DOI: 10.5902/1983465913305

MIP–BASED NEIGHBORHOOD SEARCH FOR THE UNRELATED PARALLEL MACHINE SCHEDULING PROBLEM WITH SEQUENCE AND MACHINE-DEPENDENT SETUP TIMES

Heurística baseada em programação matemática para o problema de programação de tarefas em máquinas paralelas não relacionadas com tempo de preparação dependente da sequência e da máquina.

Recebido 28-03-2014
Aceito 20-01-2015

Felipe Martins Muller[1]
Olinto Bassi Araújo[2]
Fernando Stefanello[3]
Marcelo Zanetti[4]

ABSTRACT

In this paper we study the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. We consider the objective of minimizing the maximum completion time of the latest job, usually referred to as makespan. We propose a new MIP-based heuristic combining atomic moves such as insertion, ejection and closure, in order to generate sequences of such atomic moves minimizing the makespan. This heuristic employs a commercial solver to search the neighborhood in a multi-start algorithm. Our approach performed well in computational experiments targeting two sets of benchmark instances previously used in the literature

Keywords: *MIP-based neighborhood, hybrid metaheuristics, unrelated parallel machine scheduling problem, makespan.*

[1] Possui graduação em Engenharia Elétrica pela Universidade Federal de Santa Maria – UFSM, mestrado e doutorado em Engenharia Elétrica pela Universidade Estadual de Campinas – UNICAMP. Atualmente é professor titular na Universidade Federal de Santa Maria – UFSM. Santa Maria, Rio Grande do Sul, Brasil. Email: felipe@inf.ufsm.br

[2] Possui graduação em Matemática pela Universidade Federal da Campanha – URCAMP, mestrado em Modelagem Matemática pela Universidade Regional do Noroeste do Estado do Rio Grande do Sul – UNIJUI, doutorado em Engenharia Elétrica pela Universidade Estadual de Campinas. Atualmente é professor no Colégio Técnico Industrial de Santa Maria – CTISM. Santa Maria, Rio Grande do Sul, Brasil. Email: olinto@ctism.ufsm.br

[3] Possui graduação em Matemática pela Universidade Federal de Santa Maria – UFSM. Atualmente é mestrando em Informática pela Universidade Federal de Santa Maria – UFSM. Santa Maria, Rio Grande do Sul, Brasil. Email: stefanello@inf.ufsm.br

[4] Possui graduação em Engenharia Elétrica pela Universidade Federal de Campinas – UNICAMP, mestrando em Ciência da Computação e doutorado em Físicas de Sistemas Complexos e Engenharia de Software pelo Instituto Federal de Tecnologia de Zurique. Atualmente é professor na Universidade Federal do Maranhão – UFMA. São Luís, Maranhão, Brasil. Email: mzanetti@ethz.ch.

RESUMO

Neste trabalho é estudado o problema de programação de tarefas em máquinas paralelas com tempo de preparação dependente da sequência e da máquina. Como objetivo é considerado a minimização da duração total da programação, usualmente referido como makespan. É proposta uma nova heuristica MIP que combina movimentos atômicos, tais como inserção, ejeção e fecho, para gerar sequências destes movimentos que minimizem o makespan. Esta heurística utiliza um resolvedor comercial para realizar a busca na vizinhança em uma estrutura de algoritmo de múltiplos inícios. A abordagem apresenta um bom desempenho computacional considerando dois conjuntos de instâncias testes previamente publicados na literatura científica.

Palavras-chave: vizinhança MIP, metaheurísticas híbridas, programação de máquinas paralelas não relacionadas, makespan.

1 INTRODUCTION

In the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times, there is a set of *n* jobs that must be, one by one, processed without interruption by exactly one machine among *m* available. In the case where the machines are unrelated, the processing time of a job depends on the machine where it is processed. The machine preparation time (setup time) for this problem is machine and sequence dependent, i.e., the setup time computed on the machine *k* between jobs *i* and *j* is different from the setup time computed on the same machine between jobs *j* and *i*. Similarly, the setup time between jobs *i* and *j* on machine *k* is different from the setup time between jobs *i* and *j* on machine *k'*. We considered the objective of minimizing the maximum completion time of the latest job (makespan criteria). This problem is denoted as $R|S_{ijk}|C_{max}$ Pinedo (2008), following the standard three-fields classification scheme introduced by Graham et al. (1979). Unrelated machines represent more adequately the problems found in the real world and is a generalization of scheduling problems on identical and uniform machines. There are several works in the literature addressing the unrelated parallel machine scheduling problem, such as França et al. (1996), Weng et al. (2001), Kim et al. (2002), Low (2005), Chen (2006), Chen and Wu (2006), Rabadi et al. (2006), de Paula et al. (2007), Logendran, McDonell and Smucker (2007) and Armentano and de França Filho (2007). In the production chain, machines must be reconfigured (setup) depending on the jobs to be executed and therefore the setup time might not be the same for different sequences and machines. Therefore, when modeling this process mathematically, setup time involving distinct activities must be specified separately from the jobs processing times. Allahverdi, Gupta and Aldowaisan (1999) present a review of scheduling problems with setup times. For the problem at hand, Vallada and Ruiz (2011) present a genetic algorithm and propose a set of instances with distinct characteristics.

It is known that there must not be any algorithm able to fully handle the NP-hard class of problems, however there are many instances of this class that can be solved efficiently, including those of practical interest as emphasized by Nievergelt (2000). This scenario allows the application of a hybridization strategy, by combining the use of exact methods on well defined subproblems returned by a heuristic search. The exact method provides the optimal solution of such subproblems while the heuristic search allows the exploration of the problem space at a lower computational cost. In Talbi (2002), Dumitrescu and Stützle (2003), Puchinger and Raidl (2005), Fernandes and Lourenço (2006) and Jourdan, Basseur and Talbi, (2009) are described distinct taxonomies for the hybrid methods and a survey of published studies using this technique, including works on scheduling problem.

Here we proposed a new approach that hybridizes local search based metaheuristic and exact algorithms to minimize the makespan for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. We propose a new MIP-based neighborhood used to find a sequence of job movements in order to minimize the objective function. The level of intensification of the search can be controlled by limiting computation time and the size of the MIP subproblems. According to Dumitrescu and Stützle (2003) our approach can be classified as an exact algorithm that explores large neighborhoods within local search. Furthermore, according to the classification of Puchinger and Raidl (2005), our approach can be named as an integrative combination that incorporates exact algorithms to search neighborhoods in local search based metaheuristics. Computational results show that the proposed approach outperforms the genetic algorithm presented by Vallada and Ruiz (2011) on its respective benchmark instances.

This article is organized as follows: Section 2 describes the mathematical model proposed by Vallada and Ruiz (2011) for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. Section 4.1 describes two constructive heuristics. Section 3 presents a MIP-based neighborhood and some important considerations are discussed. Section 4 defines two constructive heuristics, a local search procedure, and a multi-start algorithm. Computational results are discussed in Section 5, followed by conclusions and references.

2 PROBLEM FORMULATION

In this section it is presented a mixed integer programming model for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times as described in Vallada and Ruiz (2011). In order to simplify the model description we introduce additional notation as described below:

n : number of jobs.

m : number of machines.

M: {1,...,*m*} machine set.

N: {1,...,*n*} job set.

V: sufficiently large number.

p_{ij}: processing time of job $j \in N$, on machine $i \in M$.

s_{ijk}: setup time on machine $k \in M$, in order to process job $j \in N$, immediately after job $i \in N$.

Each machine *k* is instantiated with a dummy job 0.

The model involves the following decision variables:

$$u_{kij} = \begin{cases} 1 & \text{f job } i \text{ immediately precedes job } j \text{ in machine } k, \\ 0 & \text{otherwise.} \end{cases}$$

$C_{ki} \geq 0$: completion time of job *i* at machine *k*.

$C_{max} \geq 0$: maximum completion time.

The problem is to allocate *n* jobs in *m* machine to be processed such that the maximum completion time or makespan is minimized. The mathematical model is defined as follows:

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

Subject to:

$$\sum_{k\in M}\sum_{\substack{i\in\{0\}\cup\{N\}\\ i\neq j}} u_{ijk} = 1 \qquad \forall j\in N \tag{2}$$

$$\sum_{k\in M}\sum_{\substack{j\in N\\ i\neq j}} u_{kij} \leq 1 \qquad \forall i\in N \tag{3}$$

$$\sum_{j\in N} u_{k0j} \leq 1 \qquad \forall k\in M \tag{4}$$

$$\sum_{\substack{h\in\{0\}\cup\{N\}\\ h\neq i, h\neq j}} u_{khi} \geq u_{kij} \qquad \forall i, j\in N, i\neq j, \forall k\in M \tag{5}$$

$$C_{kj} + V(1-u_{kij}) \geq C_{ki} + s_{kij} + p_{kj} \qquad \forall i\in \{0\}\cup\{N\}\; \forall j\in N, i\neq j, \forall k\in M \tag{6}$$

$$C_{k0} = 0 \qquad \forall k\in M \tag{7}$$

$$C_{ki} \geq 0 \qquad \forall i\in N, \forall k\in M \tag{8}$$

$$C_{\max} \geq C_{ki} \qquad \forall i\in N, \forall k\in M \tag{9}$$

$$u_{kij} \in \{0,1\} \qquad \forall i\in \{0\}\cup\{N\}\; \forall j\in N, i\neq j, \forall k\in M \tag{10}$$

The objective function minimizes the maximum completion time or makespan (1). Constraint set (2) ensures that each job is assigned to exactly one machine and has exactly one predecessor. Set (3) requires that each job can have only one successor. The constraint set (4) limits the number of successors of the dummy job 0 to a maximum of one on each machine. Set (5) ensures that a job *i* and its predecessor *h* are allocated to the same machine. Set (6) regulates the jobs completion times at the machines (see Vallada and Ruiz, 2011). Constraint sets (7) and (8) define the completion time as 0 for the dummy job and the operative completion time values for other jobs, respectively. Set (9) defines the maximum completion time. Finally, set (10) defines the variable's domain.

Computational experiments with this mathematical model using a mixed integer linear solver and a computer described in Section 5, show that the performance is relatively poor, and we are not able to prove the optimality or to find a good solution for most instances over 50 jobs.

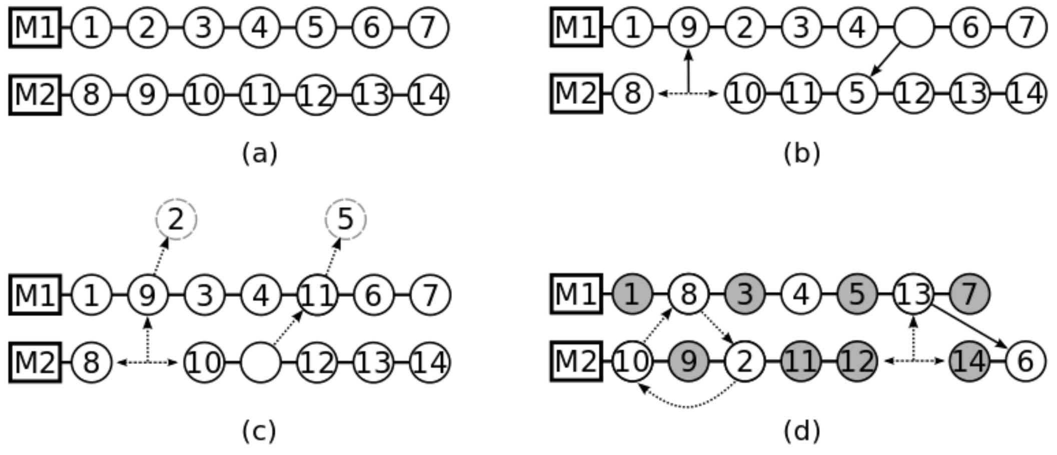
3 MIP–BASED NEIGHBORHOOD SEARCH

This section describes our proposed MIP-based neighborhood search. The neighborhood is based on three atomic moves that change the job position to the same or to another machine. The atomic moves for a job *j* are defined as follow:

- Atomic insertion move: inserts the job *j* as the successor of the job *i*.
- Atomic ejection move: replaces job *i* by job *j*, ejecting *i* of its original position.
- Atomic closure move: link the predecessor to successor of job *j*.

Observe that the definition of the atomic insertion and ejection moves does not specify the origin of job *j*. In practice, the job *j* can be derived by one ejection move or by simply selecting the job in its original position. Because of the second case, it is necessary to define the third atomic move (closure) that links the predecessor to the successor of job *j*. Furthermore, in the definition of atomic ejection move, the destination of job *i* is also not specified, but considering the overall procedure the job will be linked with another atomic move (insertion or ejection). Therefore by using those atomic moves as building blocks, it is possible to construct two types of move sequences: *cycle* and *chain*. The cycle move sequence is composed by a set of atomic ejection moves, while the chain move sequence is composed by a set of one insertion, one closure and one atomic ejection moves. Figure 1 illustrates the atomic moves and the move sequences. In (a) two machines, M1 and M2, are displayed having 7 jobs each. In (b), two atomic insertion moves are illustrated. The first atomic insertion move, where *i* = 1 and *j* = 9, is a simple insertion move. Observe that an atomic closure move is necessary to link the jobs 8 and 10. In the second insertion move (*i* = 11 and *j* = 5), the origin of job 5 is given by an atomic ejection move. The same representation is used in (c), where two atomic ejection moves are shown. Finally in (d), is presented an illustration of a cycle move sequence involving jobs 2, 8, and 10 concomitantly with a chain move sequence involving jobs 6 and 13. Note that jobs in gray can not belong to other moves. This constraint is detailed in the next subsection.

Figure 1 – Representation of atomic moves: insertion (b), ejection (c) and cycle and chain (d).



The mathematical model described in subsection 3.1 is used to select the best combination of independent sequences of moves that lead to an improvement in the objective function.

3.1 Mathematical Model for the Neighborhood Search

Formally, the mathematical model applied to search the neighborhood is defined on the current solution and the goal is to find sequences of atomic moves that lead to a reduction of the makespan.

Let *pre*(*i*) and *suc*(*i*) be the predecessor and successor of the job *i* in the current solution, respectively. Observe that the predecessor of the first job in the machine *m* is a dummy job named 0. Furthermore, the successor of the last job in the machine is a dummy job as well, and all costs associated to this dummy job are zero. Thus, the costs of movements are calculated as follows. Let $Q_{kij} = p_{kj} + s_{kij}$ be the cost of allocation of the job *j* as immediate successor of the job *i* allocated on the machine *k*. Note that in the current solution, the machine *k* is the machine in which job *i* is allocated. Therefore, the cost of removing job *j* is calculated by $cO_j = -Q_{k,pre(j),j} - Q_{kj,suc(j)}$.

The cost to insert job *j* in the position *i* is calculated by $cX_{ij} = Q_{k,pre(i),j} + Q_{kj,suc(i)}$, and the cost to insert job *j* as immediate successor of job *i* is calculated by $cY_{ij} = -Q_{ki,suc(i)} + Q_{kij} + Q_{kj,suc(i)}$. Finally, $cW_i = Q_{k,pre(i),suc(i)}$ is the cost of linking the predecessor and the successor of job *i*.

In the following, we present some additional notation to describe the model.

D: set of dummy jobs.

$N^* = D \cup N$: set of jobs including the dummy jobs.

U_m: set of jobs assigned to machine $m \in M$.

cM_m: current completion time on machine *m*.

W: a sufficiently large number.

The following variables are used in the model.

$$x_{ij} = \begin{cases} 1 & \text{if job } j \text{ is inserted at the position from which job } i \text{ is ejected.} \\ 0 & \text{otherwise.} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if job } j \text{ is inserted as the immediate sucessor of job } i. \\ 0 & \text{otherwise.} \end{cases}$$

$$w_i = \begin{cases} 1 & \text{if the sucessor of the job } i \text{ succeeds its predecessor after removing job } i. \\ 0 & \text{otherwise.} \end{cases}$$

$C_{max} \geq 0$: maximum completion time (makespan).

$C_m \geq 0$: completion time on machine m.

The MIP-based neighborhood search model can be formally described as follows:

$$\text{Min } W C_{\max} + \sum_{m \in M} C_m \qquad (11)$$

Subject to:

$$\sum_{j \in N} x_{ij} + \sum_{j \in N} y_j + w_i \leq 1 \qquad \forall i \in N \qquad (12)$$

$$\sum_{j \in N} y_{ij} \leq 1 \qquad \forall i \in D \qquad (13)$$

$$\sum_{i \in N} x_{ij} + \sum_{i \in N*} y_j \leq 1 \qquad \forall j \in N \qquad (14)$$

$$\sum_{i \in N} x_{ij} + \sum_{\substack{i \in N \\ pre(j) \notin D}} x_{pre(j)\,i} + \sum_{i \in N} y_j + \sum_{i \in N} y_{pre(j)\,i} \leq 1 \qquad \forall j \in N \qquad (15)$$

$$\sum_{i \in N} x_{ij} + \sum_{i \in N} x_{i,suc(j)} + \sum_{i \in N^*} y_j + \sum_{i \in N^*} y_{i,suc(j)} \leq 1 \qquad \forall j \in N \qquad (16)$$

$$\sum_{j \in N} x_{ij} \leq \sum_{j \in N} x_j + \sum_{j \in N*} y_j \qquad \forall i \in N \qquad (17)$$

$$\sum_{j \in N^*} y_{ji} + \sum_{j \in N} x_j \leq \sum_{j \in N} x_j + w_i \qquad \forall i \in N \qquad (18)$$

$$\sum_{j\in N} y_{ij} + \sum_{j\in N} x_{ij} + \sum_{j\in N^*} y_{ij} \le 1 \qquad \forall i\in N \qquad (19)$$

$$\sum_{i\in N}\sum_{j\in U_m} c^{O}_{ij} x_{ij} + \sum_{i\in U_m}\sum_{j\in N} c^{X}_{ij} x_{ij} + \sum_{i\in N^*}\sum_{j\in U_m} c^{O}_{ij} y_{ij} + \sum_{i\in U_m\cup\{0\}}\sum_{j\in N} c^{Y}_{ij} y_{ij} + \sum_{i\in U_m} c^{W}_{i} w_i + c^{M}_{m} \le C \qquad \forall m\in M \qquad (20)$$

$$C_m \le C_{\max} \qquad \forall m\in M \qquad (21)$$

$$\sum_{j\in N} x_{ji} + \sum_{j\in N} y_{ij} - w_i \ge 0 \qquad \forall i\in N \qquad (22)$$

$$C_{max} \ge 0,\ C_m \ge 0,\ x_{ij}, y_{kj}, w_i \in \{0,1\} \qquad \forall m\in M,\ \forall i, j\in N,\ \forall k\in N^* \qquad (23)$$

The objective is to minimize the maximum completion time (makespan) and the sum of the completion time of all jobs (flow time). The parameter *W* ensures that the makespan will be the first to be minimized. Even when it is not possible to improve the makespan, a better flow time configuration may improve the makespan in the next iteration (by default we use $W = 10^6$). Constraint sets (12), (13) and (14) determine that a job can be involved in only one move. Sets (15) and (16) state that the predecessor and successor of the job *i* remains the same when the job is involved in a move. These constraints are necessary so that the current cost of the moves may be accurately computed. Sets (17) and (18) determine that an ejected job must be involved in another move. Specifically about the constraint set (18), if a job is removed from its original position, another job should occupy that position or its predecessor and its successor must be connected in sequence. This fact is represented by the variables denoted by *x* and *w*. Set (19) implies that when a new successor is defined to a job, this job remains the same. In addition, it also implies that if a job is moved, no other job can be inserted as its predecessor. Set (20) determines the completion time of each machine (see the extended example in Appendix A for more details of this constraint set). Set (21) determines the new makespan value. Set (22), strictly speaking, can be dropped from the model, since it is used to prevent variables *w* to assume value 1 without being part of an atomic move. The only drawback is the need to analyze which variables *w* with value 1 do not compose the model solution. Finally, set (23) defines the variable's domain.

The number of variables in the mathematical model for the neighborhood search is highly dependent on the number of jobs. For the atomic insertion moves (y_{ij}), variables are created for each job $j \in N$ to insert its jobs as successors of $i \in N^*$ (with $i \neq j$ and $i \neq pre(j)$). Thus, the number of variables *y* is given by $n^2 + n(m - 2)$, where *m* and *n* represent respectively the number of machines and jobs of the instance under analysis. Similarly, for the atomic ejection move, the number of variables *x* is $n^2 - 3n + 2m$ and the number of the variables *w* is $n–m$ for the atomic closure move. Considering the cost variables, the total number of variable is $2n^2 + n(m - 4) + 2m + 1$.

The costs of moving a job *j* is calculated based on its predecessor and its successor. This creates constraints to forbid any further moves that would modify these two jobs adjacent to *j*, since the computation of the cost of moving *j* depends on them. These constraints are defined in (15), (16) and (19). Thus, in the best case, this neighborhood allows approximately 50% of the jobs to be moved. This provides an idea of the size of the neighborhood explored by our approach. To mitigate such a time consuming search, we devise two techniques. First we abdicate of proving the optimality of the model and stop the solver by adding an extra parameter that limits the computation time (t_{max}). Second, we consider a technique to eliminate some variables with less probability of belonging to good move sequences. This technique is explained in more details on the next subsection.

3.2 Size-Reduction Method

Some of atomic moves have an associated cost (processing time) higher than others. In these cases, if our objective is to minimize the cost on each machine, we expect that moves with higher cost have little chance to be chosen and belong to the optimal solution of the neighborhood. Thus, a pre-processing technique discards heuristically some variables improving the performance of the solver. This process is called size-reduction, and is similar to the techniques applied to quadratic assignment problem in Mautor and Michelon (1997), Mautor and Michelon (2001). Stefanello and Müller (2009) applied a similar method to solve the capacitated p-median problem. Fanjul-Peyro and Ruiz (2011) use this technique on the mathematical model of unrelated parallel machines problem.

In order to describe our technique to eliminate variables of the mathematical model of the neighborhood, we define the parameters ω_y and ω_x given by the following formula:

$$w_y = \frac{\Omega}{n_y}\left(\sum_{\substack{i\in N^*, j\in N \\ i\neq j, j\neq suc(i)}} CY_{ij} \right) \text{ and } w_x = \frac{\Omega}{n_x}\left(\sum_{\substack{i, j\in N \\ i\neq j, j\neq suc(i)}} CX_{ij} \right),$$

where $CY_{ij} = cY_{ij} + cW_j$ and $CX_{ij} = cX_{ij}$ is an associated cost, and n_y and n_x are respectively the number of atomic insertion and ejection moves. The parameter Ω represents an expansion/decreasing factor, which defines the number of variables to be eliminated. Parameters ω_y and ω_x represent an upper bound in such a way that a variable x_{ij} is eliminated if $CX_{ij} > \omega_x$ and a variable y_{ij} is eliminated if $CY_{ij} > \omega_y$.

4 LOCAL SEARCH AND MULTI-START ALGORITHM

This section describes a multi-start algorithm and a local search procedure that use the MIP-based neighborhood search described in Section 3. We start from two constructive heuristic to provide an initial solution from the local search and multi start-algorithm method.

4.1 Constructive Heuristic

We need to generate an initial solution for the local search algorithm as well as for the multi-start algorithm described in subsection 4.2 and 4.3, respectively. We implemented two constructive heuristics. In the first constructive heuristic, called greedy initial solution, the jobs are assigned to the machines that produce the lowest total processing time increase. This constructive heuristic is divided into two steps. First step rank the jobs in an increasing order of cost $c_j = \sum_{m\in M} \sum_{i\in N^*} Q_{mij}$. The second step assigns each job j in the previous order to machine $m' = \arg\min_m \left(L_m + Q_{mlj}\right)$, were L_m is the current processing time on machine m and Q_{mlj} is the cost of inserting the job j after the last job on machine m.

In the second constructive heuristic, called random initial solution, a non assigned job j is randomly selected (by uniform distribution) and assigned to machine $m' = \arg\min_m \left(L_m\right)$. This constructive heuristic is very simple, with low complexity and when embedded in the multi-start algorithm, allows starting from distinct points in the search space.

4.2 Local Search Algorithm

Local search is a general approach for finding and improving solutions to hard combinatorial optimization problems. The most basic strategy of local search algorithms is to start from an initial solution and iteratively try to replace the current solution by a better neighbor solution, until no improvement can be reached. In this subsection we describe a basic local search procedure that use the moves returned by the mathematical model describe in the subsection 3.1 to obtain a neighbor solution.

Two parameters are defined for the local search procedure: the minimum time (t_{min}) and the maximum time (t_{max}). These parameters are used to limit the computation time allocated for the solver to explore the neighborhood. The parameter t_{min} determines the minimum time that the solver works after having found a feasible solution and the parameter t_{max} is adopted to keep the computational time within reasonable limits.

Algorithm 4.1 illustrates the proposed local search algorithm. The main loop of the algorithm is iterated as long as possible to update the incumbent solution S^*. The procedure, called Size_Reduction(S), pre-processes the solution S to eliminate some variables as describe in subsection 3.2. The procedure *Neighborhood_Search*(S,R,t_{max},t_{min}) uses the mathematical model of the neighborhood (described in section 3.1), solved by a commercial solver to find moves that improve the objective function. Yet, the set R of variables are eliminated from the model and the parameters t_{max} and t_{min} bound the execution of the solver. In the next step (line 5 -7), the solution is updated if the incumbent solution was improved. The solution S is evaluated by $f(S) = WC_{max} + \sum_{m \in M} C_m$ (as described in (11)). At the end, the solution S^* is returned (line 9).

Algorithm 4.1: Local_Search(S,t_{max},t_{min})

```
1  Initialize(f(S*), ∞);
2  while f (S) <f (S*) do
3          R ←Size_Reduction(S);
4          Neigborhood_Search(S,R,t_max,t_min);
5          if f(S) <f(S*) then
6                  S*←S;
7          end
8  end
9  return(S*);
```

4.2 Multi-Start Algorithm

Sometimes, the local search is not able to find the best known solution for small instances. This is because a small number of jobs generate few possible moves for the local search, which converges quickly to a local optimal in such a case. To handle this issue, we implemented a multi-start algorithm to improve the coverage of our approach during the neighborhood search.

Algorithm 4.2 shows a pseudo-code for the proposed multi-start algorithm. In the first phase (lines 1–3) a greedy initial solution is generated by the procedure *Greedy_Constructive*(S) followed by a *Local_Search*(S,t_{max},t_{min}) procedure, as described in the subsection 4.2. The incumbent solution is initialized on line 3. The second phase (lines 5–9), is similar to the first phase, however this phase is repeated by nK_{iter} times, and each iteration start with a random initial solution (describe in subsection 4.1) represented by the procedure *Random_Constructive*(*S*). Finally, the best solution is returned.

Algorithm 4.2: Multi_Start_Alg(K_{iter})

```
1   Greedy_Constructive(S);
2   Local_Search(S,$t_{max}$,$t_{min}$);
3   $S^* \leftarrow S$;
4   for iter← 1 to $nK_{iter}$
5           Random_Constructive(S);
6           Local_Search(S,$t_{max}$,$t_{min}$);
7           if $f(S) < f(S^*)$ then
8                   $S^* \leftarrow S$;
9           end
10  end
11  return($S^*$);
```

We observe that when $nK_{iter} = 0$ the steps in lines 4 to 10 are not executed, thus referring to the local search procedure in its original form.

5 COMPUTATIONAL RESULTS

The computational results presented were obtained using CPLEX 12.1 C++ API with default configuration on a Intel Quad-Core Xeon X3360 2.83 GHz. The set of test instances is the one proposed by Vallada and Ruiz (2011), available at http://soa.iti.es, and is divided into two groups: 640 small instances and 1000 large instances. Each group is subdivided into 10 instances with distinct numbers of machines and jobs and different ranges for the setup time values. We define the average relative deviation parameter (*ARD*) to evaluate the quality of solutions in accordance with the equation $ARD = 100(S_h - S_g)/\min(S_h,S_g)$, where S_h corresponds to the value of makespan obtained by our approach and S_g is the value of objective function (makespan) found by the genetic algorithm proposed by Vallada and Ruiz (2011), which uses a stopping criteria based on computational time defined by $n(m/2)t$, where t assumes 50 milliseconds as its maximum value. Thus, the CPU time limit for the genetic algorithm varies from 12.5 to 187.5 seconds on a computer with an Intel Core 2 Duo, 2.4 GHz with 2 GB of RAM memory. Several tests were conducted to evaluate the proposed neighborhood. First, we used the multi-start algorithm for small instances. Since this approach provided good results, we concentrated on tests of large instances using local search with size-reduction strategy.

5.1 Results for Small Size Instances

In this subsection are presented the results for the multi-start algorithm applied on small instances.

In this set of instances, the solver is able to prove the optimality of the mathematical model for the neighborhood in small computational time. Thus, in this set of experiments we do not use the size-reduction procedure and the parameters t_{min} and t_{max} were set to zero and to a sufficiently large value, respectively. The multi-start algorithm was applied to a maximum number of iteration equal nK_{iter} with $K_{iter} = 3$ starting from a random initial solution plus the first iteration with the greedy initial solution, totalizing $3n + 1$ iterations.

Table 1 presents the results relative to the first iteration that found the best known solution and the total execution of multi-start algorithm. The first two columns represent the number of jobs and machines, respectively. Note that each line represents a set of 40 instances (four sets of ten instances with setup times belonging to the following intervals: [1 , 9], [1 , 49], [1 , 99] and [1 , 124]). The next three columns are the following: the average number of iterations (avg), the maximum number of iterations (max) and the average running time in seconds for the first iteration where the multi-start algorithm found the best known solution (time (s)). The last two columns show the total number of iterations of each instance, and the average run time in seconds of the multi-start algorithm, respectively.

Table 1: Results of multi-start algorithm to find the best known solution and total number of iterations and running time for small instances

		Best			**Total**	
n	*M*	Avg	Max	time (s)	*nKiter*	time (s)
6	2	1.45	6	0.05	19	0.55
	3	1.85	13	0.05	19	0.56
	4	1.20	3	0.03	19	0.58
	5	1.15	3	0.02	19	0.50
8	2	2.43	15	0.15	25	1.56
	3	1.85	11	0.14	25	1.89
	4	2.08	9	0.14	25	1.77
	5	1.38	6	0.07	25	1.52
10	2	4.88	29	0.46	31	3.03
	3	2.50	13	0.31	31	4.07
	4	3.25	18	0.48	31	4.40
	5	2.20	13	0.24	31	3.78
12	2	6.10	31	1.00	37	6.58
	3	4.78	26	0.89	37	7.24
	4	5.18	33	1.33	37	8.96
	5	4.73	31	0.96	37	7.70
Average		2.94		0.39		3.42

Table 1 shows that the best known solutions are found in a relative small number of iterations and a low running time of the multi-start algorithm. In average, the algorithm need less than 3 iterations, and at most 2 seconds to find the optimal solution. For the worst case, it took 33 iterations.

Table 2 show the results for different values of K_{iter} indicated on the first column. The second column indicates the number of jobs representing a set of 160 instances with different numbers of machines and range of setup time. The third column shows the average value of *ARD* and the next column (best) shows the percentage of instances which the multi-start algorithm found the best known solution. Finally, the last column shows the average computational time in seconds.

Table 2: Average results for small instances

K_{iter}	Jobs	*ARD*	best (%)	Time (s)
0	6	1.45	81.25	0.02
	8	3.78	63.13	0.06

K_{iter}	Jobs	*ARD*	best (%)	Time (s)
	10	2.87	55.00	0.12
	12	3.44	40.00	0.22
Average		2.88	59.84	0.11
0.5	6	0.20	97.50	0.11
	8	0.28	95.00	0.35
	10	0.36	88.75	0.77
	12	0.63	80.00	1.45
Average		0.37	90.31	0.67
1	6	0.02	99.38	0.21
	8	0.04	98.75	0.64
	10	0.12	93.75	1.38
	12	0.24	90.00	2.68
Average		0.11	95.47	1.23
2	6	0.00	100.00	0.39
	8	0.00	100.00	1.16
	10	0.01	98.75	2.60
	12	0.02	96.25	5.13
Average		0.01	98.75	2.32
3	6	0.00	100.00	0.55
	8	0.00	100.00	1.68
	10	0.00	100.00	3.82
	12	-0.04	100.00	7.62
Average		-0.01	100.00	3.42

Table 2 shows that by using only the local search procedure (K_{iter} = 0), our approach obtains the best solution known for 59.84% of the cases. The multi-start algorithm provides an improvement in *ARD* values and in the percentage of the best known solutions found without requiring expressive extra computational time. When K_{iter} = 3 we were able to find the best known solutions for all instances tested.

An interesting fact to be considered is a negative *ARD* value obtained for one instance with 12 jobs and 5 machines. A value *ARD* < 0 implies that the solution found by us is better than the global optimal reported by Vallada and Ruiz (2011). A reasonable explanation is that the problem can be caused by a misconfiguration of the tolerance and accuracy values of the CPLEX solver used by Vallada and Ruiz (2011).

5.2 Results for Large Size Instances

In this subsection, the computational results are reported for the large instances using local search procedure and local search enhanced with size-reduction method. The first experiment considers only the local search (with t_{max} = (n-m/2)0.5, t_{min} =0.01t_{max} and without the size-reduction method). The solutions obtained by local search overcame genetic algorithm solutions, because we found 93.5% of the best known solutions so far, and 90.00% of them are strictly better. The ARD for all sets of instances with the parameters mentioned above was -11.17%, with reduction up to -38.10% for an instance with 250 jobs and 30 machines. The average time per instance was 201.46 seconds.

The second experiment evaluates the local search procedure enhanced with the size-reduction method. For this case, we set t_{max} = (n-m/2)0.2, reduction factor Ω = 0.3 and the initial solution was generated by the greedy constructive method. Table 3 shows the average results obtained at each set of 40 instances, in which first and second columns correspond to the number of machines and jobs. The next column (best) shows the percentage of instances for which the local search could match or improve the results of the genetic algorithm. Finally, the last columns report the average time in seconds for the genetic algorithm and the proposed method, respectively.

Table 3: Average results for local search procedure + size-reduction.

Jobs	Machine	*ARD*	best (%)	Time GA (s)	Time LS (s)
50	10	3.92	30.00	13	4.40
	15	-3.57	80.00	19	2.71
	20	-7.30	95.00	25	2.05
	25	-10.66	100.00	31	1.30
	30	-12.23	97.50	38	1.18
100	10	0.52	50.00	25	16.40
	15	-4.03	87.50	38	19.30
	20	-7.22	95.00	50	17.65
	25	-15.51	100.00	63	18.00
	30	-21.11	100.00	75	15.72
150	10	-2.52	82.50	38	32.97
	15	-5.19	90.00	56	32.86
	20	-11.81	97.50	75	39.15
	25	-15.82	100.00	94	33.83
	30	-20.05	100.00	113	32.91
200	10	-5.40	100.00	50	51.19
	15	-8.58	100.00	75	52.73
	20	-12.93	100.00	100	56.00
	25	-17.72	100.00	125	58.98
	30	-21.24	100.00	150	54.86
250	10	-6.38	100.00	63	71.59
	15	-10.98	100.00	94	71.17
	20	-13.58	100.00	125	72.65
	25	-20.12	100.00	156	82.18
	30	-22.99	100.00	188	79.00
Average		-10.90	92.20	75	36.83

The results reported in Table 3 show that the average computational time was approximately 2 times lower than those reported in Vallada and Ruiz (2011), which is representative even considering the differences in hardware. We were able to match or improve 92.20% of the results, with an average improvement of 10.90% in comparison with the genetic algorithm. As another remark, the size-reduction method allows a significant reduction of computational time. With a small number of variables the solver improves the solution in small time allowing the reduction of parameter t_{max} and consequently the total running time. In comparison with the local search without size-reduction method, the average running time was reduced from 201.46 to 36.83 seconds without significantly reduction in the solution quality, which was 11.17% to 10.90%.

The local search reduces by 50% the initial solution value in approximately 17 iterations, and it works this way even for low quality initial solutions.

6 CONCLUSIONS AND FUTURE RESEARCH

This work presented a hybridization of exact and heuristic methods to solve the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times in order to minimize the makespan criteria. As a strategy to solve this problem, we proposed a MIP-based neighborhood that makes use of commercial solvers. This neighborhood was applied with a local search and a multi-start algorithm. A size-reduction method was proposed too. The experiments show that our approach is effective in obtaining good solutions for the considered problem. Our strategy produced results which overcame those reported by Vallada and Ruiz (2011), who applied a genetic algorithm on the same set of instances. For the small instances, the multi-start algorithm obtained better or equal results in all cases with K_{iter} = 3. For large instances it was possible to find better solutions in most of the cases. Our instances and solutions are available on the web at www.inf.ufsm.br/~stefanello/instances/.

ACKNOWLEDGEMENTS

The first author was partially supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

REFERENCES

ALLAHVERDI, A.; GUPTA, J. N. D.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. **Omega**, v.27, n.2, p.219–239, 1999.

ARMENTANO, V. A.; DE FRANÇA FILHO, M. F. Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach. **European Journal of Operational Research**, v.183, n.1, p.100–114, 2007.

CHEN, J.-F. Minimization of maximum tardiness on unrelated parallel machines with process restrictions and setups. **The International Journal of Advanced Manufacturing Technology**, v.29, n.5, p.557–563, 2006.

CHEN, J.-F.; WU, T.-H. Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. **Omega**, v.34, n.1, p.81–89, 2006.

DE PAULA, M. R.; RAVETTI, M. G.; MATEUS, G. R.; PARDALOS, P. M. Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighborhood search. **IMA Journal of Management Mathematics**, v.18, n.2, p.101–115, 2007.

DUMITRESCU, I.; STÜTZLE, T. Combinations of Local Search and Exact Algorithms. In:

CAGNONI, S.; JOHNSON, C.; CARDALDA, J.; MARCHIORI, E.; CORNE, D.; MEYER, J.-A.; GOTTLIEB, J.; MIDDENDORF, M.; GUILLOT, A.; RAIDL, G.; HART, E. (Eds.), **Applications of Evolutionary Computing**, v.2611, Springer Berlin Heidelberg, p.211–223, 2003.

FANJUL-PEYRO, L.; RUIZ, R. Size-reduction heuristics for the unrelated parallel machines scheduling problem. **Computers & Operations Research,** v.38, n.1, p.301–309, 2011.

FERNANDES, S.; LOURENÇO, H. Optimised Search Heuristics: A Mapping of procedures and Combinatorial Optimisation Problems. **Technical Report**, Universidade do Algarve, Faro, Portugal, 2006.

FRANÇA, P. M.; GENDREAU, M.; LAPORTE, G.; MÜLLER, F. M. A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. **International Journal of Production Economics,** v.43, n.2–3, p.79–89, 1996.

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; KAN, A. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. **Annals of discrete mathematics**, v.5, p287–326, 1979.

JOURDAN, L.; BASSEUR, M.; TALBI, E.-G. Hybridizing exact methods and metaheuristics: A taxonomy. **European Journal of Operational Research,** v.199, n.3, p.620–629, 2009.

KIM, D.-W.; KIM, K.-H.; JANG, W.; CHEN, F. F. Unrelated parallel machine scheduling with setup times using simulated annealing. **Robotics and Computer-Integrated Manufacturing**, v.18, n.3–4, p.223–231, 2002.

LOGENDRAN, R.; MCDONELL, B.; SMUCKER, B. Scheduling unrelated parallel machines with sequence dependent setups. **Computers & Operations Research,** v.34, n.11, p.3420–3438., 2007.

LOW, C. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. **Computers & Operations Research**, v.32, n.8, p.2013–2025, 2005.

MAUTOR, T.; MICHELON, P. MIMAUSA: A new hybrid method combining exact solution and local search. **Proceedings of the 2nd International Conference on Metaheuristics**, p.15, Sophia-Antipolis, France, 1997.

MAUTOR, T.; MICHELON, P. MIMAUSA: an application of referent domain optimization. **Technical Report**, 260, Laboratoire d'Informatique, Université d'Avignon et des Pays de Vaucluse, 2001.

NIEVERGELT, J. Exhaustive Search, Combinatorial Optimization and Enumeration: Exploring the Potential of Raw Computing Power. **SOFSEM 2000: Theory and Practice of Informatics**, LNCS, v.1963, p.18-35, 2000.

PINEDO, M. Scheduling: **Theory, Algorithms, and Systems**. 3th ed, New Jersey: Prentice Hall, 678 p, 2008.

PUCHINGER, J.; RAIDL, G.R. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. **International Work-Conference on the Interplay Between Natural and Artificial Computation - IWINAC**, Part II. LNCS 3562, p.41–53, 2005.

RABADI, G.; MORAGA, R.; SALEM, A. Heuristics for the unrelated parallel machine scheduling problem with setup times. **Journal of Intelligent Manufacturing**, v.17, n.1, p.85–97, 2006

TALBI, E. G. A taxonomy of hybrid metaheuristics. **Journal of Heuristics**, v. 8, n. 5, p.541–564, 2002.

VALLADA, E.; RUIZ, R. Genetic algorithms for the unrelated parallel machine scheduling problem with sequence dependent setup times. **European Journal of Operational Research**, v.211, n.3, p.612–622, 2011.

WENG, M. X.; LU, J.; REN, H. Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. **International Journal of Production Economics**, v.70, n.3, p.215–226, 2001.

APPENDIX A. EXTENDED EXAMPLE

In order to elucidate how the local search and the MIP-based neighborhood work, we extend a small and simple example explaining all process and computation of the resolution. First, we present the instance. The number of jobs is n = 4 and the number of machines m = 2. For each job j such that j = {1,2,3,4}, the processing time in the first machine is p_{1j} = {4,8,4,6} and in the second machine is p_{2j} = {2,6,7,7}. The setup time is presented in Table A.4, where s_{213} = 1 is the setup time to process the job 3 immediately after processing the job 1 on machine 2.

Table A.4: Setup time of job on the machines 1 and 2

	Machine 1				Machine 2			
Job	1	2	3	4	1	2	3	4
1	-	1	8	1	-	5	1	6
2	4	-	7	3	6	-	7	7
3	7	3	-	2	7	6	-	4
4	3	8	3	-	3	7	3	-

We start from a solution m_1: 1 → 2 and m_2: 3 → 4; the cost of allocation each job is given by $Q_{kij} = s_{kij} + p_{kj}$. We represent jobs a and b as the dummy jobs of first and second machine, respectively. Observe that the setup time to allocate the first job in the machine is zero. Thus, we can calculate the cost:`

Job 1: $Q_{1a1} = 0 + 4 = 4$
Job 2: $Q_{112} = 1 + 8 = 9$

Job 3: $Q_{2b3} = 0 + 7 = 7$
Job 4: $Q_{234} = 4 + 7 = 11$

obtaining a cost of 15 and 18 for the first and second machine, respectively. The value of makespan is max{15,18} = 18.

The next step is to build the mathematical model of neighborhood and we start calculating the costs cO_j, cW_j, cX_{ij}, cY_{ij} on the current solution. Let $pre(i)$ and $suc(i)$ be the predecessor and successor of the job i, respectively (as defined in subsection 3.1). The cost cO_j is the cost of remove the job j from machine k, and is calculated by $cO_j = -Q_{k,pre(j),j} - Q_{k,j,suc(j)}$. In our example we have:

cO_1: $-Q_{1a1} - Q_{112} = -4 + 9 = -13$
cO_2: $-Q_{112} = -9$

cO_3: $-Q_{2b3} - Q_{234} = -7 - 11 = -18$
cO_4: $-Q_{234} = -11$

The cost cW_j is the cost of linking the predecessor and the successor of job j allocated on machine k. This cost is calculated by $cW_j = Q_{k,pre(j),suc(j)}$.

cW_1: $Q_{1a2} = 8$
cW_3: $Q_{2b4} = 7$

Specifically for the ejection move, the cost $cX_{ij} = Q_{k,pre(i),j} + Q_{k,j,suc(i)}$ is the cost of inserting job j in the position of job i. In this case we have 8 possibilities.

cX_{31}: $Q_{2b1} + Q_{214} = 2 + 13 = 15$
cX_{32}: $Q_{2b2} + Q_{224} = 6 + 14 = 20$
cX_{13}: $Q_{1a3} + Q_{132} = 4 + 11 = 15$
cX_{14}: $Q_{1a4} + Q_{142} = 6 + 16 = 22$

cX_{41}: $Q231 = 9$
cX_{42}: $Q_{232} = 12$
cX_{23}: $Q_{113} = 12$
cX_{24}: $Q_{114} = 7$

Associated with the atomic insertion move, cost of inserting job j as immediate successor of job *i* is calculated by $cY_{ij} = -Q_{ki,suc(i)} + Q_{kij} + Q_{kj,suc(i)}$. In this case we have 16 possibilities.

cY_{21}: $Q_{121} = 8$	cY_{b1}: $-Q_{2b3} + Q_{2b1} + Q_{213} = 3$
cY_{31}: $-Q_{234} + Q_{231} + Q_{214} = 11$	cY_{41}: $Q_{241} = 5$
cY_{a2}: $-Q_{1a1} + Q_{1a2} + Q_{121} = 12$	cY_{b2}: $-Q_{2b3} + Q_{2b2} + Q_{223} = 13$
cY_{32}: $-Q_{234} + Q_{232} + Q_{242} = 15$	cY_{42}: $Q_{242} = 13$
cY_{a3}: $-Q_{1a1} + Q_{1a3} + Q_{131} = 11$	cY_{13}: $-Q_{112} + Q_{113} + Q_{132} = 14$
cY_{23}: $Q_{123} = 11$	cY_{43}: $Q_{243} = 10$
cY_{a4}: $-Q_{1a1} + Q_{1a4} + Q_{141} = 9$	cY_{14}: $-Q_{112} + Q_{114} + Q_{142} = 14$
cY_{24}: $Q_{124} = 9$	cY_{b4}: $-Q_{2b3} + Q_{2b4} + Q_{243} = 10$

With the costs computed, we can construct the mathematical model of neighborhood. The constraint (20) is responsible to evaluate the costs of the active atomic moves and we explain that in details. In our example, we have:

Table A.5: Variables and costs of constraint set (20)

		Variables relative m_1	Variables relative m_2
(a)	$\sum_{i \in N} \sum_{j \in U_m} cO_{ij} x_{ij}$	$-13x_{31} - 13x_{41}$	$-18x_{13} - 18x_{23}$ $-11x_{14} - 11x_{24}$
(b)	$+\sum_{i \in U_m} \sum_{j \in N} cX_{ij} x_{ij}$	$-9x_{32} - 9x_{42}$	$+ 15x_{31} + 20x_{32}$ $+ 9x_{41} + 12x_{42}$
(c)	$+\sum_{i \in N^*} \sum_{j \in U_m} cO_{ij} y_{ij}$	$+ 15x_{13} + 22x_{14}$ $+ 12x_{23} + 7x_{24}$	$-18y_{13} - 18y_{23} - 18y_{43}$ $-11y_{14} - 11y_{24} - 18y_{a3}$
(d)	$+\sum_{i \in U_m \cup \{0\}} \sum_{j \in N} cY_{ij} x_{ij}$	$-13y_{21} - 13y_{31} - 13y_{41}$ $-9y_{a2} - 9y_{32} - 9y$ $-13y_{b1} - 9y_{b2}$	$+ 3y_{b1} + 3y_{b2} + 10y_{b4}$ $+ 11y_{31} + 15y_{32}$ $+ 5y_{41} + 13y_{42} + 10y_{43}$
(e)	$+\sum_{i \in U_m} cW_{ij} w_i$	$+ 8w_1$	$+ 7w_3$
		$+ 13 \le C_1$	$+ 18 \le C_2$

On one hand, the parcels (a) and (c), indicate the cost to remove the job *j* on the machine *m*, if is made by one atomic ejection and insertion move, respectively. On the other hand, the parcels (b) and (d), indicate the cost on the machine *m* to insert the job *j* on position *i* by the atomic ejection move and the cost to insert the job *j* after the position *i* by the atomic insertion move, respectively. In (e), is the cost of link the successor and predecessor of job *j*. The remaining parcel indicates the current processing time on the machine *m* and the variable to store the new value of processing time. With the other constraints and using a MIP–solver, we obtain the following solution: $w_3 = 1$, $x_{13} = 1$ and $y_{41} = 1$ (the values of the remaining variables is zero). This means that the solver find one atomic ejection move (x_{13}) that insert the job 3 in the position of job 1, one atomic insertion move (y_{41}) that insert the job 1 after the position of job 4 and finally, one atomic closure move (w_3) that link the job 4 to dummy job *b*. This set of atomic move is a chain move sequence. With the variables and costs showing in Table A.5, the new values of processing time on each machine can be calculated.

m_1: +15-13 + 13 = 15
m_2: -18 + 5 + 7 + 18 = 12

Performing the movements, we obtain a new solution with makespan of max{15,12}=15. Starting from the previous solution, in next iteration of the local search we obtain a new chain move sequence ($w_4 = 1$, $x_{24} = 1$ and $y_{12} = 1$) obtaining the solution m_1: 3 → 4 and m_2: 1 → 2 with makespan equal to 13. Then the procedure stops because there was no more makespan improvement.