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

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

Fatma Zohra Baout  
LaROMaD, USTHB  
BP 32 El Alia, 16111 Alger, Algérie  
Email: fbaout@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 to diversify 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 \( C_{\text{max}} \) denotes the makespan of the schedule to minimize. As described in Graham et al. [5], such a problem is defined as \( P_2^{\text{||C_{\text{max}}} \}} \), where:

- \( P_2 \): represents two processors on which all tasks must be executed.
- \( f_{ixj} \): 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.
- \( C_{\text{max}} \): 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
\[ P2|\sum(jr_j,pmtn)|C_{\text{max}} \]

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 with processor \( P_1 \) and \( P_2 \). Those provided by Manaa and Chu’s [10] tight lower bounds.

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 \( C_{\text{max}} \) 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 \( P_1 - Tasks \) (resp. \( P_2 - Tasks \)) are separately scheduled on each processor. Thus,

- \( P_1 - Tasks \) and \( P_12 - Tasks \) should be scheduled on processor \( P_1 \).
- \( P_2 - Tasks \) and \( P_22 - Tasks \) 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 \( C_{\text{opt}} \) for the first subproblem with processor \( P_1 \) and \( C_{\text{opt}}^1 \) for the second one with processor \( P_2 \). Hence, ST2P’s lower bound corresponds to

\[ \max(C_{\text{opt}}^1, C_{\text{opt}}^2) \]

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 \( \frac{p_j}{r_j} \geq \cdots \geq \frac{p_j}{r_j} \geq \cdots \geq \frac{p_j}{r_j} \).

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](image1)

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 $S$ be the current solution): (1) Select two random tasks from $S$; (ii) Select 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 $S'''$) around the solution at hand $S$.

![Fig. 2. The 3-opt operator](image2)

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 \( S \), DRO tries to reduce the problem, by randomly fixing a subset of tasks of \( S \) as follows. **Step 1:** From the solution \( S \), 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)

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

**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 \( S^* \) with its objective value \( C^*_{max} \). 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 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 \( S^* \), the best solution found so far with its objective value \( C^*_{max} \).

**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] and the tight Lower Bound (noted LB) proposed in Manaa and Chu [10] (as used in Kacem and Dammak [9]).

**TABLE I**

<table>
<thead>
<tr>
<th>Type of task</th>
<th>Characteristics of the instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S M A L L A N D M E D I U M I N S T A N C E S</td>
</tr>
</tbody>
</table>

**TABLE II**

**Performance of both RSBA and GA on instances of Set 1:**

<table>
<thead>
<tr>
<th>Tasks</th>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( \gamma )</th>
<th>( \delta )</th>
<th>( \epsilon )</th>
<th>( \zeta )</th>
<th>( \eta )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>( \beta )</td>
<td>( \gamma )</td>
<td>( \delta )</td>
<td>( \epsilon )</td>
<td>( \zeta )</td>
<td>( \eta )</td>
<td>( \theta )</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>( \beta )</td>
<td>( \gamma )</td>
<td>( \delta )</td>
<td>( \epsilon )</td>
<td>( \zeta )</td>
<td>( \eta )</td>
<td>( \theta )</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>( \beta )</td>
<td>( \gamma )</td>
<td>( \delta )</td>
<td>( \epsilon )</td>
<td>( \zeta )</td>
<td>( \eta )</td>
<td>( \theta )</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>( \beta )</td>
<td>( \gamma )</td>
<td>( \delta )</td>
<td>( \epsilon )</td>
<td>( \zeta )</td>
<td>( \eta )</td>
<td>( \theta )</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>( \beta )</td>
<td>( \gamma )</td>
<td>( \delta )</td>
<td>( \epsilon )</td>
<td>( \zeta )</td>
<td>( \eta )</td>
<td>( \theta )</td>
<td></td>
</tr>
</tbody>
</table>

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 \( n_1 \)) 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 $P_2^b$) 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 $\alpha \times \frac{(k+1)(k+2)}{2}$ such that $\alpha \in \{0.5; 1.1.5\}$ (the density of the instance) and $s_1$ (resp. $s_2$ and $s_2^b$) denotes the total duration related of the tasks belonging to $P_1$ (resp. $P_2$ and $P_2^b$).

### Table III

**Performance of both RSBA and GA on instances of Set 2:**

<table>
<thead>
<tr>
<th>$n = 100$ (large-scale instances)</th>
<th>$n = 50$</th>
<th>$n = 30$</th>
<th>$n = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tasks</strong></td>
<td><strong>LB</strong></td>
<td><strong>UB</strong></td>
<td><strong>Av</strong></td>
</tr>
<tr>
<td>$\alpha = 0.5$</td>
<td>5 050.3</td>
<td>5 185.2</td>
<td>5 078.9</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>5 130.3</td>
<td>5 255.2</td>
<td>5 203.1</td>
</tr>
<tr>
<td>$\alpha = 1.5$</td>
<td>5 113.3</td>
<td>5 242.2</td>
<td>5 211.1</td>
</tr>
<tr>
<td>$\alpha = 2$</td>
<td>5 030.3</td>
<td>5 165.2</td>
<td>5 093.1</td>
</tr>
<tr>
<td>$\alpha = 3$</td>
<td>5 010.3</td>
<td>5 145.2</td>
<td>5 073.1</td>
</tr>
<tr>
<td>$\alpha = 5$</td>
<td>5 000.3</td>
<td>5 135.2</td>
<td>5 083.1</td>
</tr>
<tr>
<td>$\alpha = 10$</td>
<td>5 000.3</td>
<td>5 135.2</td>
<td>5 083.1</td>
</tr>
</tbody>
</table>

**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


