A Matheuristic to the Unrelated Parallel Machine Scheduling Problem

Guilherme Baumgratz Figueiroa
Universidade Federal de Ouro Preto - Departamento de Computação
Campus Universitário Morro do Cruzeiro, s/n, CEP 35400-000, Ouro Preto/MG
guilherme.baumgratz@aluno.ufop.edu.br

George Henrique Godim da Fonseca
Universidade Federal de Ouro Preto - Dep. de Eng. de Computação e Sist. de Informação
Rua 36, Número 115, Loanda, CEP 35931-088, João Monlevade/MG
george@decea.ufop.br

Túlio Ângelo Machado Toffolo
Universidade Federal de Ouro Preto - Departamento de Computação
Campus Universitário Morro do Cruzeiro, s/n, CEP 35400-000, Ouro Preto/MG
tulio@toffolo.com.br

ABSTRACT

This paper presents a Fix-and-Optimize approach for the Unrelated Parallel Machine Scheduling Problem (UPMSP). In short, the UPMSP consists of assigning jobs to unrelated parallel machines where different processing times incur for the same job in different machines. Additionally, a setup time is considered between the execution of jobs in the same machine. Fix-and-Optimize is a matheuristic that iteratively selects a subset of variables to be fixed to their current values so that the remaining variables will compose a sub-problem to be optimized by an integer programming solver. In the proposed approach, each sub-problem consists of a subset of jobs that are assigned to a subset of machines in the incumbent solution. In the conducted experiments on benchmark instances, the proposed Fix-and-Optimize algorithm achieved remarkable results. It outperformed two state-of-the-art exact approaches to the UPMSP and achieved competitive results when compared to the literature's best performing heuristic method for this problem.

KEYWORDS. UPMSP, Fix-and-Optimize, Matheuristic

1. Introduction

The Machine Scheduling Problem consists in assigning and ordering the execution of a set of jobs to a set of machines respecting a set of constraints. There are several variations of this problem. Tasks may be related (there is precedence constraints between tasks) or independent; machines can be homogeneous (all the machines take the same time to execute the same task) or heterogeneous; and the objective may be to reduce the makespan (completion time of all jobs), to reduce the task delay, or to reduce the execution time for each machine. This paper addresses the Unrelated Parallel Machine Scheduling Problem (UPMSP) with sequence dependent and setup times, denoted as $R|S_{ijk}|C_{max}$ in the $\alpha|\beta|\gamma$ notation introduced by [Graham et al., 1979].

The UPMSP is composed of a set of heterogeneous machines $\mathcal{M}$ and a set of jobs $\mathcal{N}$. Each job k has an execution time p_{ik} for each machine i. Every job must be assigned to a single machine. A setup time s_{ijk} incurs to initialize a job k after a job j in a machine i. All jobs are available at time 0 and there is no precedence constraints between jobs. The objective is to assign each job to a machine in such a way that the completion time of the last job (C_{max}) is minimum. As example, consider the following input data for UPMSP:

$$\mathcal{M} = \{m_1, m_2\} \qquad \mathcal{N} = \{n_1, n_2, n_3, n_4\}$$

$$p_{ik} = \begin{array}{c} \\ m_1 \\ m_2 \end{array} \begin{array}{c} \begin{array}{cccc} n_1 & n_2 & n_3 & n_4 \end{array} \\ \begin{bmatrix} 2 & \mathbf{2} & \mathbf{3} & 4 \\ \mathbf{3} & 3 & 5 & \mathbf{3} \end{bmatrix} \end{array} \qquad s_{m_1 jk} = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \end{array} \begin{array}{c} \begin{array}{cccc} n_1 & n_2 & n_3 & n_4 \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 1 \\ 2 & 0 & 1 & 2 \\ 3 & \mathbf{1} & 0 & 3 \\ 2 & 1 & 2 & 0 \end{bmatrix} \end{array} \qquad s_{m_2 jk} = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \end{array} \begin{array}{c} \begin{array}{cccc} n_1 & n_2 & n_3 & n_4 \end{array} \\ \begin{bmatrix} 0 & 3 & 2 & 1 \\ 2 & 0 & 3 & 2 \\ 3 & 2 & 0 & 3 \\ \mathbf{2} & 2 & 3 & 0 \end{bmatrix} \end{array}$$

Figure 1 presents an example of solution to this UPMSP having $C_{max} = 8$. Active values of the input data in the solution are highlighted in bold. Blank spaces represent the setup times between jobs.

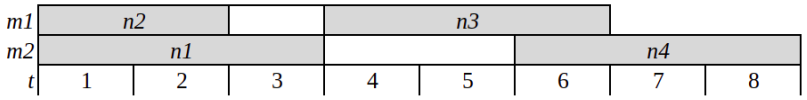


Figure 1: Example of solution for the given example of input data for UPMSP.

The UPMSP is present in several industrial applications. For instance, [Kim et al., 2002] presented an application of the UPMSP in the compound semi conductor of wafer industry. In this application it is very important to design efficient scheduling plans based on current production capacity rather than to purchase more capacity, given the high capital associated with typical semiconductor production equipment. Another example of industrial application of the UPMSP is presented by [Arnaout et al., 2009], which consists in the fabrication process of inks and plastics as an UPMSP. In this application, it is necessary to clean the machine between the end of a job and the beginning of the next one, therefore incurring setup times.

The UPMSP is a generalization of the Parallel Machine Scheduling Problem, and therefore classified as $\mathcal{NP}$-hard [Lenstra et al., 1977]. Due to the practical importance and to the hardness of solving, several solution approaches have been proposed for this problem. The early computational studies on the UPMSP started with [Guinet, 1993]'s mathematical programming formulation to plan the utilisation of textile machines in French industries. [Rabadi et al., 2006] extended the formulation devised by [Guinet, 1993] and proposed a metaheuristic approach based on the same

ideas applied to solve Travelling Salesman problems. [Vallada e Ruiz, 2011] introduced new variables to the formulation described in [Rabadi et al., 2006] and proposed a Genetic Algorithm to the problem. Moreover, they proposed a set of challenging instances for the UPMSP, which are still used as benchmark to evaluate solution approaches. Later, [Avalos-Rosales et al., 2015] presented some reformulations and valid inequalities that resulted in a very strong formulation to the UPMSP. [Fanjul-Peyro et al., 2019] proposed reformulations and an exact algorithm for the UPMSP, being able to obtain solutions for extremely large instances with relative deviations from lower bounds below 0.8%. More recently, [Santos et al., 2019] developed several Stochastic Local Search algorithms in a multi-neighbourhood structure for the UPMSP and achieved remarkable results even within very short processing time.

In the present paper, a matheuristic is proposed for the UPMSP. More specifically, a Fix-and-Optimize (Fix-Opt) approach is presented. Matheuristics are heuristic algorithms made by the cooperation between metaheuristics and mathematical programming methods [Raidl e Puchinger, 2008] [Maniezzo et al., 2010]. In the Fix-and-Optimize approach, a metaheuristic works at the master level, controlling low level local search procedures. These local searches are reduced Mixed Integer Programming (MIP) models, in which a subset of variables is fixed to their current values in the incumbent solution, and the remaining variables of the model can be freely modified by the MIP solver [Fonseca et al., 2016].

Fix-and-Optimize approaches are achieving remarkable results for timetabling and scheduling problems in the recent literature. For instance, [Fonseca et al., 2016] proposed a Fix-and-Optimize-based approach to the Educational Timetabling problem and improved the best known solution for 15 out of 17 open instances; and [Santos et al., 2016] proposed a Fix-and-Optimize algorithm to the Nurse Rostering problem that improved the best known solution of the vast majority of tackled open instances in up to 15%. Other successful examples of matheuristic approaches applied to timetabling and scheduling problems are presented in [Arbaoui et al., 2016] and [Lindahl et al., 2018].

To the best of the authors' knowledge, no Fix-and-Optimize approach has been proposed to the UPMSP yet. Therefore, this paper aims to propose and investigate the effectiveness of a problem-specific Fix-and-Optimize approach to the UPMSP. Benchmark instances from [Vallada e Ruiz, 2011] were considered to evaluate the algorithm. The conducted computational experiments confirmed that Fix-and-Optimize is a strong approach to solve Machine Scheduling problems. It achieved competitive results when compared to exact and heuristic state-of-the-art solution methods.

The remainder of this paper is organized as follows. Section 2 presents the adopted Integer Programming formulation for the problem. Section 3 presents the proposed Fix-and-Optimize approach for the UPMSP. Section 4 presents the conducted computational experiments and a discussion of these results. Finally, Section 5 presents some concluding remarks and directions for future research on the topic.

2. Integer Programming Formulation for UPMSP

This section reproduces the integer programming formulation proposed by [Avalos-Rosales et al., 2015] to the UPMSP. It is an extension of the formulation developed by [Vallada e Ruiz, 2011] including some valid inequalities and reformulations. This formulation considerably outperforms the previously published formulations regarding size of instances and computational time to reach optimal solutions. Thereby, it was adopted in this work. It takes as input the following sets and parameters:

Input	Description
$\mathcal{N}$	Set of Jobs.
$\mathcal{N}_0$	Set of Jobs including a dummy job (0 index).
$\mathcal{M}$	Set of Machines.
p_{ik}	Execution time for job j on machine k.
s_{ijk}	Matrix with the setup time for jobs i and j in machine k.
$\mathbb{U}$	Valid upper bound for makespan.

and the following variables, along with their domains:

Variables	Description
$x_{ijk} \in \{0,1\}$	1, if job j is scheduled in sequence of job k in machine i; 0, otherwise.
$C_j \geq 0$	Completion time of job j.
$C_{max} \geq 0$	Maximum completion time among all jobs (makespan).
$y_{ik} \in \{0,1\}$	1, if job k is assigned to machine i; 0 otherwise.

Dummy jobs, denoted by 0, are needed to represent the beginning and the end of activities for each machine. Thus, x_{i0j} represents that j is the first job in machine i, and x_{ik0} represents that k is the last job in machine i. If a machine i is not used, x_{i00} equals 1. The problem can be stated as:

$$\textbf{min} \quad C_{max} \tag{1}$$

$$\textbf{s.t.} \quad \sum_{i \in \mathcal{M}} y_{ik} = 1 \qquad \forall k \in \mathcal{N} \tag{2}$$

$$y_{ik} = \sum_{j \in \mathcal{N}_0, j \neq k} x_{ijk} \qquad \forall k \in \mathcal{N}, \forall i \in \mathcal{M} \tag{3}$$

$$y_{ij} = \sum_{k \in \mathcal{N}_0, k \neq j} x_{ijk} \qquad \forall j \in \mathcal{N}, \forall i \in \mathcal{M} \tag{4}$$

$$\sum_{k \in \mathcal{N}} x_{i0k} \leq 1 \qquad \forall i \in \mathcal{M} \tag{5}$$

$$C_k - C_j + \mathbb{U}(1 - x_{ijk}) \geq s_{ijk} + p_{ik} \qquad \forall j \in \mathcal{N}, \forall k \in \mathcal{N}, j \neq k, \forall i \in \mathcal{M} \tag{6}$$

$$C_0 = 0 \tag{7}$$

$$C_j \leq C_{max} \qquad \forall j \in \mathcal{N} \tag{8}$$

$$\sum_{j \in \mathcal{N}_0, j \neq k} \sum_{k \in \mathcal{N}} s_{ijk} \times x_{ijk} + \sum_{k \in \mathcal{N}} p_{ik} \times y_{ik} \leq C_{max} \qquad \forall i \in \mathcal{M} \tag{9}$$

Objective (1) minimizes the makespan. Constraints (2) ensure that each job is assigned exactly to one machine. Constraints (3) establish that every job has exactly one predecessor and both are assigned to the same machine. Constraints (4) guarantee that every job has exactly one successor and both are assigned to the same machine. Constraints (5) ensure that at most one job is scheduled as the first job on each machine. Constraints (6) are sub-tour elimination constraints which guarantee a right processing order. Constraints (7) sets the completion time of the dummy job to 0. Constraints (8) linearize the objective function. Finally, Constraints (9) are valid inequalities which strengthen the formulation (see [Avalos-Rosales et al., 2015]).

3. Proposed Approach

A Fix-and-Optimize approach is proposed to solve the UPMSP. At first, an initial solution is generated by a simple greedy algorithm. Then, a Fix-and-Optimize algorithm runs until the

available time limit is reached. The next subsections describe these algorithms.

3.1. Generation of Initial Solutions

Although the proposed approach does not rely on good quality initial solutions, they can speed up the search process because smaller models are created when the incumbent solution has a short makespan. Therefore, rather than generating a random solution we propose a simple greedy algorithm to build initial solutions for Fix-and-Optimize. In this constructive procedure, at each iteration one job is randomly selected. Then, it is assigned to the machine that have the so far shortest makespan in the position that yields the smallest makespan increase for that machine. The algorithm finishes when all jobs have been processed.

3.2. Fix-and-Optimize Matheuristic

Given an initial incumbent solution $\mathbb{S}$, the proposed Fix-and-Optimize (Fix-Opt) algorithm selects, at each iteration, a set of jobs $\mathcal{N}' \subseteq \mathcal{N}$ to have their assignment and ordering optimized by a mathematical programming solver. The remainder of the jobs $\mathcal{N} - \mathcal{N}'$ are fixed to the same machine and execution order as in solution $\mathbb{S}$. In fact, rather than directly selecting jobs to be optimized, we select a machine and then its jobs to be optimized. This is done because it gives more and better options for the solver to reorder and reassign the jobs. The makespan machine always have its jobs selected to be optimized since it is the one that can directly improve the objective value of a solution. To allow exchanges in job-machine assignment, at least one additional machine is selected. Note that additional machines are selected according to a roulette wheel which prioritizes machines with smallest makespan:

$$p_i = 1 - \frac{C^M_{max} - C^M_i}{C^M_{max}}, \forall i \in \mathcal{M} \tag{10}$$

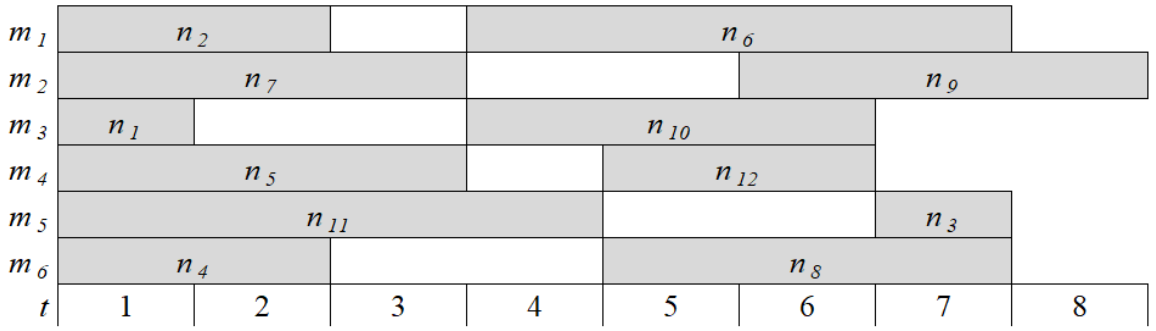
where C^M_i represents the makespan of machine i. Note that, for each machine, $p_i \in [0,1]$. These values are later normalized in such a way that the sum of the odds for all machines is 1.

The number of jobs to be freed (n) plays an important role in the proposed Fix-and-Optimize approach since it guides the size of the sub-problems. Too small values for n leads to a poor exploration of the search space, whilst too large values for n leads to excessively hard sub-problems that cannot be solved in reasonable time. Moreover, different values for this parameter can be more interesting at specific stages of the search. For example, when you already have a high quality solution it is less likely that a small n will lead to a better solution. Therefore, a large n might be more effective in such a case. To cope with that, we propose an auto-adaptive procedure that automatically adjusts the size of the sub-problem (n) throughout the optimization process.

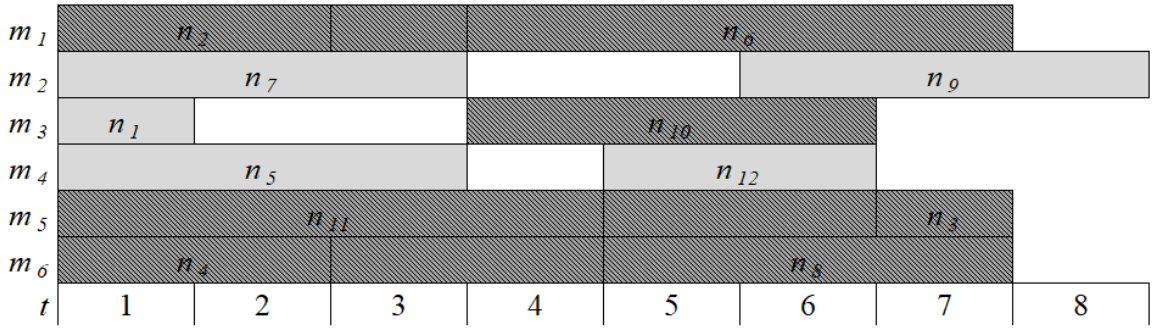
Another important feature of the proposed approach is that it allows a subset of jobs of the last selected machine $j \in \mathcal{M}$ to be released when the selection of all jobs from j surpasses sub-problem maximum size n. This is an important feature because, often, selecting the jobs of m machines to be optimized leads to a way too easy sub-problem, while the release of jobs of $m+1$ machines leads to a way too hard sub-problem. In such a case, jobs are selected randomly from machine j to be freed until n jobs in total are selected.

Figure 2 presents an example of the proposed solution approach for a small problem. Figure 2(a) represents the current solution $\mathbb{S}$. Suppose $n = 5$ jobs to be freed at this iteration. Machine m_2 is the makespan machine, therefore its jobs must be freed. Yet suppose that machine m_4 was chosen in sequence. The sum of jobs from machines m_2 and m_4 is still less than n, thereby a third machine, say m_3, is selected. Now considering m_2, m_4 and m_3 we have 6 jobs to release. In this case, only one job from m_3 (say n_1) will be released. Figure 2(b) presents the correspondent sub-problem, where diagonally dashed cells represent fixed jobs. Figure 2(c) presents the new solution

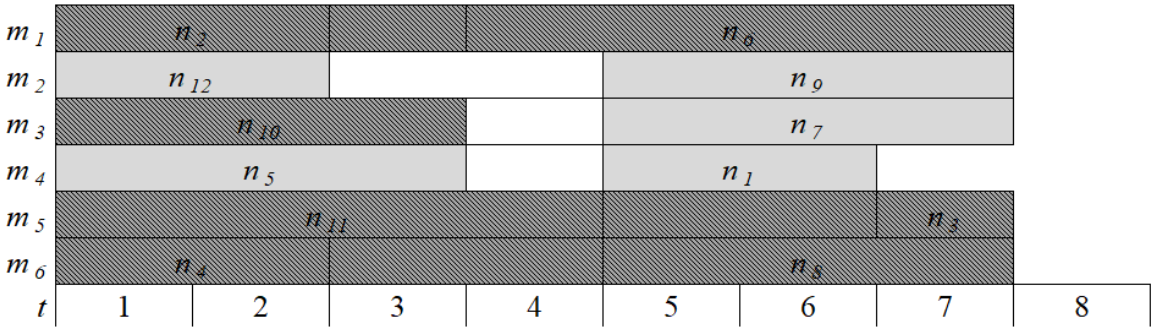
obtained after a MIP solver runs on the sub-problem of Figure 2(b). In this new solution, job n_1 has been moved to machine m_4, job n_{12} has been moved to m_2, and job n_7 has been moved to m_3, after n_{10}. Note that n_{10} is fixed at machine m_3 but can be reordered by the solver if needed.



(a) Current solution $\mathbb{S}$.



(b) Sub-problem fixing machines m_1, m_5, m_6 and job n_{10} of m_3.



(c) Optimal solution $\mathbb{S}'$ for sub-problem of Figure 2(b).

Figure 2: Example of an iteration of the proposed Fix-and-Optimize approach.

The incumbent solution for the next iteration has a special characteristic: there is a tie among the makespan machines m_1, m_2, m_3, m_5 e m_6. On the one hand, if we leave any of these machines fixed, the objective value cannot be improved. On the other hand, if we release all of them (for a bigger problem) the sub-problem may be too hard to be solved. Observe that even if the objective value cannot be improved in this iteration, a more promising solution can be obtained if the makespan of one (or more) of the machines is reduced, making it easier to improve the objective value in the next iterations. With that in mind, we load, at each iteration, only the sub-problem to

be solved in the solver rather than loading all the machines (and their makespans).

Algorithm 1 summarizes the proposed Fix-and-Optimize approach. It takes as input a problem instance $\mathbb{P}$ containing the input data; an initial solution $\mathbb{S}$; an initial number of jobs to be freed; an execution time-limit (in seconds); and a time-limit (in seconds) to solve each sub-problem. As output, it provides the best solution found $\mathbb{S}$. To simplify, notation $\mathbb{S}_i$ represents the set of jobs assigned to machine i in solution $\mathbb{S}$.

Algorithm 1: Fix-Opt: Proposed Fix-and-Optimize approach to the UPMSP.

Input: (i) Problem instance $\mathbb{P}$ ($\mathcal{N}$, $\mathcal{M}$, p_{ik}, s_{ijk}); (ii) Initial solution $\mathbb{S}$; (iii) Execution *timelimit*; (iv) Sub-problem size n; (v) Sub-problem time limit t.
Output: (i) Updated solution $\mathbb{S}$.

1 **while** *timelimit not reached* **do**
2 $i^{max} \leftarrow$ Makespan machine in $\mathbb{S}$;
3 $\mathcal{M}' \leftarrow \{i^{max}\}$; ▷ Machines selected to compose sub-problem $\mathbb{P}'$
4 $\mathcal{N}' \leftarrow \{\mathbb{S}_{i^{max}}\}$; ▷ Jobs selected to compose sub-problem $\mathbb{P}'$
5 **while** $|\mathcal{N}'| \leq n$ **do** ▷ Run until n jobs are selected to the sub-problem $\mathbb{P}'$
6 $i \leftarrow$ select a machine from $\mathcal{M} - \mathcal{M}'$ according to a roulette wheel (Eq. 10);
7 $\mathcal{M}' \leftarrow \mathcal{M}' \cup \{i\}$; ▷ Adds selected machine i to $\mathcal{M}'$
8 **if** $|\mathcal{N}'| + |\mathbb{S}_i| \leq n$ **then**
9 $\mathcal{N}' \leftarrow \mathcal{N}' \cup \{\mathbb{S}_i\}$; ▷ Add the jobs from selected machine i to $\mathcal{N}'$
10 **else**
11 **while** $|\mathcal{N}'| \leq n$ **do**
12 $j \leftarrow$ select a random job from those assigned to machine i;
13 $\mathcal{N}' \leftarrow \mathcal{N}' \cup \{j\}$; ▷ Add selected job j to $\mathcal{N}'$
14 $\mathbb{M} \leftarrow$ Create MIP model for sub-problem $\mathbb{P}'(\mathcal{N}', \mathcal{M}', p_{ik}, s_{ijk})$;
15 Load solution $\mathbb{S}$ into model $\mathbb{M}$;
16 $(\mathbb{S}, status) \leftarrow$ Solve $\mathbb{M}$ with a time limit t;
17 **if** *status is Optimal* **then**
18 $n \leftarrow \lceil n \times 1.2 \rceil$; ▷ Increase sub-problem size by 20%
19 **else**
20 $n \leftarrow \lfloor n \times 0.8 \rfloor$; ▷ Decrease sub-problem size by 20%
21 **return** $\mathbb{S}$;

Line 1 is the main loop that runs until the given *timelimit* is reached. Lines 2 and 3 select the makespan machine and adds it to the set of machines $\mathcal{M}'$ to be optimized in the sub-problem $\mathbb{P}'$. Line 4 adds the jobs from makespan machine i^{max} to the set of jobs $\mathcal{N}'$ to be optimized in the sub-problem $\mathbb{P}'$. Loop at lines 5 to 14 is executed until n jobs are selected to compose the sub-problem. Line 6 selects another machine i not selected yet by a roulette wheel (Eq. 10) and line 7 adds machine i to set $\mathcal{M}'$. If the addition of the jobs from machine i does not surpass the number of jobs to be selected n (line 8), all the jobs assigned to machine i will be selected (line 9); otherwise, a loop will randomly select jobs from i until n jobs are selected (lines 11 to 14). Line 15 creates a mixed integer programming (MIP) model $\mathbb{M}$ for sub-problem $\mathbb{P}'$ and line 16 loads solution $\mathbb{S}$ into model $\mathbb{M}$. Line 17 calls an IP solver with default parameters to solve $\mathbb{M}$ within a time limit of t seconds. Lines 18 to 21 adjust the sub-problem size. If the model $\mathbb{M}$ is solved to optimality within

t seconds, then sub-problem size n is increased by 20% (line 19); otherwise, it is decreased by 20% (line 21). Finally, line 22 returns updated solution $\mathbb{S}$.

4. Computational Experiments

Experiments were performed on a Dell G7-7588 A30P notebook, Intel Core i7-8750H (cache 9M, 2.20 GHz up to 4,10 GHz) processor, 16 GB RAM memory under Ubuntu 20.04 LTS operational system. The software was coded in Python 3.8 using Python-MIP 1.11.0. State-of-the-art solver Gurobi 9.0, with default parameter setting was used to solve the MIP models. In the spirit of reproducible science, the source code, problem instances and solution files are available online[1].

4.1. Instance Characterization

We adopted the instances proposed by Vallada and Ruiz [Vallada e Ruiz, 2011] to perform computational experiments. Only the set of large instances was considered because the small ones are not challenging for state-of-the-art solvers. The considered instances have a variate number of jobs to be scheduled $|\mathcal{N}| = \{50, 100, 150, 200, 250\}$ and, for each number of jobs, datasets having $|\mathcal{M}| = \{10, 15, 20, 25, 30\}$ available machines. For each combination of $|\mathcal{N}|$ and $|\mathcal{M}|$, ten instances were generated with random values for setup and preparation times. Due to time limitations to perform experiments, we considered only one randomly selected instance for each $|\mathcal{N}|$ - $|\mathcal{M}|$ combination, yielding 25 problem instances to be solved.

4.2. Parameter Calibration

The proposed approach has two parameters to calibrate: the initial size of the sub-problems n and the time limit for each sub-problem t. We believe the latter has a larger impact on the results because the former is auto-adapted throughout the optimization process. Thus, $n = 33$ was empirically defined and t was adjusted by the iRace package [López-Ibáñez et al., 2016]. Interval $n = [10, 120]$ and 500 runs of budget were considered for calibration. To avoid over-fitting in parameter selection, different instances were chosen to calibrate the solver. Finally, $t = 22$ was suggested by iRace and adopted in the conducted experiments.

4.3. Results and Discussion

Table 1 compares the obtained results by the proposed approach (Fix-Opt) with three state-of-the-art approaches in literature. IP stands for the Integer Programming formulation proposed by Avalós-Rosales *et al.* (2015) [Avalos-Rosales et al., 2015] and adopted to solve the sub-problems; MPA stands for the Mathematical programming algorithm proposed by Fanjul-Peyro *et al.* (2019) [Fanjul-Peyro et al., 2019]; and SLS stands for the Stochastic Local Search proposed by Santos *et al.* (2019) [Santos et al., 2019]. To provide a fair comparison, all solvers were executed in the same machine with a $3,600$s time limit. To compute the results, exact methods (IP and MPA) were executed once for each instance, whilst heuristic approaches (SLS and Fix-Opt) were executed five times. In the former case the obtained bounds are reported, whilst in later case the best and the average solution found are reported. For each instance the best solution found is highlighted in bold.

From the analysis of Table 1, we conclude that the proposed Fix-Opt approach is considerably better than its stand-alone integer programming model (IP) and the mathematical programming algorithm (MPA) to provide high quality solutions. When compared to SLS, there are instances in which Fix-Opt is better and others where SLS is better. Overall, the average of the best solutions found by these heuristic solvers are really close. This is an interesting result since SLS is a really strong solver for the UPMSP that generated 901 new best know solutions for the $1,000$ benchmark instances proposed by [Vallada e Ruiz, 2011]. Another interesting observation is that MPA considerably outperformed [Avalos-Rosales et al., 2015]'s IP model, which was believed to be the best exact approach for the UPMSP instances considered so far.

[1] https://github.com/Baumgratz/math-heuristic_to_upmsp

Instance	IP [Avalos-Rosales et al., 2015]		MPA [Fanjul-Peyro et al., 2019]		SLS [Santos et al., 2019]		Fix-Opt	
	$\mathcal{LB}$	Best ($\mathcal{UB}$)	$\mathcal{LB}$	Best ($\mathcal{UB}$)	Best	Avg.	Best	Avg.
I_50_10_S_1-124_5	100	111	102	143	111	113.00	**110**	112.00
I_50_15_S_1-99_4	51	67	51	90	58	58.00	58	58.80
I_50_20_S_1-49_5	29	29	28	31	29	29.80	29	29.00
I_50_25_S_1-9_4	20	20	20	20	20	20.00	20	20.00
I_50_30_S_1-9_4	16	16	16	16	16	16.00	16	16.00
I_100_10_S_1-124_5	186	222	186	235	203	206.40	**202**	205.80
I_100_15_S_1-49_4	68	83	69	83	**76**	76.60	77	78.80
I_100_20_S_1-124_6	68	82	63	108	74	75.20	74	77.20
I_100_25_S_1-124_8	42	73	41	85	**52**	53.20	55	55.40
I_100_30_S_1-9_8	21	26	21	24	23	23.00	23	23.00
I_150_10_S_1-99_9	223	269	224	247	246	246.60	**235**	236.80
I_150_15_S_1-49_6	100	139	100	120	110	110.40	110	112.00
I_150_20_S_1-9_1	61	69	61	65	**63**	63.80	64	65.00
I_150_25_S_1-124_7	68	278	67	138	**82**	83.20	89	91.40
I_150_30_S_1-9_1	28	35	28	31	**30**	30.20	31	31.00
I_200_10_S_1-49_6	238	259	239	245	252	253.00	**246**	246.00
I_200_15_S_1-99_1	158	248	158	192	180	181.00	**175**	176.40
I_200_20_S_1-124_2	114	166	114	170	**134**	137.60	141	143.20
I_200_25_S_1-49_5	67	164	67	90	**76**	76.80	80	81.20
I_200_30_S_1-124_2	62	150	62	118	**76**	76.40	85	90.40
I_250_10_S_1-124_9	352	548	351	373	395	400.40	**369**	371.60
I_250_15_S_1-9_8	125	145	125	129	129	129.00	**128**	129.00
I_250_20_S_1-49_9	105	147	105	119	117	118.00	117	118.80
I_250_25_S_1-124_8	103	221	103	154	**124**	124.80	133	136.00
I_250_30_S_1-49_1	58	19452	58	79	**67**	67.60	72	73.20
Average	98.52	920.76	98.36	124.2	109.72	110.8	**109.56**	111.12

Table 1: Comparison of obtained results from the proposed approach (Fix-Opt) and three state-of-the-art solvers (IP, MPA, and SLS).

5. Concluding Remarks

This paper presented a Fix-and-Optimize algorithm for the UPMSP. In the proposed approach each sub-problem consists of optimizing the assignment and processing order of a subset of the jobs $\mathcal{N}' \subseteq \mathcal{N}$ that are assigned to a subset of machines $\mathcal{M}' \subseteq \mathcal{M}$ in the incumbent solution. Computational experiments shown that the proposed approach outperforms two state-of-the-art exact approaches and achieve competitive results when compared to the literature's best performing heuristic method for the problem. An interesting feature of the proposed approach is that it is based on a model so it can be easily extended to other variants of machine scheduling problems, while purely heuristic methods are often tailored to the problem's specific features.

It should be noticed that there is still room for improvement in the proposed Fix-and-Optimize approach. As directions for future research we suggest: (i) implementing the MPA to solve the sub-problems instead of [Avalos-Rosales et al., 2015]'s formulation; (ii) evaluating the use of SLS as method to generate initial solutions to the proposed Fix-and-Optimize algorithm; and, finally, (iii) performing systematic experiments with more instances and larger time limits.

References

Arbaoui, T., Boufflet, J.-P., e Moukrim, A. (2016). A matheuristic for exam timetabling. *IFAC-PapersOnLine*, 49(12):1289 – 1294. ISSN 2405-8963. URL http://www.sciencedirect.com/science/article/pii/S2405896316309776. 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016.

Arnaout, J.-P., Rabadi, G., e Musa, R. (2009). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 21:693–701.

Avalos-Rosales, O., Angel-Bello, F., e Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 76 (9-12):1705–1718.

Fanjul-Peyro, L., Ruiz, R., e Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & Operations Research*, 101: 173–182.

Fonseca, G. H., Santos, H. G., e Carrano, E. G. (2016). Integrating matheuristics and metaheuristics for timetabling. *Computers & Operations Research*, 74:108 – 117. ISSN 0305-0548. URL http://www.sciencedirect.com/science/article/pii/S0305054816300879.

Graham, R., Lawler, E., Lenstra, J., e Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In Hammer, P., Johnson, E., e Korte, B., editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, p. 287 – 326. Elsevier. URL http://www.sciencedirect.com/science/article/pii/S016750600870356X.

Guinet, A. (1993). Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *International Journal of Production Research*, 31(7):1579–1594. URL https://doi.org/10.1080/00207549308956810.

Kim, D.-W., Kim, K.-H., Jang, W., e Chen, F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18 (3):223 – 231. ISSN 0736-5845. URL http://www.sciencedirect.com/science/article/pii/S0736584502000133. 11th International Conference on Flexible Automation and Intelligent Manufacturing.

Lenstra, J. K., Kan, A. R., e Brucker, P. (1977). Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, p. 343–362. Elsevier.

Lindahl, M., Sørensen, M., e Stidsen, T. R. (2018). A fix-and-optimize matheuristic for university timetabling. *Journal of Heuristics*, 24(4):645–665.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., e Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43 – 58. ISSN 2214-7160. URL http://www.sciencedirect.com/science/article/pii/S2214716015300270.

Maniezzo, V., Stützle, T., e Voß, S. (2010). *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*. Springer, 1st edition.

Rabadi, G., Moraga, R. J., e Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17(1):85–97.

Raidl, G. e Puchinger, J. (2008). Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In Blum, C., Aguilera, M. J. B., Roli, A., e Sampels, M., editors, *Hybrid Metaheuristics*, volume 114 of *Studies in Computational Intelligence*, p. 31–62. Springer Berlin Heidelberg. ISBN 978-3-540-78294-0.

Santos, H. G., Toffolo, T. A., Gomes, R. A., e Ribas, S. (2016). Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, 239(1):225–251.

Santos, H. G., Toffolo, T. A., Silva, C. L., e Vanden Berghe, G. (2019). Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem. *International Transactions in Operational Research*, 26(2):707–724.

Vallada, E. e Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211 (3):612–622.