 | 0.031 | 402.80 | 403.10 | 0.525 |
| | $\alpha$=1 | 651.00 | 738.10 | 810.30 | 0.103 | 651.10 | 651.10 | 0.030 | 651.10 | 651.10 | 0.859 |
| | $\alpha$=1.5 | 914.80 | 1002.70 | 1067.20 | 0.093 | 925.90 | 925.90 | 0.033 | 914.80 | 914.80 | 0.579 |
| Type 3 | $\alpha$=0.5 | 494.90 | 594.70 | 640.20 | 0.104 | 528.50 | 528.50 | 0.028 | 494.90 | 494.90 | 0.696 |
| | $\alpha$=1 | 664.00 | 841.30 | 895.20 | 0.094 | 696.60 | 696.60 | 0.029 | 664.10 | 664.10 | 0.541 |
| | $\alpha$=1.5 | 924.80 | 1062.80 | 1125.40 | 0.079 | 936.10 | 936.10 | 0.035 | 924.80 | 924.80 | 0.560 |
| Type 4 | $\alpha$=0.5 | 547.60 | 690.90 | 751.30 | 0.118 | 582.10 | 582.10 | 0.035 | 548.70 | 549.00 | 0.684 |
| | $\alpha$=1 | 732.20 | 959.10 | 1025.80 | 0.115 | 781.70 | 781.70 | 0.032 | 732.20 | 732.20 | 0.832 |
| | $\alpha$=1.5 | 674.50 | 767.10 | 811.60 | 0.093 | 681.20 | 681.20 | 0.040 | 674.50 | 674.50 | 1.149 |
| Type 5 | $\alpha$=0.5 | 373.00 | 444.80 | 475.30 | 0.073 | 397.00 | 397.00 | 0.010 | 373.60 | 373.60 | 0.466 |
| | $\alpha$=1 | 545.00 | 655.40 | 709.40 | 0.063 | 560.80 | 560.80 | 0.024 | 545.00 | 545.20 | 0.459 |
| | $\alpha$=1.5 | 595.50 | 667.70 | 712.30 | 0.061 | 601.60 | 601.60 | 0.011 | 595.50 | 595.50 | 0.476 |
| | Average | 612.57 | 718.09 | 768.70 | 0.086 | 634.71 | 634.71 | **0.025** | **612.73** | **612.78** | 0.668 |
| $n = 20$ | | LB | UB | Av. UB | $T_{GA}$ | UB | Av. UB | $T_{RSBA}$ | UB | Av. UB | $T_{LA-BM}$ |
| Type 1 | $\alpha$=0.5 | 354.50 | 405.80 | 427.70 | 0.162 | 359.10 | 361.80 | 0.005 | 354.50 | 354.50 | 0.595 |
| | $\alpha$=1 | 411.60 | 523.40 | 567.10 | 0.165 | 423.80 | 439.50 | 0.006 | 411.60 | 411.60 | 0.594 |
| | $\alpha$=1.5 | 536.30 | 656.10 | 691.30 | 0.163 | 543.30 | 553.90 | 0.005 | 536.30 | 536.30 | 0.628 |
| Type 2 | $\alpha$=0.5 | 318.60 | 466.00 | 491.50 | 0.258 | 325.00 | 331.20 | 0.005 | 318.70 | 318.80 | 0.684 |
| | $\alpha$=1 | 471.10 | 630.60 | 666.40 | 0.257 | 486.50 | 500.90 | 0.005 | 471.10 | 471.10 | 0.713 |
| | $\alpha$=1.5 | 703.50 | 838.40 | 882.30 | 0.259 | 712.90 | 726.00 | 0.006 | 703.50 | 703.50 | 0.730 |
| Type 3 | $\alpha$=0.5 | 392.80 | 519.90 | 541.50 | 0.246 | 397.20 | 403.30 | 0.008 | 392.80 | 392.80 | 0.689 |
| | $\alpha$=1 | 491.80 | 690.50 | 722.40 | 0.247 | 517.90 | 534.20 | 0.006 | 491.80 | 491.80 | 0.681 |
| | $\alpha$=1.5 | 670.70 | 842.50 | 884.50 | 0.245 | 690.90 | 709.00 | 0.007 | 671.00 | 671.00 | 0.711 |
| Type 4 | $\alpha$=0.5 | 443.40 | 611.40 | 640.00 | 0.357 | 457.10 | 466.50 | 0.008 | 443.60 | 443.60 | 0.940 |
| | $\alpha$=1 | 568.50 | 810.20 | 853.50 | 0.3578 | 595.80 | 612.80 | 0.009 | 568.50 | 568.50 | 0.998 |
| | $\alpha$=1.5 | 843.30 | 1059.20 | 1107.30 | 0.358 | 870.00 | 892.10 | 0.012 | 843.30 | 843.30 | 1.032 |
| Type 5 | $\alpha$=0.5 | 287.10 | 391.30 | 413.10 | 0.157 | 295.80 | 300.70 | 0.005 | 287.60 | 287.60 | 0.528 |
| | $\alpha$=1 | 395.10 | 549.30 | 582.50 | 0.164 | 419.00 | 434.30 | 0.003 | 395.10 | 395.10 | 0.548 |
| | $\alpha$=1.5 | 643.00 | 755.80 | 793.60 | 0.158 | 652.50 | 664.50 | 0.005 | 643.10 | 643.10 | 0.602 |
| | Average | 502.09 | 650.03 | 684.31 | 0.237 | 516.45 | 528.71 | **0.007** | **502.17** | **502.17** | 0.712 |

Table 7: Performance of LA-BM vs GA and RSBA on instances of Set 1: small and medium instances.

## 5.2 Behavior of LA-BM vs available methods (Set 1)

First, in order to evaluate the performance of the proposed method LA-BM, we compare its provided results to those achieved by GA (Genetic Algorithm − Kacem and Dammak (2014)),

Manaa and Chu's (2010) tight lower bound (LB) and the more recent algorithm RSBA (Reactive Search-Based Algorithm − Aïder, Baatout, and Hifi (2020)).

Table 7 reports the bounds related to LB and those achieved by both GA, RSBA, and LA-BM. Column 3 of the table shows the average lower bound (LB) related to each group of tested instances of Set 1 (with $n = 10$ and $n = 20$). Column 4 (resp. column 5 and column 6) tallies the best average objective value achieved by GA (resp. the average value over the ten trials and the average runtime over the ten trials). Column 7 (resp. column 8 and column 9) reports the best average objective value achieved by RSBA (resp. the average values for all trials and the average runtime needed for the same trials) while column 10 (resp. column 11 and column 12) reports the best average objective value achieved by LA-BM (resp. both average values and average runtime needed for the same trials). Finally, the last line of the table displays the average values of all values represented in each column (we note that the value in "boldface" (last line of the table) means that the best (average) solution values have been provided by the considered algorithm).

In what follows, we comment results of Table 7:

1. For the small instances (with $n = 10$),

   (a) LA-BM outperforms GA even when considering the global average value (the last line of the table of the first block) corresponding to the average solution values over the ten trials of both GA and LA-BM. Indeed, LA-BM achieves an average global value of 612.73 while GA provides an average global value of 768.70. In this case, the gap between both values is closest to 156 units even GA's average runtime remains slightly smaller than that of LA-BM.

   (b) RSBA performs better than GA: both its global average value and its global average runtime are better than those of GA (601.60 instead of 634.71 and, 0.086 sec instead of 0.025 sec, respectively).

   (c) LA-BM outperforms both GA and RSBA. Indeed, LA-BM's global average value (612.78) is better (smallest) than those achieved by both GA (768.70) and RSBA (634.71). Because RSBA is a core of the proposed LA-BM, one can observe that its global average runtime grows more and more.

2. For the medium-sized instances (with $n = 20$), the same phenomenon can be observed. Indeed, the best LA-BM's average global value (502.17) is better (smallest) than those

achieved by both GA (650.03) and RSBA (516.45). However, as expected, LA-BM's global average runtime remains higher (0.712 sec) than that required for GA (0.237 sec) and RSBA (0.007 sec), for the medium instances, even if the average runtime is less than one second.

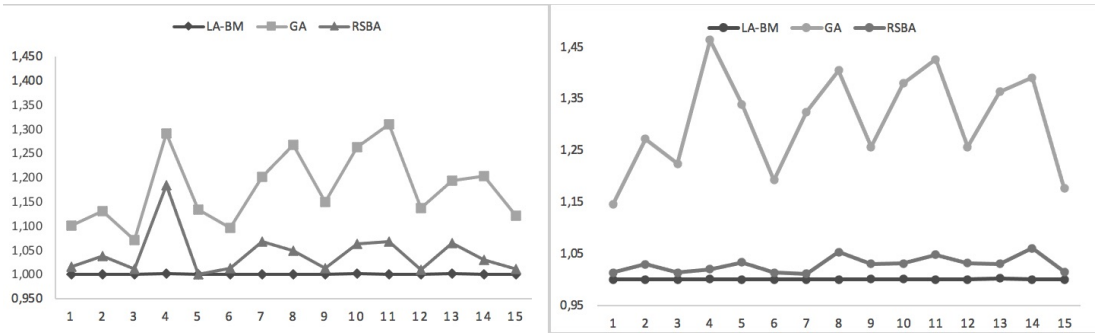


Figure 5: Variation of the experimental approximation ratios for the three tested algorithms GA, RSBA and LA-BM: (i) the left-side of the figure for $n = 10$ and, (ii) the right-side of the figure for $n = 20$.

Figure 5 shows the variation of the experimental approximation ratios achieved by the three tested methods: GA, RSBA, and LA-BM. As we can show from that figure, LA-BM's experimental approximation ratio is often better than those reached by both GA and RSBM; in this case, it varies from 1 to 1.002.

### 5.3 Behavior of LA-BM vs available methods (Set 2)

In this section, LA-BM's behavior is analyzed on the instances of Set 2 which contains three groups representing the large-scale instances: a first group with $n = 100$ tasks, a second group with $n = 500$ tasks and, a third group with $n = 1000$ tasks. Its provided results are also compared to those achieved by the best method available in the literature: GA, RSBA, and Manaa and Chu's lower bound (LB). Table 8 reports the average bounds provided by LA-BM, GA, RSBA, and LB on the instances of `Set 2`, where all its information are the same as used in Table 7 (there are three blocks, where each block corresponds to each group of `Set 2`).

In what follows, we comment on the results of Table 8:

| | | | GA | | | RSBA | | LH-BM | |
|---|---|---|---|---|---|---|---|---|---|
| $n = 100$ | | LB | UB | Av | $T_{GA}$ | UB | Av | $T_{RSBA}$ | UB | Av | $T_{LA-BM}$ |
| Type1 | $\alpha$=0.5 | 3 708.60 | 5 649.80 | 5 839.70 | 4.04 | 3 742.40 | 3 748.40 | 0.42 | 3 711.50 | 3 714.30 | 7.86 |
| | $\alpha$=1 | 4 885.80 | 7 711.70 | 7 935.40 | 4.01 | 5 167.30 | 5 261.30 | 0.40 | 4 910.90 | 4 960.20 | 7.81 |
| | $\alpha$=1.5 | 7 465.60 | 10 260.10 | 10 537.20 | 4.05 | 7 508.40 | 7 595.90 | 0.41 | 7 467.80 | 7 472.20 | 8.06 |
| Type2 | $\alpha$=0.5 | 3 793.10 | 6 753.60 | 6 940.80 | 6.50 | 4 875.80 | 4 877.10 | 0.37 | 3 843.50 | 3 865.30 | 11.02 |
| | $\alpha$=1 | 6 113.90 | 9 581.90 | 9 770.80 | 6.49 | 6 463.10 | 6 609.10 | 0.55 | 6 161.10 | 6 231.70 | 11.30 |
| | $\alpha$=1.5 | 9 374.60 | 12 641.50 | 12 958.60 | 6.49 | 9 518.90 | 9 621.80 | 0.37 | 9 376.10 | 9 393.70 | 11.37 |
| Type3 | $\alpha$=0.5 | 4 931.60 | 7 617.00 | 7 805.90 | 6.44 | 5 012.10 | 5 026.50 | 0.60 | 4 931.80 | 4 934.20 | 11.00 |
| | $\alpha$=1 | 6 181.40 | 10 312.30 | 10 570.10 | 6.41 | 6 790.50 | 6 896.20 | 0.42 | 6 381.50 | 6 460.50 | 11.43 |
| | $\alpha$=1.5 | 9 019.70 | 12 908.80 | 13 181.30 | 6.42 | 9 134.50 | 9 285.40 | 0.55 | 9 041.70 | 9 078.30 | 11.36 |
| Type4 | $\alpha$=0.5 | 4 981.60 | 8 821.40 | 9 032.20 | 10.25 | 6 073.30 | 6 079.30 | 0.68 | 5 101.20 | 5 122.80 | 14.59 |
| | $\alpha$=1 | 7 270.70 | 12 010.30 | 12 265.50 | 9.58 | 8 098.70 | 8 214.40 | 0.67 | 7 587.50 | 7 697.70 | 15.38 |
| | $\alpha$=1.5 | 10 918.80 | 15 372.10 | 15 769.10 | 9.66 | 11 120.40 | 11 259.60 | 0.77 | 10 961.30 | 11 035.00 | 15.45 |
| Type5 | $\alpha$=0.5 | 3 855.90 | 6 204.30 | 6 394.10 | 3.97 | 4 383.80 | 4 386.20 | 0.27 | 3 866.20 | 3 871.70 | 7.45 |
| | $\alpha$=1 | 4 951.80 | 8 257.70 | 8 476.00 | 3.96 | 5 383.80 | 5 468.50 | 0.41 | 5 124.80 | 5 200.10 | 7.62 |
| | $\alpha$=1.5 | 7 378.20 | 10 436.40 | 10 748.60 | 3.97 | 7 408.70 | 7 473.10 | 0.36 | 7 379.90 | 7 395.30 | 7.77 |
| | Average | 6322.087 | 9 635.93 | 9 881.69 | 6.15 | 6 712.11 | 6 786.85 | 0.48 | **6 389.79** | **6 428.87** | 10.63 |
| $n = 500$ | | LB | UB | Av | $T_{GA}$ | UB | Av | $T_{RSBA}$ | UB | Av | $T_{LA-BM}$ |
| Type1 | $\alpha$=0.5 | 18467.40 | 30480.90 | 30795.60 | 168.19 | 18937.30 | 19131.90 | 11.06 | 18526.60 | 18552.10 | 121.01 |
| | $\alpha$=1 | 24451.50 | 41949.80 | 42429.60 | 168.17 | 28185.70 | 28520.60 | 11.16 | 26943.80 | 27204.90 | 120.87 |
| | $\alpha$=1.5 | 36495.10 | 53447.70 | 54128.90 | 90.20 | 39208.70 | 39587.50 | 6.33 | 38409.40 | 38705.00 | 120.84 |
| Type2 | $\alpha$=0.5 | 18669.00 | 36427.90 | 36819.70 | 147.75 | 24866.90 | 24980.30 | 8.93 | 20460.40 | 20544.70 | 185.04 |
| | $\alpha$=1 | 30397.70 | 50443.20 | 51160.30 | 245.19 | 34720.30 | 35304.60 | 13.18 | 33732.90 | 33906.80 | 185.26 |
| | $\alpha$=1.5 | 45492.80 | 65196.20 | 65816.10 | 165.81 | 48927.10 | 49443.70 | 9.77 | 48096.10 | 48469.10 | 185.37 |
| Type3 | $\alpha$=0.5 | 24422.20 | 40219.20 | 40675.80 | 159.43 | 24992.10 | 25184.40 | 7.69 | 24484.30 | 24511.50 | 184.76 |
| | $\alpha$=1 | 30345.30 | 54303.20 | 54890.10 | 141.28 | 35747.30 | 36164.10 | 7.47 | 34528.20 | 34787.20 | 184.58 |
| | $\alpha$=1.5 | 45402.20 | 68597.60 | 69516.80 | 196.38 | 48813.50 | 49551.30 | 9.89 | 48440.00 | 48828.40 | 184.54 |
| Type4 | $\alpha$=0.5 | 24742.10 | 46583.10 | 47127.50 | 289.12 | 30913.90 | 31003.40 | 14.04 | 25848.50 | 25998.70 | 252.80 |
| | $\alpha$=1 | 36931.70 | 64601.00 | 65368.60 | 179.78 | 42963.30 | 43754.30 | 8.88 | 41777.00 | 42061.20 | 262.41 |
| | $\alpha$=1.5 | 54813.00 | 81707.90 | 82506.30 | 347.97 | 59530.50 | 60180.70 | 17.88 | 58415.10 | 58816.10 | 262.63 |
| Type5 | $\alpha$=0.5 | 18501.50 | 32603.00 | 33034.20 | 90.11 | 21434.90 | 21478.70 | 6.13 | 18794.40 | 18867.40 | 118.15 |
| | $\alpha$=1 | 24508.10 | 44201.30 | 44705.60 | 159.29 | 28729.00 | 29114.00 | 9.59 | 27641.70 | 27837.90 | 110.85 |
| | $\alpha$=1.5 | 36516.30 | 55650.00 | 56238.50 | 130.22 | 39033.10 | 39597.60 | 8.12 | 38730.40 | 39140.00 | 110.52 |
| | Average | 31343.727 | 51094.13 | 51680.91 | 178.59 | 35133.57 | 35533.14 | 10.01 | **33655.25** | **33882.07** | 172.64 |
| $n = 1000$ | | LB | UB | Av | $T_{GA}$ | UB | Av | $T_{RSBA}$ | UB | Av | $T_{LA-BM}$ |
| Type1 | $\alpha$=0.5 | 36257.60 | 61322.00 | 61809.20 | 425.55 | 38297.10 | 38636.30 | 19.64 | 36496.10 | 36538.00 | 461.93 |
| | $\alpha$=1 | 48217.50 | 84189.30 | 84932.50 | 426.71 | 56590.70 | 57377.30 | 19.86 | 54988.70 | 55319.90 | 460.97 |
| | $\alpha$=1.5 | 72630.30 | 108094.00 | 108963.00 | 423.89 | 79063.80 | 79949.90 | 19.90 | 78165.90 | 78705.30 | 461.44 |
| Type2 | $\alpha$=0.5 | 37150.80 | 73880.30 | 74539.40 | 698.76 | 49670.40 | 49924.50 | 28.61 | 41193.80 | 41324.90 | 664.22 |
| | $\alpha$=1 | 61249.70 | 103502.30 | 104327.50 | 690.86 | 71958.80 | 73364.70 | 28.44 | 69470.20 | 69751.60 | 659.49 |
| | $\alpha$=1.5 | 91304.70 | 132736.80 | 133681.50 | 691.01 | 99721.30 | 101230.50 | 28.36 | 98365.60 | 98756.60 | 657.18 |
| Type3 | $\alpha$=0.5 | 48673.70 | 81141.80 | 81726.00 | 697.23 | 50405.50 | 50853.30 | 27.20 | 48901.70 | 48972.10 | 687.31 |
| | $\alpha$=1 | 60644.10 | 110423.60 | 111298.80 | 703.04 | 71806.20 | 73023.50 | 27.73 | 70492.70 | 70889.30 | 712.35 |
| | $\alpha$=1.5 | 91361.40 | 140367.80 | 141512.60 | 709.17 | 101323.80 | 102708.10 | 28.02 | 99826.20 | 100346.50 | 712.21 |
| Type4 | $\alpha$=0.5 | 49041.30 | 94507.80 | 95303.00 | 1040.77 | 61509.00 | 61748.20 | 37.39 | 52437.10 | 52780.30 | 947.46 |
| | $\alpha$=1 | 73385.70 | 130331.40 | 131307.10 | 1038.16 | 88197.70 | 89749.10 | 37.40 | 84468.00 | 84845.10 | 984.56 |
| | $\alpha$=1.5 | 109891.60 | 165420.30 | 167058.30 | 1047.41 | 120859.70 | 122430.70 | 37.58 | 119926.00 | 120314.50 | 1068.74 |
| Type5 | $\alpha$=0.5 | 36624.50 | 65940.80 | 66546.40 | 445.58 | 42858.20 | 43007.30 | 19.51 | 37804.90 | 37970.20 | 425.95 |
| | $\alpha$=1 | 48729.40 | 89429.70 | 90141.50 | 452.29 | 58294.70 | 59218.30 | 19.45 | 56715.40 | 57076.40 | 458.93 |
| | $\alpha$=1.5 | 73181.70 | 113430.10 | 114302.60 | 452.35 | 80018.10 | 81205.10 | 19.62 | 79708.60 | 80168.20 | 459.07 |
| | Average | 62556.27 | 103647.87 | 104496.63 | 662.85 | 71371.67 | 72295.12 | 26.58 | **68597.39** | **68917.26** | 654.79 |

Table 8: Performance of LA-BM vs GA and RSBA on instances of Set 2: large-scale instances.

1. The first group with $n = 100$: LA-BM's best global average bound (last line of bloc 1) is equal to 6389.79 while those provided by both GA and RSBA are equal to 9635.93 and 6712.11. In this case, on the one hand, the gap between both LA-BM and GA (resp. RSBA) is closest to 3246.14 (resp. 322,33). On the other hand, EA(LA-BM) is equal to 1.01 (when comparing its result to that achieved by LB). Unfortunately, LA-BM's average runtime remains higher for instances with $n = 100$ (tasks).

2. The second group with $n = 500$: LA-BM's best global average bound (last line of bloc 2) is equal to 33655.25 while those achieved by both GA and RSA are equal to 51094.13 and 35133.57. Moreover, on the one hand, one can observe that the gap increases, in this case: it is close to 17438.88 when compared to GA's result and to -1478.32 for RSBA. On the other hand, LA-BM consumes the smallest runtime when compared to GA's average runtime. We note also that LA-BM's EA is equal to 1.07 while GA's (resp. RSBA's) experimental approximation ratio is equal to 1.67 (resp. 1.13).

3. The third group with $n = 1000$: the best global average bound (last line of bloc 3) of LA-BM is equal to 68597.39, which is better (smallest) than those achieved by both GA (103647.87) and RSBA (71371.67). In this case, LA-BM realizes a gap closest to 35050.47 when compared to GA's results and closest to 2774.27 when compared to RSBA's result. We note that LA-BM's experimental approximation ratio (EA(LA-BM)) is equal to 1.09 whereas EA(GA) is equal to 1.70 and EA(RSBA) is equal to 1.15. Finally, LA-BM average runtime remains smallest than that required for GA for matching the results displayed in the third block (column 12, of Table 8).

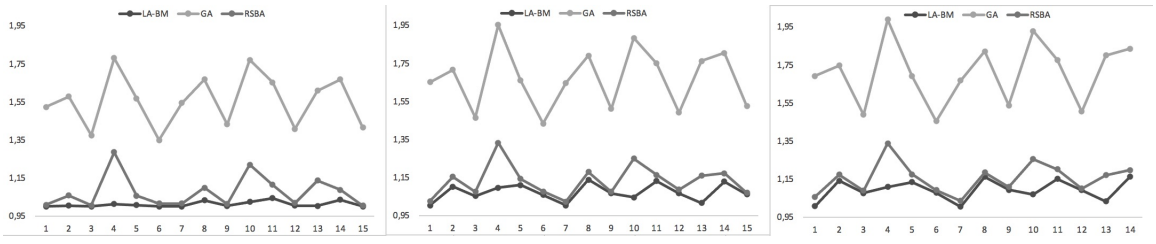


Figure 6: Variation of the experimental approximation ratios for the three tested algorithms GA, RSBA and LA-BM: (i) on the left-side of the figure for $n = 100$, (ii) on the middle of the figure for $n = 500$ and, (iii) on the right-side for $n = 1000$.

Figure 6 shows the variation of the experimental approximation ratios achieved by the

three tested methods: GA, RSBA and LA-BM. As we can show from that figure, LA-BM's the experimental approximation ratio is often better than those reached by both GA and RSBM. Indeed, it varies from 1 to 1.04 for $n = 100$, from 1 to 1.14 for $n = 500$ and, from 1 to 1.16 for $n = 1000$. While for GA (resp. RSBA) it varies from 1.35 to 1.78 (resp. from 1 to 1.29) for $n = 100$, from 1.43 to 1.95 (resp. from 1.02 to 1.33) for $n = 500$ and, from 1.45 to 1.99 (resp. from 1.04 to 1.34) for $n = 1000$.

# 6 Conclusion

In this paper, the problem of scheduling tasks on two dedicated processors was solved with an iterative search using a look-ahead strategy combined with path relinking. The proposed method combines four main features. First, a starting solution was built by tailoring a constructive greedy procedure following the knapsack problem rule. Second, an intensification search was introduced in order to visit a series of local solutions, where a standard tabu list was also incorporated for avoiding cycling solutions. Third, the drop and rebuild operator was added as a diversification strategy for searching unvisited subspaces. Fourth and last, the method was augmented by introducing the path relinking combined with a look-ahead strategy that applies a special beam search. Finally, the performance of the proposed method was evaluated on a set of benchmark instances containing small, medium, and large-scale instances, where its provided results were compared to those provided by more recent methods of the literature and to the tight lower bound tailored for the problem. The experimental part showed that the proposed look-ahead method performed better than all existing methods by providing tight experimental approximation ratios on the considered instances.


# Acknowledgements.

The authors thank the anonymous referees for their helpful comments and suggestions which contributed to the improvement of the presentation and the contents of this paper.


# References


[1] Aïder, M., Baatout, F.Z., & Hifi, M. (2020). A reactive search-based algorithm for scheduling multiprocessor tasks on two dedicated processors. In: *Proceedings*

*of the IEEE, Federated Conference on Computer Science and Information Systems*, M. Ganzha, L. Maciaszek, M. Paprzycki (eds). ACSIS, 21, 257-261, DOI: 10.15439/2020F134.

[2] Al-douri, T., Hifi, M., & Zissimopoulos, V. (2019). An iterative algorithm for the max-min knapsack problem with multiple scenarios. *Operational Research*, https://doi.org/10.1007/s12351-019-00463-7.

[3] Bianco, L., Blazewicz, J., Dell'Olmo, P., & Drozdowski, M. (1997). Preemptive multiprocessor task scheduling with release times and time windows, *Annals of Operations Research*, 70(1), 43-55, https://doi.org/10.1023/A:1018994726051.

[4] Blazewicz, J., Dell'Olmo, P., Drozdowski, M., & Speranza, M.G. (1992). Scheduling multiprocessor tasks on three dedicated processors. *Information Processing Letters*, 41, 275-280, https://doi.org/10.1016/0020-0190(92)90172-R.

[5] Blazewicz, J., Dell'Olmo, P., & Drozdowski (2002). Scheduling Multiprocessor Tasks On Two Parallel Processors. *RAIRO Operations Research*, 36, 37-51, DOI: 10.1051/ro:2002004.

[6] Brucker, P. (2007). Scheduling algorithms. Springer, ISBN 978-3-540-20524-1 4th ed. Springer Berlin Heidelberg New York.

[7] Buffet, O., Cucu, L., Idoumghar, L., & Schott, R. (2010). Tabu search type algorithms for the multiprocessor scheduling problem. In: *Proceedings of the 10th International Conference on Artificial Intelligence and Applications*, DOI: 10.2316/P.2010.674-070.

[8] Bukchin, Y., Raviv, T., & Zaides, I. (2000) The consecutive multiprocessor job scheduling problem, European Journal of Operational Research, 284(2), 427-438, https://doi.org/10.1016/j.ejor.2019.12.043.

[9] da Silva, E.C., & Gabriel, P.H.R. (2020). A comprehensive review of evolutionary algorithms for multiprocessor DAG scheduling. *Computation*, 8(2), 26, doi:10.3390/computation8020026.

[10] Drozdowski, M., (1996). Scheduling multiprocessor tasks-an overview. European Journal of Operational Research, 94(2), 215-230, https://doi.org/10.1016/0377-2217(96)00123-3.

[11] Graham, R.L., Lower, E.L, Lenstra, J.K., & Rinnoy, A.H.G. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling Theory' A Survey. *Annals of Discrete Mathematics*, 5, 287-326, https://doi.org/10.1016/S0167-5060(08)70356-X.

[12] Hifi M., & Michrafy M. (2006). A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society.* 57(6), 718-726, https://doi.org/10.1057/palgrave.jors.2602046.

[13] Hoogeveen, J.A., van de Velde, S.L., & Veltman, B. (1994). Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, 55, 259-272, https://doi.org/10.1016/0166-218X(94)90012-4.

[14] Kacem, A., & Dammak, A. (2014). A genetic algorithm to minimize the makespan on two dedicated processors. In: *Proceedings of the International Conference in Control, Decision and Information Technologies (CoDIT)*, pp. 400-404, https://doi.org/10.1109/CoDIT.2014.6996927.

[15] Kononova, A., Kononovaa, P. & Gordeev, A. (2020). Branch-and-bound approach for optima localization in scheduling multiprocessor jobs. *International Transactions in Operational Research*, 27, 381-393, DOI:10.1111/itor.12503.

[16] Laguna, M., Marti, R. (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11, 44-52, https://doi.org/10.1287/ijoc.11.1.44.

[17] Lei, D., & Cai, J. (2020). Multi-population meta-heuristics for production scheduling: a survey. *Swarm and Evolutionary Computation* (to appear), https://doi.org/10.1016/j.swevo.2020.100739.

[18] Manaa, A., & Chu, C. (2010). Scheduling multiprocessor tasks to minimise the makespan on two dedicated processors. *European Journal Industrial Engineering*, 4(3), 265-279, https://dx.doi.org/10.1504/EJIE.2010.033331.

[19] Priya, A., & Sahana, S.K. (2019). A survey on multiprocessor scheduling using evolutionary technique. In: *Nanoelectronics, Circuits and Communication Systems. Lecture Notes in Electrical Engineering*, Nath V., Mandal J. (eds), vol 511, pp. 149-160, Springer, Singapore, https://doi.org/10.1007/978-981-13-0776-8_14.

[20] Resende, MGC., Marti, R., Gallegoc, M., & Duarte, A. (2010). GRASP and path relinking for the max-min diversity problem. *Computers and Operations Research* 37(3), 498-508, https://doi.org/10.1016/j.cor.2008.05.011.

[21] Song, Y., Xing, L., Wang, M., Yi, Y., Xiang, W., & Zhang, Z. (2020). A knowledge-based evolutionary algorithm for relay satellite system mission scheduling problem. *Computers and Industrial Engineering*, 150, doi.org/10.1016/j.cie.2020.106830.

[22] Thesen. A. (1998). Design and evaluation of tabu search algorithms for multiprocessor scheduling. *Journal of Heuristics*, 4, 141-160, https://doi.org/10.1023/A:1009625629722.

[23] Zhang, X., Li, X. & Wang, J., (2016). Local search algorithm with path relinking for single batch-processing machine scheduling problem. *The Natural Computing Applications Forum*, 28, 313-326, DOI 10.1007/s00521-016-2339-z.

[24] Zhao, F., Zhang, L., Cao, J., & Tang, J. (2021). A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Computers and Industrial Engineering*, 153, doi.org/10.1016/j.cie.2020.107082.