Proceedings of the Federated Conference on Computer Science and Information Systems pp. 257–261
DOI: 10.15439/2020F134
ISSN 2300-5963 ACSIS, Vol. 21

A Reactive Search-Based Algorithm for Scheduling Multiprocessor Tasks on Two Dedicated Processors

Méziane Aïder
LaROMaD, USTHB
BP 32 El Alia, 16111 Alger, Algérie
Email: m-aider@usthb.dz

Fatma Zohra Baatout
LaROMaD, USTHB
BP 32 El Alia, 16111 Alger, Algérie
Email: fbaatout@usthb.dz

Mhand Hifi*
EPROAD, UPJV
7, rue du Moulin Neuf, 80000 Amiens, France
Email: hifi@u-picardie.fr

***Abstract*—In this paper, we propose a reactive search-based algorithm for solving the problem of scheduling multiprocessor tasks on two dedicated processors. An instance of the problem is characterized by 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 determine the scheduling of all tasks minimizing the execution of the last assigned task. The proposed reactive search starts with a starting greedy solution. Next, a series of local operators combined with a tabu list are introduced in order to intensify the search process. The method is also reinforced with a drop and rebuild operator that is applied for diversifying the search process. Finally, the performance of the proposed method is evaluated on a set of benchmark instances, where its provided results are compared to those achieved by a recent method available in the literature. Encouraging results have been reached.**

I. Introduction

THE problem of Scheduling multiprocessor Tasks on Two dedicated Processors (noted ST2P) is an NP-hard combinatorial optimization problem (cf. Hoogeveen *et al.* [8]), where its aims is to assign available tasks to two different processors. Generally, for the scheduling problems, the measures of performance are often categorized into three main groups: criteria based on completion time, criteria based on due dates, and those based on inventory cost and use. The studied problem is a special case of the scheduling problems family, where the set of tasks is divided into three groups, where the first group contains the tasks that need to be performed on the first processor, the second group contains those executed on the second processor while the third group contains the tasks that must be performed simultaneously on both processors. For such problem, on the one hand, several objective functions can be considered, like (i) minimizing the makespan, (ii) to minimize the summation of the delays of all tasks, (iii) to minimize both delays and makespan, etc. On the other hand, several versions of the scheduling problem can be accessed (i) on the number of available processors, (ii) how tasks are assigned on certain processors, etc.

Herein, we study the multiprocessor tasks scheduling on two dedicated processors problem. Its goal is to minimize the completion time of the last assigned/executed task (makespan). Such a version of the problem can be encountered in several real-world applications, like production and data transfer (cf. Manaa and Chu [10]). An instance of ST2P problem may be defined as follows: let N denote the set containing n tasks to scheduling 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 the j-th task while C_{max} denotes the makespan of the schedule to minimize. As described in Graham *et al.* [5], such a problem is defined as $P2|f_ix_j, r_j|C_{max}$, where:

- $P2$: represents two processors on which all tasks must be executed.
- f_ix_j: means that task j is affected to both processors.
- r_j: denotes the release date of the j-th task.
- p_j: is the processing time of the j-th task 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 II reviews some related works tackling scheduling problems. A nice decomposition of ST2P, proposed by Manaa and Chu [10], providing a tight lower bound is given in Section III. Section IV describes the proposed reactive search-based algorithm for approximately solving ST2P. A starting solution, using a knapsack greedy rule, is described in Section IV-A. The intensification operators, combined with a tabu list, are discussed in Section IV-B. The diversification strategy, using the drop and rebuild operator, is discussed in Section IV-C. Section V exposes the experimental part, where the performance of the proposed method is evaluated on a set of benchmark instances. The provided results are compared to those achieved by a recent algorithm of the literature and to the results achieved by Manaa and Chu's lower bound. Finally, Section VI summarizes the content of the paper.

II. Related Works

The scheduling problems family contains a huge number of problem types as underlined in Brucker [3]. Generally, the performance measures for scheduling problems are often categorized into three main groups of criteria: those based on completion time, those based on due dates, and those based on inventory cost and utilization. Due to the complexity of the studied problem, there are few available papers tackling it in the literature.

Bianco *et al.* [1] tackled the problem of scheduling tasks on two dedicated processors with preemptive constraints (noted

*Corresponding Author (M. Hifi)

$P2|f_i x_j, r_j, pmtn|C_{max}$), where the task can be interrupted and completed later. An exact algorithm has been designed that is based on two steps polynomial time complexity.

Manaa and Chu [10] proposed an exact algorithm for solving the problem studied in this paper. The method is based upon a branch and bound 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 [9] tailored an effective genetic algorithm for approximately solving the same problem. 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's [10] generator and the experimental evidence showed that the method was able to achieve bounds closest to those provided by Manaa and Chu's [10] tight lower bounds.

Thesen [11] designed a tabu search for tackling general multiprocessor scheduling problems. The method combines tabu strategy and local search operator. Several strategies have been considered, like random blocking related to the size of the tabu list, frequency-based penalties for diversifying the search, and the hashing operator for stocking high solutions. The experimental part showed that some combinations have better behavior than others.

Blazewicz *et al.* [2] tackled the problem of scheduling multiprocessor tasks on three dedicated processors. The authors studied the complexity analysis, where different cases were considered for which they proposed optimal solutions in polynomial time complexity.

Buffet *et al.* [4] developed two tabu search for solving the scheduling problem with m processors. A standard tabu search was followed, where a starting solution is built by respecting a legal schedule, the intensification strategy that checks possible permutations between tasks, the diversification strategy using a local search for exploring unvisited subspaces.

III. A Lower Bound for ST2P

Manaa and Chu [10] proposed a nice lower bound for ST2P that is based on relaxing the original problem into two subproblems to solve. They also proved that bound provides an optimal solution for the preemptive case of the problem, i.e., $P2|f_i x_j, r_j, pmtn|C_{max}$. The calculation of such a bound is explained in what follows.
Let $N = \{1, \ldots, n\}$ be the set of tasks and P_1 and P_2 two processors 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 the j-th task while C_{max} denotes the makespan of the schedule to minimize. A task $j \in N$ is called a $P_1 - task$ (resp. $P_2 - task$) if it is affected to the processor P_1 (resp. P_2) while it is called $P_{12} - task$ whenever the j-th task requires simultaneously both processors P_1 and P_2; that is a bi-processor 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. $P_{12}^2 - Tasks$) are 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^{opt} for the first subproblem with processor P_1 and C_2^{opt} for the second one with processor P_2. Hence, ST2P's lower bound corresponds to

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

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

IV. A Reactive Search for ST2P

In this section, we expose the cooperative method for scheduling tasks on two dedicated processors problem. The main principle of the reactive search can be summarized as follows:

1) Starting the search process by an initial solution using a basic knapsack's greedy procedure (cf. Section IV-A).
2) Building an improved solution using a series of permutations (cf. Section IV-B).
3) Perturbing the search process and re-constructing a new current solution with a basic greedy procedure according to the new order (cf. Section IV-C).
4) Steps (1)-(3) are repeated until a satisfactory solution is reached.

A. A Constructive Procedure

Generating a solution is equivalent to generate a sequence of positions of the tasks on the processors. Herein, the starting solution can be provided by using a standard scheduling's greedy procedure that can be adapted for ST2P. The procedure can be viewed as a Constructive Procedure (noted CP) that applies two main steps: (i) reordering the objects (tasks) according to given criteria and (ii) selecting step by step a non-affected item (task) and assigning it to a knapsack (processor). The second step is repeated until positioning all the items (tasks) on their corresponding knapsack (processor).

Indeed, let r_j be the release date of the j-th task and p_j its processing time. Then,

1) Compute all ratios representing the processing time per release date, i.e., $\frac{p_j}{r_j},\ j \in N$.
2) Reorder all ratios in non-increasing order; that is $\frac{p_1}{r_1} \geq \ldots \geq \frac{p_j}{r_j} \geq \ldots \geq \frac{p_n}{r_n}$.

Finally, by applying the principle of the greedy knapsack procedure to each task, according to the aforementioned order,

a starting solution is provided for ST2P; that forms a sequence of tasks assigned to either the first processor, or the second processor, or both processors.

B. Intensification Search

Determining an improved solution (with a new sequence) is equivalent to solve a reduced problem by fixing some tasks. Making some moves between tasks is equivalent to fix some of them and to reassign the rest of the tasks on their corresponding processors(s).

1) A 2-opt Operator: A 2-opt operator is a simple local search/improvement procedure, which is even based upon simple local modifications of the current solution. Given a (current) feasible solution, the operator repeatedly makes some moves/swaps/shakes as long as the quality of the induced solution is improved. Whenever the improvement stagnates around the same objective value, we say that the 2-opt operator reaches its limits; that is a situation where the method is trapped into a local optimum. Herein, the 2-opt operator consists of swapping two randomly chosen positions of the sequence. The series related to these swaps induces the current neighborhood around the solution at hand.

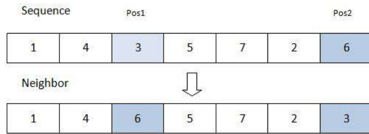

Fig. 1. The 2-opt operator

Figure 1 illustrates the swapping operator used at each step of the intensification search. One can observe that making a simple swapping between two tasks may provide either a feasible solution or (i) an unfeasible one. In the case of the unfeasible solution, we propose a repairing operator, which can be viewed as a two-step procedure. Let i and j denote the two positioned tasks (after a swap), such that i is positioned before j. Then the following two-steps procedure is applied to the provided configuration.

The first-step. The first step of the repairing operator can be applied as follows: (**i**) According to the position of the i-th task, move all tasks from the left to the right till removing all overlapping; (**ii**) According to the position of the j-th task (with its new position), move all tasks from the left to the right till removing all overlapping.

The second-step. Observe that swapping two tasks induces a new sequence and so, a simple knapsack greedy procedure CP can be applied to that order.
Hence, by applying both steps for the current solution, a series of solutions are built; that are the solutions forming the current 2-opt neighborhood.

2) A 3-opt Operator: In this section, we propose a local search based upon the 3-opt operator. As observed above (Section IV-B1), a current solution may be locally improved by using a simple 2-opt operator that is based on small moves. Herein, we propose to introduce a neighbor operator with higher freedom, which can mix two consecutive solutions around the current solution.

The idea is to repeat a series of small moves around the current solution. After some iterations, apply another search operator with higher moves and continue searching with small moves. Such a search is repeated until satisfying a predefined stopping criteria. One step of the higher move-based operator can be described as follows (let $\underline{S}$ be the current solution): (**i**) Select two random tasks from $\underline{S}$, permute both tasks for forming a new configuration S'; (**ii**) Select two random tasks from S' (different from the already swapped tasks), permute both tasks for forming a new configuration S''; (**iii**) Call the 2-opt operator on S'' for providing the best solution (noted $\underline{S}'$) around the solution at hand $\underline{S}$.

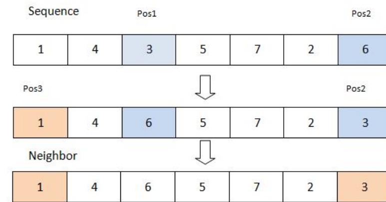

Fig. 2. The 3-opt operator

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

3) Using a Tabu List: Generally, both 2-opt and 3-opt operators try to build a series of solutions belonging to a series of subspaces. Because a new solution built can be provided by exchanging the positions of two tasks, one can observe that repeating the same process may lead toward the same local optimum and so, the method can be trapped into that optimum. Among the techniques that can be introduced to avoid cycling towards the same solutions, the tabu search remains one of the simplest strategies that can be introduced whenever the studied problem belongs to the combinatorial optimization problems family. Because the method uses swaps between tasks, it is interesting to reinforce the search process by adding a tabu list. It contains a list of temporarily inverse-moves that avoids returning to the solutions already visited.

C. Diversification Search

The intensification strategy tries to find a series of feasible solutions to the problem, which are often considered as local optima. The objective of the building procedure is to provide a series of neighborhoods, issuing from the solution at hand, which might contain better solutions. Despite some improvements that can be realized, and because of the number of achievable solutions with the same objective value, it is interesting to provide a manner capable to drive the search process through other unvisited subspaces.

Herein, we propose a diversification search that consists of removing a subset of tasks from the current sequence (i.e. a feasible solution of the problem). The removing strategy tries to diversify the search process by degrading the quality of the solution at hand with the aim of avoiding stagnating in a local optimum. Then, a partial solution is obtained and it is completed using the constructive procedure as a tool for refining the quality of the partial solution, according to the new

order associated with the remaining tasks. Such a strategy was already used with success for solving variants of the knapsack type problems (cf., Hifi [6] and Hifi and Michrafy [7]).

Herein, the diversification strategy can be applied by using the Drop and Rebuild Operator (DRO) that is described as follows. According to the current solution $\underline{S}$, DRO tries to reduce the problem, by randomly fixing a subset of tasks of $\underline{S}$, as follows. **Step 1:** From the solution $\underline{S}$, drop $\beta\%$ of the tasks belonging to that sequence; **Step 2:** Solve the reduced instance by calling the constructive procedure CP (cf., Section IV-A) and **Step 3:** Complete the current solution by calling CP, with the already removed tasks.

Algorithm 1 A Reactive Search-Based Algorithm (RSBA)

Input. An instance of SP2P.
Output. A feasible solution $S^\star$ with its objective value $C^\star_{max}$.

1: Set $S^\star = \emptyset$ and $C^\star_{max} = +\infty$.
2: Call CP for solving the original problem providing the solution $\underline{S}$ with objective value C_{max}.
3: **repeat**
4: **while** (the stopping criterion is not performed) **do**
5: **if** ($C_{max} < C^\star_{max}$) **then**
6: set $S^\star = \underline{S}$ and $C^\star_{max} = C_{max}$.
7: **end if**
8: **while** (2-opt local iterations is not matched) **do**
9: Call 2-opt using $\underline{S}$'s neighborhood and let $\underline{S}'$ be the neighbor solution with the best objective value C'_{max}.
10: **if** ($C'_{max} < C^\star_{max}$) **then**
11: set $S^\star = \underline{S}$ and $C^\star_{max} = C'_{max}$.
12: **end if**
13: Update the local iterations and set $\underline{S} = \underline{S}'$.
14: **end while**
15: (i) Call 3-opt using $\underline{S}$'s neighborhood and let $\underline{S}'$ be the neighbor solution with the best objective value C'_{max}.
(ii) Set $\underline{S} = \underline{S}$ and $C_{max} = C'_{max}$.
16: **end while**
17: (i) Apply DRO to the best current solution $S^\star$ and let $\underline{S}$ be the solution reached.
(ii) Reinitialize the 2-opt local iterations.
18: **until** (the global criterion is performed).
19: **return** $S^\star$ with its objective value $C^\star_{max}$.

D. An Overview of the Reactive Search

Algorithm 1 describes the main steps of the Reactive Search-Based Algorithm (denoted RSBA). The input of RSBA is an instance of SP2P and its output is an (near)optimal solution $S^\star$ with its objective value $C^\star_{max}$. The algorithm begins by generating a starting solution (line 2) provided by calling the constructive procedure CP. RSBA is composed of three loops, a global loop, and two internal loops. The global loop `repeat` from line 3 to line 18 that is applied for generating a series of current solutions, which are enhanced by using both intensification and diversification phases. Its stopping condition is defined according to the number of iterations based on the size of the instance. The first internal loop `repeat` from line 8 to line 14 serves to intensify the search by using the 2-opt operator while the second internal loop (from line 4 to line 16) is used for calling the 3-opt operator. The diversification procedure is considered whenever both internal loops stagnate on a local optimum (points (i) and (ii) of line 17). Both internal loops are embedded into the global loop `repeat` which serves to repeat the enhancement and the scattering on a new solution generated by the drop and rebuild operator DBO. The global loop is iterated until either the runtime limit or the number of global iterations is performed. Finally (line 19), RSBA returns $S^\star$, the best solution found so far with its objective value $C^\star_{max}$.

V. Computational Results

The solution method proposed in this study, the Reactive Search-Based Algorithm (noted RSBA), is evaluated on two sets instances, where each set is composed of five groups such that each group is related to the type of instances considered (as suggested in Manaa and Chu [10]). The proposed method was coded in C++ language and run on an Intel Pentium Core i7-8550U 1.99 GHz and 16 Gb of RAM. In order to evaluate the behavior of the proposed RSBA, we also compared its provided results to those achieved by both the Genetic Algorithm (noted GA) proposed in Kacem and Dammak [9]([1]) and the tight Lower Bound (noted LB) proposed in Manaa and Chu [10] (as used in Kacem and Dammak [9]).

TABLE I
Characteristics of the instances

Type of task	T1	T2	T3	T4	T5
$n1$	n	n	n	n	$[n/2]$
$n2$	$[n/2]$	n	$[n/2]$	n	$[n/2]$
$n12$	$[n/2]$	$[n/2]$	n	n	n

TABLE II
Performance of both RSBA and GA on instances of Set 1: small and medium instances

	Tasks		GA			RSBA		
	$n = 10$	LB	UB	Av. UB	T_{GA}	UB	Av. UB	T_R
T1	$\alpha = 0.5$	400.90	441.30	467.80	0.068	**407.20**	407.20	0.2064
	$\alpha = 1$	478.70	540.90	575.50	0.0654	**496.40**	496.40	0.1961
	$\alpha = 1.5$	789.30	845.50	907.20	0.0734	**797.90**	797.90	0.225
T2	$\alpha = 0.5$	402.40	519.10	556.00	0.0935	**476.60**	476.60	0.2606
	$\alpha = 1$	651.00	738.10	810.30	0.1025	**648.80**	648.80	0.1975
	$\alpha = 1.5$	914.80	1002.70	1067.20	0.0929	**925.90**	925.90	0.1824
T3	$\alpha = 0.5$	494.90	594.70	640.20	0.1042	**528.50**	528.50	0.1924
	$\alpha = 1$	664.00	841.30	895.20	0.0943	**696.60**	696.60	0.2354
	$\alpha = 1.5$	924.80	1062.80	1125.40	0.079	**936.10**	936.10	0.1512
T4	$\alpha = 0.5$	547.60	690.90	751.30	0.1175	**582.10**	582.10	0.1808
	$\alpha = 1$	732.20	959.10	1025.80	0.1149	**781.70**	781.70	0.2287
	$\alpha = 1.5$	674.50	767.10	811.60	0.0931	**681.20**	681.20	0.232295
T5	$\alpha = 0.5$	373.00	444.80	475.30	0.0727	**397.00**	397.00	0.2149
	$\alpha = 1$	545.00	655.40	709.40	0.0627	**560.80**	560.80	0.1938
	$\alpha = 1.5$	595.50	667.70	712.30	0.061	**601.60**	601.60	0.1736
	Average	612.57	718.09	768.70	**0.086**	**634.56**	**634.56**	0.205
	$n = 20$	LB	UB	Av. UB	T_{GA}	UB	Av. UB	T_{RSBA}
T1	$\alpha = 0.5$	354.50	405.80	427.70	0.16234	**353.20**	353.20	0.280971
	$\alpha = 1$	411.60	523.40	567.10	0.164691	**418.40**	418.50	0.191233
	$\alpha = 1.5$	536.30	656.10	691.30	0.163243	**554.10**	554.30	0.196438
T2	$\alpha = 0.5$	318.60	466.00	491.50	0.258109	**387.10**	387.10	0.232317
	$\alpha = 1$	471.10	630.60	666.40	0.256941	**491.00**	491.30	0.226608
	$\alpha = 1.5$	703.50	838.40	882.30	0.259082	**708.90**	708.90	0.234278
T3	$\alpha = 0.5$	392.80	519.90	541.50	0.245839	**396.00**	396.00	0.218839
	$\alpha = 1$	491.80	690.50	722.40	0.246592	**507.10**	507.80	0.219218
	$\alpha = 1.5$	670.70	842.50	884.50	0.245212	**680.20**	680.60	0.224639
T4	$\alpha = 0.5$	443.40	611.40	640.00	0.356819	**504.90**	504.90	0.263979
	$\alpha = 1$	568.50	810.20	853.50	0.357753	**596.00**	597.00	0.252089
	$\alpha = 1.5$	843.30	1059.20	1107.30	0.357673	**854.20**	854.20	0.265056
T5	$\alpha = 0.5$	287.10	391.30	413.10	0.156887	**319.20**	319.20	0.187833
	$\alpha = 1$	395.10	549.30	582.50	0.163567	**422.60**	422.60	0.182792
	$\alpha = 1.5$	643.00	755.80	793.60	0.158044	**651.00**	651.00	0.1956
	Average	502.09	650.03	684.31	0.237	**522.93**	**523.11**	**0.225**

The generator suggested by Manaa and Chu [10] considered five types of instances, related to the number of tasks n to use and those affected to both P_1 and P_2 and the bi-processor tasks affected to both P_1 and P_2 simultaneously: (i) the number of tasks $n = 10$ for small instances, $n = 20$ for medium-sized ones and $n = 100$ for large-scale ones, where thirty instances are considered for each value, (ii) the number n_1 (resp. n_2 and n_{12}) denotes the number of tasks assigned to the processor

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

P_1 (resp. P_2 and P_{12}) and generated according to the values illustrated in Table I, where $[x]$ denotes the integral value of x, (iii) the processing time p_j related to the duration of the j-th task is randomly generated in $\{1, \ldots, 50\}$ and (iv) the release date r_j of task j, is randomly generated in the interval $\{1, \ldots, k\}$, where k is setting equal to $\alpha \times \frac{(s_{12}+(s_1+s_2)}{2}$ such that $\alpha \in \{0.5; 1; 1.5\}$ (the density of the instance) and s_1 (resp. s_2 and s_{12}) denotes the total duration related of the tasks belonging to P_1 (resp. P_2 and P_{12}).

TABLE III
PERFORMANCE OF BOTH RSBA AND GA ON INSTANCES OF SET 2: $n = 100$ (LARGE-SCALE INSTANCES)

	Tasks n=100	LB	GA UB	GA Av. UB	T_{GA}	RSBA UB	RSBA Av. UB	T_R
T1	$\alpha = 0.5$	3708.6	5 649.8	5 839,7	4.035	**3742.4**	3 748.4	1.161
	$\alpha = 1$	4 885.8	7 711.7	7 935.4	4.014	**5 167.3**	5 261.3	1.136
	$\alpha = 1.5$	7 465.6	10 260.1	10 537.2	4.053	**7 508.4**	7 595.9	1.172
T2	$\alpha = 0.5$	3 793.1	6 753.6	6 940.8	6.502	**4 875.8**	4 877.1	1.532
	$\alpha = 1$	6 113.9	9 581.9	9 770.8	6.488	**6 463.1**	6 609.1	1.452
	$\alpha = 1.5$	9 374.6	12 641.5	12 958.6	**6.485**	9 518.9	9 621.8	1.453
T3	$\alpha = 0.5$	4 931.6	7 617.0	7 805.9	6.440	**5 012.1**	5 026.5	1.421
	$\alpha = 1$	6 181.4	10 312.3	10 570.1	6.410	**6 790.5**	6 896.2	1.399
	$\alpha = 1.5$	9 019.7	12 908.8	13 181.3	6.422	**9 134.5**	9 285.4	1.410
T4	$\alpha = 0.5$	4 981.6	8 821.4	9 032.2	10.245	**6 073.3**	6 079.3	1.877
	$\alpha = 1$	7 270.7	12 010.3	12 265.5	9.580	**8 098.7**	8 214.4	1.727
	$\alpha = 1.5$	10 918.8	15 372.1	15 769.1	9.658	**11 120.4**	11 259.6	1.739
T5	$\alpha = 0.5$	3 855.9	6 204.3	6 394.1	3.974	**4 383.8**	4 386.2	1.141
	$\alpha = 1$	4 951.8	8 257.7	8 476.0	3.960	**5 383.8**	5 468.5	1.084
	$\alpha = 1.5$	7 378.2	10 436.4	10 748.6	3.973	**7 408.7**	7 473.1	1.122
	Average	6 322.1	9 635.9	9 881.7	6.15	**6 712.11**	**6 786.85**	**1.388**

A. Behavior of RSBA vs GA on small and medium instances

First, in order to evaluate the performance of the proposed method RSBA, we compare its provided results to those of GA and to the lower bound LB of Manaa and Chu [10]. Table II shows LB, Kacem and Dammak's algorithm (GA) and those provided by RSBA. Columns 1 and 2 display the data information, column 3 reports LB of each instance, column 4 (resp. column 5 and column 6) tallies the GA's bound (resp. the average value and the average runtime over the ten trials) while column 7 (resp. column 8 and column 9) reports the best RSBA's bound (resp. the average values and the 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 obtained by the considered algorithm). According to Table II, for the small instances with $n = 10$, RSBA outperforms GA although when considering the average value (the solution values over the ten trials). Indeed, RSBA realizes an average global value of 634.56 while GA provides an average global value equals to 768.70. The Gap between both values is closest to 134 units even GA's average runtime remains smaller than that of RSBA. For the medium-sized instances with $n = 20$, the same phenomenon can be observed. Indeed, the global RSBA's best value (522.93) is better than that achieved by GA (650.03). For the achieving results, GA's global average runtime is slightly greater (0.237 sec) than that needed by RSBA (0.225 sec), for the medium instances.

B. Behavior of RSBA vs GA on large-scale instances: Set 2

Herein, RSBA's behavior is analyzed on the instances of Set 2 which contains thirty instances representing more largest benchmark instances. Its achieved results are also compared to those achieved by GA and Manaa and Chu's lower bound. Table III reports the bounds achieved by RSBA, GA and LB on the instances of `Set 2`. From the table, one can observe that RSBA remains competitive when comparing its results to those achieved by GA. RSBA's average best solution value is equal to 6712.11 while that of GA is equal to $9635, 93$, which achieves a significant Gap closest to 2924. The global RSBA's average solution values are also better than those matched by GA and the average RSBA's runtime, in this case, is smaller than that needed by GA, i.e., 1.388 sec versus 6.150 sec. The average RSBA's best solution value provides an experimental approximation ratio of 1.062 when compared to Manaa and Chu's lower bound LB while GA's reaches an approximation ratio equal to 1.524. The larger the instance, more the behavior of RSBA is interesting, which also consumes a smaller runtime for this type of instance.

VI. CONCLUSION

The problem of scheduling tasks on two dedicated processors is solved with a reactive search-based algorithm. The method combines three main features: a starting solution built by tailoring a constructive greedy procedure, an intensification search introduced in order to visit a series of local solutions and a diversification strategy using the drop and rebuild operator. Finally, the experimental part showed the effectiveness of the proposed method when compared to the best available method in the literature.

REFERENCES

[1] Bianco, L., Blazewicz, J., Dell'Olmo, P. and Drozdowski, M. (1997). 'Preemptive multiprocessor task scheduling with release times and time windows', Annals of Operations Research,Vol. 70, No. 1, pp.43-55, https://doi.org/10.1023/A:1018994726051.

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

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

[4] Buffet. O, Cucu. L, Idoumghar. L and Schott. R. (2010). 'Tabu Search Type Algorithms for the Multiprocessor Scheduling Problem'. Conference: Artificial Intelligence and Applications, https://hal.archives-ouvertes.fr/hal-00435241.

[5] 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.V5, p287-326, https://doi.org/10.1016/S0167-5060(08)70356-X.

[6] Hifi, M. (2014). An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem. Engineering Optimization. 46(8), 1109–1122, https://doi.org/10.1080/0305215X.2013.819096.

[7] Hifi M. and 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.

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

[9] Kacem, A., Dammak, A. (2014)'A genetic algorithm to minimize the makespan on two dedicated processors', In IEEE, International Conference on Control, Decision and Information Technologies (CoDIT), pp.400-404, 3-5 Nov. 2014 Metz, France, doi: 10.1109/CoDIT.2014.6996927.

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

[11] 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