Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem

Eva Vallada, Fulgencia Villa, Luis Fanjul-Peyro

Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain. Email: evallada@eio.upv.es, mfuvilju@eio.upv.es, luifanpe@upvnet.upv.es

Abstract

A Scatter Search algorithm together with an enriched Scatter Search and an enriched Iterated Greedy for the unrelated parallel machine problem with one additional resource are proposed in this paper. The optimisation objective is to minimise the maximum completion of the jobs on the machines, that is, the makespan. All the proposed methods start from the best known heuristic for the same problem. Non-feasible solutions are allowed in all the methods and a Repairing Mechanism is applied to obtain a feasible solution from a resource constraint point of view. All the proposed algorithms apply different local search procedures based on insertion, swap and restricted neighbourhoods. Computational experiments are carried out using an exhaustive benchmark of instances. After analysing the results, we can conclude that the enriched methods obtain superior results, outperforming the best known solutions for the same problem.

Keywords: Unrelated parallel machine, Scheduling, Additional scarce resource, Metaheuristics, Makespan.

1. Introduction and problem definition

Decision-making plays a significant role in industry. Nowadays, it is important to develop intelligent methods to solve decision-making problems using optimisation techniques, in order to get both efficient and effective results. All this should contribute to the smart factory concept, that is, sustainable and intelligent industries [1].

Problems related to scheduling involve a decision-making procedure that is vital in manufacturing industries. Scheduling problems, in general, consist of the assignment of a set of jobs to a set of machines and optimising one or more objectives. There are many types of scheduling problems (Pinedo, 2016) and one of the most studied in the literature is the parallel machine scheduling problem, where there is a set of n jobs which have to be processed on just one machine from a set of m machines. If the processing times of the jobs are different because of the characteristics of the machine which they are assigned to, the variant of the problem is known as the unrelated parallel machine scheduling problem (UPM). Some recent papers dealing with this problem are Fanjul-Peyro and Ruiz (2010), Fanjul-Peyro and Ruiz (2011), Vallada and Ruiz (2011), Rodriguez et al. (2013) and Arroyo and Leung (2017), among others. The variant where secondary resources are also considered is much less studied in the literature. This variant is known as the unrelated parallel machine scheduling problem with secondary resources (UPMR) and is a more realistic extension of the problem. Each job needs an amount of one scarce and renewable resource (human resources, tools, etc). Like processing times, the secondary resource consumption of a job is also different depending on the machine it is assigned to, as is usual in real environments, where different machines from a technical point of view are working at the same time. The secondary resource is considered renewable, that is, the job needs an amount of a secondary resource during its processing time and after that, the amount of resource is freed up. As a result, the problem consists of finding the best assignment and the best sequence for each machine so that the secondary resource constraint is satisfied at any time. With respect to the optimisation objective, the most studied of these problems is the minimisation of the maximum completion of the jobs, known as makespan and denoted as $C_{\max}$.

Some assumptions are usually made regarding this problem: each job is processed by exactly one machine, each machine can process only one job at a time, preemption of jobs is not allowed and each job needs an amount of the additional resource during its entire process. More specifically, the objective of the problem is to schedule a set of n jobs (indexed by j) on one machine selected from a set of m machines (indexed by i) so that the resource constraint

[1]`https://www.capgemini.com/resources/preparing-for-smart-factories/`

is always satisfied. Machines are also considered resources in this problem and in order to simplify we will refer to the secondary additional resource as "additional resource" or just "resources". Some notation is introduced as follows:

· p_{ij}: processing time of job j on machine i.

· r_{ij}: resource consumption of job j on machine i.

· $Mach_i$: list of jobs assigned to machine i and processed following the order of the list.

· $R_{\max}$: maximum availability of the additional resource.

· C_i: completion time of machine i.

· $C_{\max}$: maximum completion time, $max\{C_1, C_2, \cdots, C_i\}$.

· S: a feasible solution.

· A: an assignment of jobs to machines without considering the additional resource.

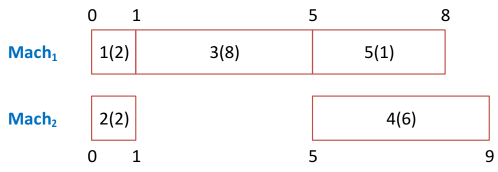
Example 1.1 illustrates the studied problem.

Regarding the complexity of the problem, the variant with identical parallel machines is already NP-complete even with one additional resource (Garey and Johnson, 1975). Also for identical machines, Błażewicz et al. (1987) showed that the problem is NP-hard in the strong sense, even with only 2 machines and one type of resource. It is clear that the complexity of the problem implies that mathematical models are only suitable for small instances and heuristic and metaheuristic algorithms are necessary to solve larger problems.

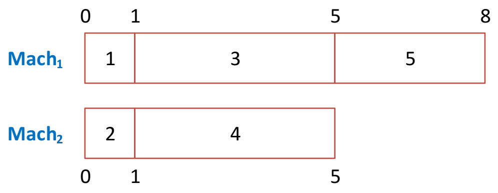
It is important to differentiate between a feasible solution (S) and just an assignment (A), which is very likely to be non-feasible. Example 1.1 illustrates this difference. Figure 1(a) shows a solution provided by a simple rule where the jobs are alternatively assigned to the machines (job 1 on machine 1, job 2 on machine 2, job 3 on machine 1 and so on). In order to obtain a feasible solution, an idle time on machine 2 is needed. In brackets we can see the resource consumption of each job on the machine. The total consumption at any time must be less than or equal to the maximum availability of the resource, that is, ten units (Figure 1(a)). Jobs start at time 0 and we can observe that, on machine 2, from time 1 to time 5, there is an idle time since jobs 3 and 4 can not be processed at the same time. Consequently, a solution implies the computation of the start and finish time for all the jobs and the satisfaction of the resource constraint at any time. In Figure 1(b), an assignment for the same example can be found. We can observe that an assignment is always compacted and resources are not considered. As a result, non-feasible solutions are very likely to be obtained from just an assignment. Therefore, the makespan value for Figure 1(a) is 9 (denoted as $C_{\max}(S) = 9$) while $C_{\max}(A) = 8$ (Figure 1(b)).

Example 1.1. *Consider the following instance of a UPMR with two machines ($m = 2$), five jobs ($n = 5$), ten units of a scarce resource ($R_{\max} = 10$) and the following processing times and resource needs:*

$$(p_{ij}) = \begin{pmatrix} 1 & 2 & 4 & 3 & 3 \\ 3 & 1 & 2 & 4 & 1 \end{pmatrix}; \quad (r_{ij}) = \begin{pmatrix} 2 & 6 & 8 & 7 & 1 \\ 8 & 2 & 3 & 6 & 2 \end{pmatrix}$$



(a) Feasible solution (S).



(b) Assignment (A).

Figure 1: Example of a Feasible Solution (S) and an Assignment (A) according to Example 1.1.

In this paper, effective and fast algorithms to optimise a real decision-making problem related to industry are proposed. A Scatter Search algorithm and two enriched metaheuristics are proposed. Specifically, Enriched Scatter Search and Enriched Iterated Greedy methods are presented for the defined problem, where a Restricted Local Search is applied in order to enrich the methods. The main objective and contribution of the paper is to improve on the best known results but without the need for a large amount of CPU time.

The rest of the paper is organised as follows: In Section 2, an overview of the literature is presented. In Section 3 details about the proposed enriched metaheuristics are provided. In Section 4 computational results are presented. Finally, in Section 5, some conclusions and future avenues of research are given.

2. Literature review

Scheduling problems have been widely studied in the literature over recent decades. In fact, unrelated parallel machine scheduling problems have been considered in much of the research. Some recent results are found in, Zeidi and MohammadHosseini (2015), Chen (2015) and Arroyo and Leung (2017), and in Kravchenko and Werner (2011) a review related to parallel machine problems is presented. However, the variant which considers additional resources has been much less investigated by researchers. Constraints related to resources are extremely important in real industrial environments, where there are several additional resources (human resources, tools, moulds, etc.) apart from the machines. There are some papers related to real manufacturing systems: Edis and Ozkarahan (2012) proposed an Integer Programming model and Constraint Programming for an injection-molding case with the objective of minimising makespan. Bitar et al. (2016) and Ventura and Kim (2003) presented methods for the semiconductor manufacturing case with different optimisation objectives. The first one minimised the weighted flow time and the second minimised the total absolute deviation of job completion times in respect to their due dates.

In Edis and Ozkarahan (2011) and Edis and Oguz (2012) mathematical models are proposed for different variants of the problem. In a very recent paper proposed by Fanjul-Peyro et al. (2017), mathematical models together with matheuristics (a combination of mathematical models and heuristics) are developed for the studied problem. Moreover, in Edis et al. (2013), we can find a review and classification of methods for the parallel machine scheduling problem with additional resources. The consideration of resources has been simplified in most of the papers. Among them, Ventura and Kim (2000) and Zheng and Wang (2016) studied one additional resource where jobs need one/zero units of resource. In other works (Daniels et al., 1999 and Ruiz-Torres et al., 2007), the amount of resource is a fixed value for each machine or the resource consumption of a job is independent from the machine (Bitar et al., 2016). From the literature review, we can conclude that there are only two papers related to the general version of the problem, that is, the unrelated parallel machine problem with one additional resource and both processing times and resource consumption dependent on the machine. The first one is that written by Fanjul-Peyro et al. (2017) and the second one, which is very recent, is presented by Villa et al. (2018), where several heuristics are proposed for the problem. These are able to solve large instances with very modest computational times. The objective of this paper, is to propose effective enriched metaheuristic methods, starting from the solution provided by the best known heuristic presented in Villa et al. (2018), so that they are able to improve on the results without needing a lot of computational time. Specifically, Enriched Scatter Search and Enriched Iterated Greedy algorithms are proposed. Both, Scatter Search and Iterated Greedy are widely used in scheduling problems. The main concepts of the Scatter Search were introduced by Glover (1977) and extended by Laguna and Martí (2003) and Martí (2006). Some recent works related to Scatter Search are Naderi and Ruiz (2014), Riahi et al. (2017) and Gonzalez et al. (2017), among others. Regarding the Iterated Greedy, it was originally proposed by Ruiz and Stützle (2007) for the permutation flowshop scheduling problem and adapted to several scheduling problems. Some results can be found in Fanjul-Peyro and Ruiz (2010), Vallada and Ruiz (2011) and Rodriguez et al. (2013).

3. Metaheuristics

In this section, two metaheuristic algorithms are proposed, the first one is an Enriched Scatter Search and the second one is an Enriched Iterated Greedy algorithm. Both methods start from an initial solution provided by an effective constructive heuristic. Source codes for all the methods are available upon request from the authors.

3.1. Initial solution

The initial solution for both metaheuristic methods is provided by the best heuristic method proposed by Villa et al. (2018). Specifically, it is a multipass heuristic denoted as M5. In Algorithm 1 a pseudocode for the M5 heuristic is shown and a brief explanation is given below. Additionally, bin files in order to run the heuristic are available with the paper and source code is also available upon request from the authors.

- **Step 1: Initial assignment, feasibility and Repairing Mechanism**. This first step consists of different parts, briefly explained in the following:

- **Step 1.1: Initial assignment.** Eight assignment rules are used, according to the order obtained from different rules which consider the processing times or the resource consumption. This is only an assignment of the jobs to the machines, denoted as A. Resource constraint is not considered so it is very likely that the assignment is non-feasible from the resource constraint point of view. Each assignment is analysed in order to check if it is feasible.

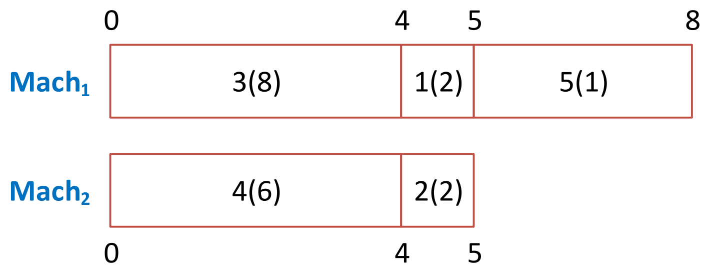


Figure 2: Checking Feasibility according to Example 1.1.

- **Step 1.2: Checking Feasibility and Repairing Mechanism.** Example 1.1 is used to illustrate how to check the feasibility of assignment A (Figure 1(b)). First, for each machine, jobs are ordered in non increasing order according to resource consumption (Figure 2). Second, we have to check if the first jobs for all the machines can be processed at the same time, that is, the total amount of resources needed is less than or equal to the maximum availability of resources (10 units in our example). In this case, job 3 (machine 1) and job 4 (machine 2) can not be processed at the same time since they need 14 units of resource and we only have 10 units. Then, in this example, the assignment is not a feasible solution and a mechanism to repair the solution is applied. In other cases, assignment A could be feasible and would result in a feasible solution. If this is the case, a local search procedure is applied (lines 4 to 6 in Algorithm 1). If assignment A is not feasible, a mechanism denoted as *Repairing Mechanism* is applied. It consists of removing jobs until a partial feasible solution is obtained. Jobs are removed one by one starting from the job with the maximum resource consumption. The removed jobs are denoted as Pending Jobs and they are reallocated following two strategies. For each pending job, following the first strategy, the job is scheduled at the end of the machine where the job was originally assigned. The second one schedules each pending job at the end of the machine so that the completion time of the machine is minimum. In both cases resource constraints have to be satisfied and finally the solution with the minimum makespan ($C^*_{\max}$) is selected. Once a pending job is scheduled at the end of a machine, the *Repairing Mechanism* swaps this pending job with the previous one if two conditions are satisfied: 1) there is no idle time between both jobs and 2) the resource consumption of the previous job is less than the resource consumption of the pending job. This swapping procedure is applied as many times as possible. According to Example 1.1 and Figure 2, job 3 is removed from the assignment since it is the one with the maximum resource consumption. Then, the remaining jobs on machine 1 are shifted left. In Figure 3 we can see this partial solution is feasible and job 3 is included in the Pending Jobs set. The next step is to insert job 3 in the partial solution. Following the first strategy, job 3 would be located at the end of machine 1 (Figure 4(a)). With the other strategy, job 3 would be located at the end of machine 2. Then, as the resource consumption of the previous job (job 2) is less than the resource consumption of job 3 and there is no idle time between the jobs, jobs 2 and 3 are swapped (Figure 4(b)).

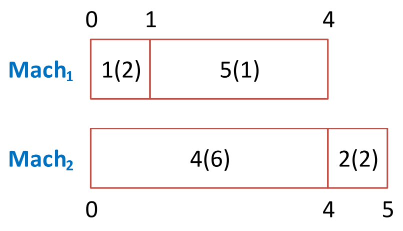
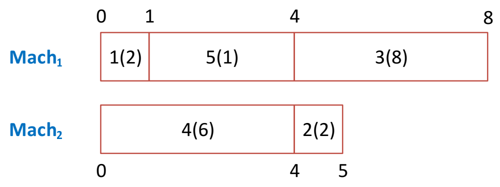
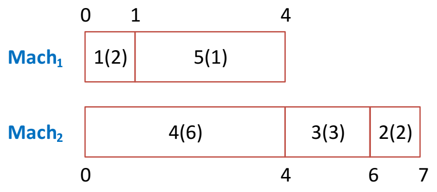


Figure 3: Partial feasible solution according to Example 1.1.

At the end of this first step, up to 8 solutions are obtained. We define the array S_A of size 8 to save each feasible solution for each assignment rule. Note that solutions with the same $C_{\max}$ are not allowed, so it would be possible to obtain fewer than 8 solutions in the S_A array if



(a) Inserting Pending Job 3 (Strategy 1).



(b) Inserting Pending Job 3 (Strategy 2).

Figure 4: Inserting Pending Jobs according to Example 1.1.

any of the assignment rules obtains the same $C_{\max}$ value as another one. The next steps are applied to each solution in a separate way.

- **Step 2: Non intensive local search.** This is based on insertion and swap neighbourhoods without considering resources and is very likely to obtain a non-feasible solution. At the end of each neighbourhood, the Repairing Mechanism is applied to obtain a feasible solution. According to Algorithm 1, we define the array S_{LS} (size 8) to save each feasible solution after the non intensive local search for each solution obtained from the previous step.

- **Step 3: Makespan machine unbalanced.** After the previous step, the local search procedure is exhausted. Machines might be too *balanced* and it is difficult to improve on the solution. Consequently, a procedure to *unbalance* the makespan machine is applied. The main idea is to insert all Pending Jobs into the makespan machine and to apply the non intensive local search again. Pending Jobs are generated from the assignment corresponding to the solution provided by the previous step (S_{LS}). The array to save each solution after this step is denoted as S_{UNB}.

- **Step 4: Intensive local search.** This is similar to the non intensive local search however, every time a movement is carried out in a neighbourhood, the Repairing Mechanism is applied to obtain a feasible solution. Every solution obtained is saved in the array denoted as S_{ILS}.

After the application of the M5 heuristic, we obtain a pool with up to 32 different solutions (up to 8 solutions for each explained step), saved in the

Algorithm 1: M5 heuristic.

```
1  for $a = 1, ..., 8$ (For each Assignment Rule) do
2      A:=Assignment Rule $a$;
3      Checking Feasibility($A$);
4      if Feasible(A) then
5          $S_A[a] := A$;
6          LocalSearch($S_A[a]$) #considering resources
7      else
8          $S_A[a]$:=Repairing Mechanism($A$)
9      $S_{LS}[a]$:=NonIntensiveLocalSearch($S_A[a]$);
10     $S_{UNB}[a]$:=Unbalance($C_{\max}$ machine,$S_{LS}[a]$);
11     $S_{ILS}[a]$ :=IntensiveLocalSearch($S_{UNB}[a]$);
```

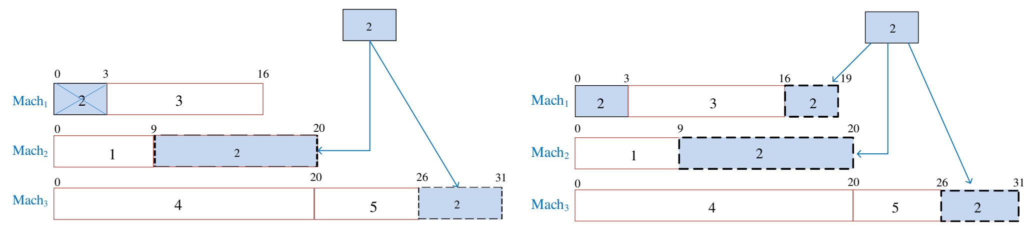
defined arrays. These solutions are used as an initial solution for the proposed metaheuristics.

3.2. Restricted Local Search (RLS)

Restricted Local Search (RLS) methods are based on restricted neighbourhoods with the objective of improving a solution using less computation time. Moreover, a greater number of different solutions can be obtained after the RLS procedures. These local search procedures can be successfully applied to parallel machine scheduling problems according to Fanjul-Peyro and Ruiz (2010). Two RLS neighbourhoods are applied based on the insertion of jobs. The RLS method starts from a feasible solution and it is improved on after several movements. In Algorithm 3 a pseudocode for RLS is provided. The first step is to obtain an assignment from the initial feasible solution, that is, the idle times due to the resource constraint are removed from the solution which obtains a compact but probably non-feasible schedule (see Example 1.1). Then, two RLS methods are applied. Pseudocode for both methods is provided in Algorithm 2.

- No Same Place (NSP): a job j is selected and placed on a machine i where $C_i + p_{ij}$ is minimum. Note that the job has to be placed on a different machine from the one to which the job was originally assigned. NSP is applied to one job from the $C_{\max}$ machine and one job randomly selected from the rest of the machines. After several tests, these values, already used in previous tests in the original paper, demonstrated the best performance.

- Virtual (VIR): a job j is selected and placed on a machine i where $C_i + p_{ij}$ is minimum. In this case, the job is also placed on the same machine to which the job was originally assigned, that is, the p_{ij} is considered twice. If after considering the processing time twice the job is assigned to the same machine, it means that the machine is the best one from a makespan point of view. VIR is applied to 5 jobs from the $C_{\max}$ machine and another 5 jobs randomly selected from the remaining ones. If there are fewer than 5 jobs on the $C_{\max}$ machine, all the jobs are selected. After several tests, those values used in the original paper performed the best.



(a) NSP example. (b) VIR example.

Figure 5: Example with 6 jobs and 3 machines for the RLS.

Note that in NSP, the job is removed and inserted into a different machine. In VIR, the job is not removed from the original machine until it is known to which machine it is going to be inserted. Moreover, that machine can be the same one as where the job was previously placed. In Figure 5 an example with 6 jobs and 3 machines is shown. Job 2 is originally assigned to Machine 1 and has to be moved to a different machine when NSP is applied (Machine 2 in this case). However, if the VIR procedure is applied, job 2 is also reinserted into Machine 1, and in this case after considering the processing time twice, Machine 1 is still the best option for job 2. In Algorithm 3 a pseudocode for the RLS procedure is given.

3.3. Enriched Scatter Search

Scatter Search (SS) is an evolutionary method widely used in scheduling problems. In this paper, an Enriched Scatter Search (ESS) is proposed. The

Algorithm 2: NSP-VIR.

```
1  A := Assignment provided;
2  d := number of jobs to remove;
3  for $j = 1, ..., d$ do
4      $i^* := C_{\max}$ machine;
5      $Mach_i^*$ := List of jobs of Machine $i^*$;
6      $k$ := Select at random one job from $Mach_i^*$;
7      if $NSP$ then
8          $w = \underset{s \in M - \{i^*\}}{\operatorname{argmin}} \{C_s + p_{sk}\}$
9      if $VIR$ then
10         $w = \underset{s \in M}{\operatorname{argmin}} \{C_s + p_{sk}\}$
11     Remove Job $k$ from Machine $i^*$;
12     Insert Job $k$ in Machine $w$;
13 for $j = 1, ..., d$ do
14     $i$ := Select at random one machine (except $i^*$);
15     $Mach_i$ := List of jobs of Machine $i$;
16     $k$ := Select at random one job from $Mach_i$;
17     if $NSP$ then
18         $w = \underset{s \in M - \{i\}}{\operatorname{argmin}} \{C_s + p_{sk}\}$
19     if $VIR$ then
20         $w = \underset{s \in M}{\operatorname{argmin}} \{C_s + p_{sk}\}$
21     Remove Job $k$ from Machine $i$;
22     Insert Job $k$ in Machine $w$;
```

Algorithm 3: Restricted Local Search (RLS).

```
1  $S_{best} := S$ ;
2  while Not TerminationCriterion do
3      $A :=$ Assignment from a feasible solution $S$ ;
4      NSP($A$);
5      $S$:=Repairing Mechanism ($A$);
6      Update $S_{best}$ ;
7      $A :=$ Assignment from the current solution $S$;
8      VIR($A$);
9      $S$:=Repairing Mechanism ($A$);
10     Update $S_{best}$ ;
11     NonIntensiveLocalSearch($S$) ;
12     Update $S_{best}$ ;
```

main steps of the ESS can be seen in Algorithm 4 where the initial population is provided by Algorithm 1. Lines 1 to 10 correspond to a standard Scatter Search. An additional step based on the RLS is applied to the best solution, obtaining the enriched version of the algorithm (ESS).

3.3.1. Building the reference set (RefSet)

According to Martí (2006), the *RefSet* consists of good and diverse solutions. Therefore, the best 5 solutions are selected from the initial population obtained in 3.1. These 5 solutions are removed from the initial population and included in the *RefSet*. Moreover, another 5 diverse solutions, different from the previous ones, are also selected. This diversification means that the 5 solutions are the most different in respect to the other ones from the *RefSet*. In order to compute the diversification value for each solution, the following steps are carried out: each solution from the initial population is compared to each solution from the *RefSet*, that is, a value is computed according to the number of times that a job is processed on the same machine in both solutions. The solution with the minimum value is selected to be in the *RefSet* and removed from the initial population. The *RefSet* is updated each time a new solution is included.
In Tables 1, 2 and 3 an example is provided to illustrate how diversity is considered. Table 1 shows the best 5 solutions from a hypothetical initial population to form part of the *RefSet*. The remaining 5 solutions should be

Algorithm 4: Enriched Scatter Search (ESS).

```
1  Generate Initial Population (Provided by Algorithm 1) ;
2  Generate $RefSet$ ;
3  $S_{best} := BestSolution(RefSet)$ ;
4  for $i = 1, ..., |RefSet|$ do
5      for $j = i + 1, ..., |RefSet|$ do
6          for $k = j + 1, ..., |RefSet|$ do
7              $S'$ = Combining($RefSet[i], RefSet[j], RefSet[k]$) ;
8              Checking Feasibility ($S'$) and Repairing Mechanism($S'$) ;
9              NonIntensiveLocalSearch($S'$) ;
10             Update $S_{best}$ ;
11 $S := S_{best}$ ;
12 while Not TerminationCriterion (100 iterations) do
13     $S$ :=RLS($S$) ;
14     Update $S_{best}$ ;
```

the most diverse in respect to the ones already in the *RefSet*. In order to simplify the example, we consider just two solutions to compute the diversity, but the process should be applied to the rest of solutions in the population. In Table 2, two candidate solutions to form part of the *RefSet* are shown. Diversity is computed according to how many times a job is located on the same machine that the best solutions of the *RefSet*. For example, in Solution 6 (S_6) job J1 is assigned to Machine 3. From Table 1, we can see that job J1 is located on Machine 3 for solutions S_4 and S_5. Then, the number of times job J1 is on the same machine is 2 (Table 3). The rest of the values are computed in the same way and the solution with the lowest total value is selected to form part of the *RefSet*. In this case, solution S_6. The *RefSet* is updated and the process is repeated until the 5 most diverse solutions are selected.

3.3.2. Combining and repairing the RefSet solutions. Local Search

Once the *RefSet* is built, the solutions are combined in order to obtain new ones. There are different strategies for combining solutions. After testing different combination methods based on selecting the machine from each solution with the minimum C_i, a voting procedure is applied according to

Solution	Machine 1	Machine 2	Machine 3
Solution 1 (S_1)	{J1, J2, J4}	{J6,J7}	{J5,J3}
Solution 2 (S_2)	{J1,J3}	{J5,J2}	{J6,J4}
Solution 3 (S_3)	{J2, J3, J4}	{J1}	{J5, J6, J7}
Solution 4 (S_4)	{J5,J6,J7}	{J2,J3}	{J1,J4}
Solution 5 (S_5)	{J5,J7}	{J6,J3, J4}	{J1,J2}

Table 1: Best solutions for the *RefSet*.

Solution	Machine 1	Machine 2	Machine 3
Solution 6 (S_6)	{J2, J6, J7}	{J5,J4}	{J1,J3}
Solution 7 (S_7)	{J3,J4,J5}	{J6,J7}	{J1,J2}

Table 2: Two candidate solutions for the *RefSet*.

Solution	J1	J2	J3	J4	J5	J6	J7	Total
Solution 6 (S_6)	2	2	1	1	1	1	2	10
Solution 7 (S_7)	2	1	2	2	2	2	1	12

Table 3: Diversity computation.

Alvarez-Valdes et al. (2006). This procedure is based on how many times a job is located on each machine. Specifically, three solutions from the *RefSet* are combined, that is, 3 solutions from a set of 10 solutions are selected which means 120 trios are combined in order to obtain a new solution. For each trio of solutions, there is a voting process where each job has a value according to how many times it is processed on each machine. The job is finally placed on the machine with the highest number of votes. Ties are broken following three strategies: placing the job on the machine in the solution with the minimum $C_{\max}$, placing the job on the machine with the minimum consumption of resources of the job and placing the job on the machine in the solution with the minimum C_i. None of the strategies outperform the remaining ones, so one of them is randomly selected in order to break the tie. The following example illustrates a small instance.

Example 3.1. *Consider the following instance of a UPMR with three machines ($m = 3$) and six jobs ($n = 6$), and the following assignment for each solution to combine (S_1, S_2 and S_3):*

Solution	Machine 1	Machine 2	Machine 3	$C_{\max}$
Solution 1 (S_1)	{J1}	{J3,J4}	{J2,J5, J6}	55
Solution 2 (S_2)	{J2,J3}	{J1,J5}	{J4,J6}	63
Solution 3 (S_3)	{J1, J2}	{J4,J6}	{J3,J5}	60

Table 4: Assignment of the jobs for three solutions.

According to Table 4, job J1 gets 2, 1 and 0 votes on machines 1, 2 and 3 respectively. Then, job J1 is finally placed on Machine 1. Following the same procedure, the final assignment is $Mach_1 = \{J1, J2\}$; $Mach_2 = \{J3, J4\}$; $Mach_3 = \{J5, J6\}$. Regarding job J3, there is a tie since job J3 gets one vote on each machine. Following the three strategies, we assume that the first one is selected randomly, that is, to place job J3 on the machine in the solution with the minimum $C_{\max}$. In this case, the minimum $C_{\max}$ is obtained by Solution S_1, then, job J3 is placed on Machine 2. After the combination, it is very likely that a non-feasible solution will be obtained so the *Repairing Mechanism* explained in 3.1 is applied. Once a feasible solution is obtained, it is improved upon by means of a local search procedure. A non intensive local search based on insertion and swap neighbourhoods without considering

resources, also explained in 3.1 is used. At the end of each neighbourhood, the *Repairing Mechanism* is applied to obtain a feasible solution.
Other strategies combining two solutions were tested which obtained worse results. For example, interchanging the $C_{\max}$ machine between both solutions (Alvarez-Valdes et al., 2015).
Once the basic Scatter Search is carried out, the Enriched version (ESS) consists of applying one hundred iterations of the RLS method (Algorithm 3) to the best solution. After one hundred iterations, the ESS can be compared to the Enriched Iterated Greedy in terms of CPU time (Section 4).

3.4. Enriched Iterated Greedy

The Iterated Greedy method is a simple and effective algorithm which has been successfully adapted to different scheduling problems, among them, unrelated parallel machine scheduling problems (Fanjul-Peyro and Ruiz (2010)). It is based on the partial destruction of a solution and the subsequent reconstruction together with a local search procedure. In this paper, an Enriched Iterated Greedy (EIG) algorithm is proposed where, apart from the main steps, several procedures focusing on the specific problem with resources are added to improve performance. The EIG algorithm starts from the best solution provided by Algorithm 1. In Algorithm 5 a pseudocode for the general structure of the EIG is provided. The Iterated Greedy (IG) method only corresponds to lines 9 to 15 and intensive work is carried out before starting the IG part. Note that although the authors have proposed in a previous paper (Fanjul-Peyro and Ruiz (2010)) an IG algorithm for the unrelated parallel machine scheduling problem with the makespan objective, this is only an assignment problem where there are no idle times and no resources. When the resource constraint is added, the problem is completely different and much more complex. In this case there are both an assignment problem, and at the same time, a scheduling (including timing) problem. This new feature implies that in some cases, when the resource constraint is not satisfied, idle times are needed in order to obtain a feasible solution. Consequently, even though both problems are related to parallel machines they are completely different. Therefore, it is clear that it is necessary to include new features in order to obtain an Enriched IG method. In this case, a basic version of the IG method does not perform well and for this reason it is not included in computational experiments (Section 4).

Algorithm 5: Enriched Iterated Greedy (EIG).

```
$S_{best} := S$; Initial solution ($S$) from heuristic M5 ;
while Not TerminationCriterion1 (10 iterations) do
    while Not TerminationCriterion2 (10 iterations) do
        $S_{RLS} := RLS(S)$ ;
    if $C_{\max}(S_{RLS}) < C_{\max}(S_{best})$ then
        IntensiveLocalSearch($S_{RLS}$);
        Update $S_{best}$ ;
    $S := S_{best}$ ;
    $S_{bestIG} := MaxInt$ (Large Number);
    while Not TerminationCriterion3 (10 iterations) do
        $A$ := Assignment from the current solution $S$;
        $A_p$ := Destruction($A$, $PJ$) ;
        $S_p$:= Checking Feasibility ($A_p$) and Repairing Mechanism($A_p$) ;
        $S$ := Construction($S_p$, $PJ$) ;
        NonIntensiveLocalSearch($S$) ;
        if $C_{\max}(S) < C_{\max}(S_{bestIG})$ then
            $S_{bestIG} := S$ ;
        Update $S_{best}$ ;
    IntensiveLocalSearch($S_{bestIG}$) ;
    Update $S_{best}$ ;
    $S := S_{best}$ ;
```

According to Algorithm 5, the EIG starts from the solution provided by the heuristic M5. This solution is improved upon by means of the Restricted Local Search (RLS) following Algorithm 3, in an iterative way, until a termination criterion is met. If the solution is improved, an intensive local search is applied in order to improve it further (line 6) and the best solution is updated. After all this previous work, the EIG method starts from the best solution obtained. In line 10, only the assignment of the solution is used in the destruction phase (line 11). During the destruction step, a partial assignment (A_p) is obtained after randomly removing 10% of the jobs on the makespan machine. The same number of jobs in total are randomly removed from the rest of the machines. Different values for the parameter number of the destructed jobs were tested with worse results. These removed jobs are denoted as Pending Jobs (PJ). Next, the partial assignment (A_p) is repaired in order to obtain a partial feasible solution (S_p) and in the construction phase, the Pending Jobs are inserted into the partial solution S_p. Pending Jobs are inserted following the strategies explained in Section 3.1. Note that the construction step is quite different from the original IG proposed by Ruiz and Stützle (2007), where each removed job is inserted into all the positions of all the machines and finally placed in the position where the makespan is minimum. Following Algorithm 5, after the construction process, a new feasible solution S is obtained and it is improved by applying the non intensive local search and finally the best solution is updated. After this, the intensive local search is applied to the best solution obtained during the process and the best solution is updated. This process is repeated until the termination criterion is met (line 2). Details about all the termination criteria are given in Section 4.

4. Computational Results

An exhaustive comparison of the proposed methods against the best known heuristic is carried out in this section. A benchmark of instances proposed by Fanjul-Peyro et al. (2017) is considered. There are three sets: small, with $n = \{8, 12, 16\}$, $m = \{2, 4, 6\}$, medium with $n = \{20, 25, 30\}$, $m = \{2, 4, 6\}$ and large $n = \{50, 150, 250, 350\}$, $m = \{10, 20, 30\}$. For each $n \times m$ combination, different sets according to processing times and resource consumption were generated. Uniform and correlated distributions are considered ($U(1, 100)$, $U(10, 100)$, $U(100, 200)$, Correlated Jobs and Correlated Machines) for processing times. Resource consumption is generated according to two distributions: uniform ($U(1, 9)$) and correlated values (the highest processing time and the highest resource consumption). The total number of the resource is computed as $R_{\max} = 5 \cdot m$. In total, five replicates for each combination were generated and there are 450 small instances, 450 medium instances and 600 large instances, available at `http://soa.iti.es`. Further details about the generation are in Fanjul-Peyro et al. (2017). The main objective is to get an exhaustive set of instances which consider several combinatorial levels. If most of the jobs require high levels of the secondary resource and the availability is low, the problem is easier to solve. Jobs can not be processed in parallel and there will be a lower combinatorial level. Additionally, the Repairing Mechanism only will assign the jobs at the end of the machine where the completion time is lower, in order to release the secondary resource as soon as possible. On the contrary, jobs with low levels of the secondary resource and a high level of availability means there are several options to process the jobs in parallel. If the number of machines is high, the combinatorial level will also be very high. In this case, the Repairing Mechanism will need more computation time when looking for the best location for each pending job (not only at the end of the machine). For this reason the number of instances is large and has several combinations. According the considered distributions of the consumption resource, it will be difficult to obtain lots of jobs according to these two extreme and non realistic cases.

All the experiments have been run on an Intel Core i7, 3.4 GHz and 20 GB RAM. Microsoft Visual Studio 2017 with C# has been used to code the different methods. Regarding the effectiveness measure, the Relative Percentage Deviation (RPD) is computed for each instance according to the usual expression:

$$\text{Relative Percentage Deviation}(RPD) = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \cdot 100, \qquad (1)$$

where Heu_{sol} is the solution obtained with a given proposed method and $Best_{sol}$ is the best known solution for a given instance. For small instances, $Best_{sol}$ is the optimal solution if it is known (Fanjul-Peyro et al. (2017)) and the best known solution when it is not. For medium and large instances, $Best_{sol}$ is the best known solution which can be optimal if the makespan is equal to the optimum makespan of the problem without resources (UPM). Note that if makespan values are the same for both problems, it does not mean that the order or assignment of the jobs on each machine is equal. Table 5 presents the number of optimal solutions found by the proposed methods.

The proposed methods (denoted as ESS and EIG) are compared against the best heuristic proposed in Villa et al. (2018), denoted as M5. A basic Scatter Search (denoted as SS) is also tested. Regarding the termination criterion, for the ESS method after the SS, 100 iterations of the RLS (Section 3.2) are applied. According to Algorithm 5 (EIG algorithm), there are three loops, so three termination criteria are needed (one per loop). For each loop, 10 iterations are run. In this way, the CPU times used by both enriched methods are quite similar.

Set/Method	M5	SS	ESS	EIG	ESS & EIG
Small	257	260	288	293	297
Medium	131	146	175	185	191
Large	51	53	66	75	80

Table 5: Number of proved optimal solutions obtained according to the size of the set of instances.

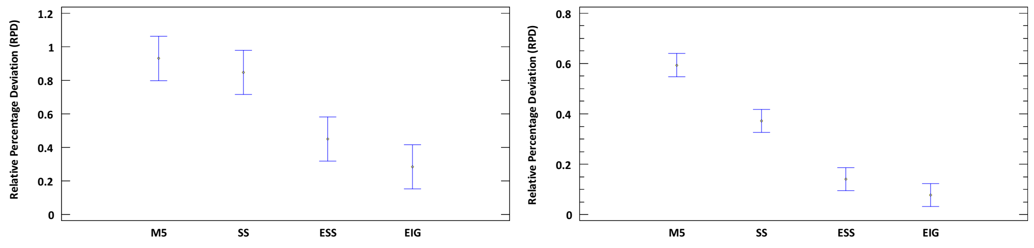
4.1. Small and medium instances

A first evaluation using small and medium instances is carried out. In Table 6 results for small instances are shown in terms of RPD and number of optimal solutions (in brackets). It seems there are differences between the average RPD among the methods. Specifically, the EIG algorithm obtains the lowest RPD. However, we apply a statistical analysis by means of MultiFactorial Analysis of the Variance (Montgomery (2012)) to check if the observed differences are statistically significant. The factors considered are: number of jobs, number of machines, p_{ij} set, r_{ij} set and algorithm. In Figure 6(a) the means plot with LSD intervals ($\alpha = 0.05$) is shown. Even when the M5 heuristic should be outperformed by the rest of the methods, it is necessary to check if the observed differences are statistically significant. For this reason M5 results are included in the analysis. Sometimes a method can obtain lower RPD values but the differences are due to chance and not because a method is performing better. According to Figure 6(a), two groups are observed, the first one includes the heuristic M5 and the SS and the second one includes the enriched methods (ESS and EIG). The second group performs better than the first one. Then, for small instances we can see that although SS obtains a lower RPD value, the statistical analysis shows that on average, SS and M5 methods perform equivalently (from a statistical point of view) and the observed differences are not statistically significant. Regarding the rest of the methods, there are no statistically significant differences between ESS and EIG since the intervals are overlapping. According to the previous explanation, ESS does not obtain better results than EIG for some small and medium instances. Even when the RPD values for ESS are slightly lower in one case (12×2 size), the statistical analysis shows that on average, the M5 and SS methods demonstrate equivalent performance (from a statistical point of view) and the observed differences are not statistically significant. For small instances, the EIG method is able to optimally solve 293 out of 450 instances and the ESS obtained 288 optimal solutions. In total, both methods obtain 297 optimal solutions, that is, there are 4 instances where the ESS method is able to obtain the optimal value and the EIG method is not. If we compare the SS method to the ESS, statistically significant differences are observed in terms of the average RPD and also in terms of the number of optimal solutions obtained (260 vs 288). This behaviour is also observed in the rest of the analysed factors. Average CPU times for each method are shown in Table 6 as well. The simplest methods (M5 and SS) need less than 1 second on average. The enriched methods (ESS and EIG) are also very fast, at around 5 seconds on average.

Instance group	M5	SS	ESS	EIG
8×2	0.24 (45)	0.17 (46)	**0.00** (50)	**0.00** (50)
8×4	1.17 (34)	1.17 (34)	0.66 (41)	**0.53** (41)
8×6	0.96 (45)	0.96 (45)	0.13 (48)	**0.04** (49)
12×2	0.55 (16)	0.49 (16)	**0.23** (18)	0.29 (17)
12×4	1.80 (26)	1.80 (26)	1.11 (29)	**0.79** (31)
12×6	1.05 (34)	0.99 (34)	0.62 (39)	**0.36** (40)
16×2	0.51 (13)	0.19 (13)	0.11 (13)	**0.02** (13)
16×4	0.81 (18)	0.63 (19)	0.31 (20)	**0.12** (22)
16×6	1.27 (26)	1.23 (27)	0.88 (30)	**0.43** (30)
Av.RPD	0.93	0.85	0.45	**0.29**
Av.Time (sec.)	0.01	0.05	5.5	5.5

Table 6: Average Relative Percentage Deviation (RPD), Average CPU Time in seconds and number of optimal solutions obtained by each method (in brackets) for small instances

In Table 7 results for medium instances can be found (number of optimal solutions found in brackets). Again, differences among the average RPD for all the methods are observed. In Figure 6(b) the means plot with LSD intervals is shown ($\alpha = 0.05$) where we can see that even though, on average, the RPD



(a) Small instances.

(b) Medium instances.

Figure 6: Means Plot and LSD intervals at the 95% confidence level for the algorithms (small and medium instances).

value for ESS is around 75% worse than the RPD of EIG, these differences are not statistically significant. Regarding the M5 heuristic and the SS method, we can state that on average the SS method performs better than the M5 heuristic. It is important to focus our attention on the average CPU times (Table 7). We can see that the simplest methods need much less time than the enriched ones. Specifically, the SS is able to obtain better solutions than the M5 heuristic using only 0.25 seconds on average. The enriched methods are also very fast, each one requiring 14 seconds on average. Therefore, we can state that the ESS and the EIG algorithms perform better than the other two methods needing just a few seconds on average. In Table 5 we can see that the ESS method obtains 175 proved optimal solutions out of 450 instances while the EIG gets 185. For both methods, the number of proved optimal solutions is 191, that is, there are 6 obtained by the ESS method that are not by the EIG. Therefore, it seems that both methods are working in different solution spaces. Differences can be seen between the M5 and the SS methods in respect to the number of proved optimal solutions, with the SS algorithm getting 15 more optimal solutions than the M5 heuristic. If we focus our attention on how many times an enriched method performs better than the other, the EIG performs better for 76 out of 450 instances, while the ESS performs better in 32 instances. Again, results are consistent regardless of the analysed factor.

4.2. Large instances

Experiments related to large instances are also carried out. In Table 8 we can see the average RPD value, the number of optimal solutions in brackets and the average CPU time for each method. Again, it seems there

Instance group	M5	SS	ESS	EIG
20×2	0.78 (6)	0.47 (6)	0.09 (6)	**0.03** (6)
20×4	0.42 (15)	0.31 (16)	0.12 (19)	**0.02** (22)
20×6	0.67 (22)	0.32 (25)	**0.12** (28)	0.18 (32)
25×2	0.47 (3)	0.37 (3)	0.13 (3)	**0.02** (3)
25×4	0.52 (19)	0.38 (20)	0.17 (23)	**0.16** (25)
25×6	0.78 (20)	0.43 (24)	**0.12** (29)	0.18 (28)
30×2	0.79 (9)	0.51 (11)	0.28 (14)	**0.07** (14)
30×4	0.48 (19)	0.24 (23)	0.12 (29)	**0.01** (30)
30×6	0.43 (18)	0.31 (18)	0.12 (24)	**0.04** (25)
Av.RPD	0.59	0.37	0.14	**0.08**
Av.Time (sec.)	0.05	0.25	14.5	14.5

Table 7: Average Relative Percentage Deviation (RPD), Average CPU Time in seconds and number of optimal solutions obtained by each method (in brackets) for medium instances.

are differences among the RPD values. After the statistical analysis by means of an ANOVA, Figure 7 shows the means plot with LSD intervals. In this case, we can state that the observed differences are statistically significant and the EIG demonstrates the best performance. There are also differences between the M5 heuristic and the SS method. Regarding the CPU times, the EIG algorithm needs around 1 minute on average and the ESS a little bit more. The simplest methods (M5 and SS) are much faster and need 20 and 51 seconds on average respectively. The EIG algorithm is the best method when considering the effectiveness and the amount of CPU time needed. In Table 5 we can see that the ESS obtains 66 proved optimal solutions out of 600 instances vs the EIG with 75. If we consider both methods together, 81 proved optimal solutions are obtained. Moreover, the EIG method performs better than the ESS algorithm in 196 instances. The ESS method performs better than the EIG in only 81 instances. In conclusion, it seems the performance of the EIG improves when the size of the instance increases. As for small and medium instances, results are consistent for all the analysed factors. As we can see in Figure 8 the EIG method performs the best regardless of the considered factor.

5. Conclusions and future research

A more realistic variant of the unrelated parallel machine problem considering one secondary additional resource has been studied. Several methods are proposed: a basic Scatter Search algorithm together with an enriched version (ESS) and an Enriched Iterated Greedy (EIG). Both enriched methods

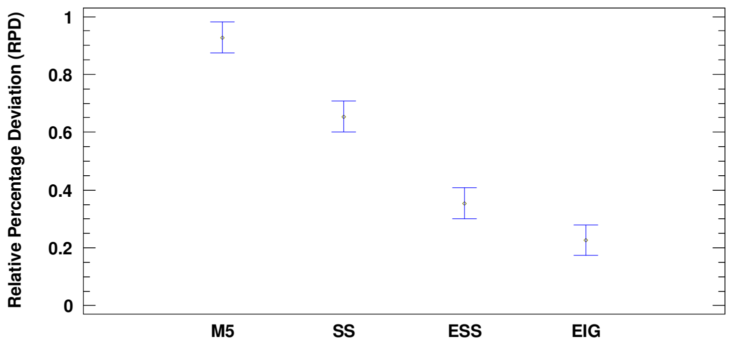
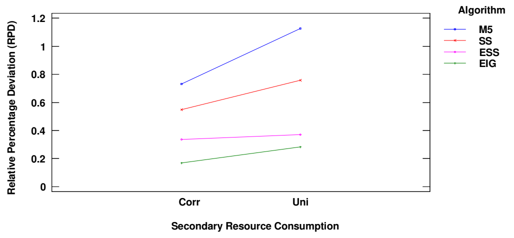
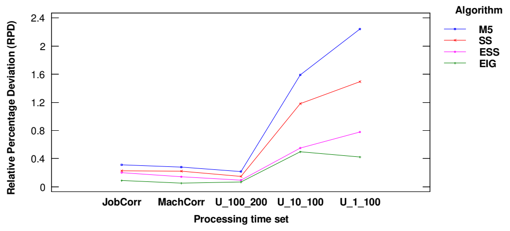


Figure 7: Means Plot and LSD intervals at the 95% confidence level for the algorithms (large instances)



(a) Large instances, interaction Algorithm and Resource Consumption Set.



(b) Large instances, interaction Algorithm and Processing Times Set.

Figure 8: Means Plot for the interaction between Algorithm and Resource Consumption Set and Algorithm and Processing Times Set (large instances).

Instance group	M5	SS	ESS	EIG
50 × 10	1.06 (13)	0.65 (13)	0.36 (15)	**0.27** (15)
50 × 20	1.36 (11)	1.21 (11)	0.48 (15)	**0.39** (15)
50 × 30	1.51 (19)	1.25 (19)	0.35 (24)	**0.34** (24)
150 × 10	0.72 (2)	0.44 (2)	0.30 (2)	**0.28** (4)
150 × 20	1.42 (1)	0.78 (3)	0.55 (3)	**0.27** (5)
150 × 30	1.23 (4)	0.93 (4)	0.45 (6)	**0.25** (7)
250 × 10	0.65 (0)	0.34 (0)	0.25 (0)	**0.15** (2)
250 × 20	0.81 (0)	0.55 (0)	0.40 (0)	**0.23** (1)
250 × 30	0.77 (0)	0.57 (0)	0.36 (0)	**0.18** (0)
350 × 10	0.48 (0)	0.26 (0)	0.23 (0)	**0.11** (0)
350 × 20	0.47 (0)	0.33 (0)	0.29 (0)	**0.15** (1)
350 × 30	0.65 (1)	0.52 (1)	0.24 (1)	**0.10** (1)
Av.RPD	0.93	0.65	0.35	**0.23**
Av.Time (sec.)	20	51.5	73.1	60.5

Table 8: Average Relative Percentage Deviation (RPD), Average CPU Time in seconds and number of optimal solutions obtained by each method (in brackets) for large instances.

use a Restricted Local Search (RLS) where the resource constraint is not considered and a Repairing Mechanism is used to obtain a feasible solution. The proposed methods are efficient and effective, which are essential requirements for decision-making in the smart factory concept. An exhaustive benchmark of instances has been used to compare the proposed algorithms against the best known method for the same problem (M5). For small and medium instances the enriched methods perform similarly and obtain better results. Regarding large instances, the EIG algorithm obtains the best results and outperforms the remaining methods. In conclusion, we can state that both enriched methods demonstrate good performance and on average only require a few seconds of CPU time. As both methods are simple and fast, they could be run in parallel and the best solution could be chosen. Regarding future research several extensions can be considered, for example the variant of the same problem with setup times and different optimisation criteria.

Acknowledgments

The authors are supported by the Spanish Ministry of Economy and Competitiveness, under the projects "SCHEYARD - Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) and "OPTEMAC - Optimización de Procesos en Terminales Marítimas de Contenedores" (No. DPI2014-53665-P), all of them partially financed with FEDER funds. The authors are also partially supported by the EU Horizon 2020 research and innovation programme under grant agreement no. 731932 "Transforming Transport: Big Data Value in Mobility and Logistics". Interested readers can download contents from `http://soa.iti.es`, like the instances used and a software for generating further instances. Source codes are available upon justified request from the authors.

Alvarez-Valdes, R., Crespo, E., Tamarit, J. M., Villa, F., 2006. A scatter search algorithm for project scheduling under partially renewable resources. Journal of Heuristics 12, 95–113.

Alvarez-Valdes, R., Tamarit, J. M., Villa, F., 2015. Minimizing weighted earliness-tardiness on parallel machines using hybrid metaheuristics. Computers and Operations Research 54, 1–11.

Arroyo, J. E. C., Leung, J. Y.-T., 2017. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. Computers and Operations Research 78, 117–128.

Bitar, A., Dauzère-Pérès, S., Yugma, C., Roussel, R., 2016. A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. Journal of Scheduling 19, 367–376.

Błażewicz, J., Kubiak, W., Röck, H., Szwarcfiter, J., 1987. Minimizing Mean Flow-Time with Parallel Processors and Resource Constraints. Acta Informatica 24, 513–524.

Chen, J., 2015. Unrelated parallel-machine scheduling to minimize total weighted completion time. Journal of Intelligent Manufacturing 26, 1099–1112.

Daniels, R. L., Hua, S. Y., Webster, S., 1999. Heuristics for parallel-machine flexible resource scheduling problems with unspecified job assignment. Computers and Operations Research 26, 143–155.

Edis, E. B., Oguz, C., 2012. Parallel machine scheduling with flexible resources. Computers and Industrial Engineering 63, 433–447.

Edis, E. B., Oguz, C., Ozkarahan, I., 2013. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. European Journal of Operational Research 230, 449–463.

Edis, E. B., Ozkarahan, I., 2011. A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. Engineering Optimization 43, 135–157.

Edis, E. B., Ozkarahan, I., 2012. Solution approaches for a real-life resource constrained parallel machine scheduling problem. International Journal of Advanced Manufacturing Technology 9, 1141–1153.

Fanjul-Peyro, L., Perea, F., Ruiz, R., 2017. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. European Journal of Operational Research 260, 482–493.

Fanjul-Peyro, L., Ruiz, R., 2010. Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research 207, 55–69.

Fanjul-Peyro, L., Ruiz, R., 2011. Size-reduction heuristics for the unrelated parallel machines scheduling problem. Computers and Operations Research 38, 301–309.

Garey, M., Johnson, D., 1975. Complexity results for multiprocessor scheduling under resource constraints. SIAM Journal on Computing 4, 397–411.

Glover, F., 1977. Heuristics for integer programming using surrogate constraints. Decision Sciences 8, 156–166.

Gonzalez, M. A., Palacios, J. J., Vela, C. R., Hernandez-Arauzo, A., 2017. Scatter search for minimizing weighted tardiness in a single machine scheduling with setups. Journal of Heuristics 23, 81–110.

Kravchenko, S., Werner, F., 2011. Parallel machine problems with equal processing times: A survey. Journal of Scheduling 14, 435–444.

Laguna, M., Martí, R., 2003. Scatter Search: Methodology and Implementations in C. Kluwer Academic Publishers, Boston, MA.

Martí, R., 2006. Scatter search - wellsprings and challenges. European Journal of Operational Research 169, 351–358.

Montgomery, D. C., 2012. Design and Analysis of Experiments, eigth Edition. John Wiley & Sons, New York.

Naderi, B., Ruiz, R., 2014. A scatter search algorithm for the distributed permutation flowshop scheduling problem. European Journal of Operational Research 239, 323–334.

Pinedo, M. L., 2016. Scheduling: theory, algorithms and systems, 5th Edition. Springer, New York,USA.

Riahi, V., Khorramizadeh, M., Newton, M., Sattar, A., 2017. Scatter search for mixed blocking flowshop scheduling. Expert Systems With Applications 79, 20–32.

Rodriguez, F. J., Lozano, M., Blum, C., García-Martínez, C., 2013. An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. Computers and Operations Research 40, 1829–1841.

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177, 2033–2049.

Ruiz-Torres, A. J., López, F. J., Ho, J. C., 2007. Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs. European Journal of Operational Research 179, 302–315.

Vallada, 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, 612–622.

Ventura, J. A., Kim, D., 2000. Parallel machine scheduling about an unrestricted due date and additional resource constraints. IIE Transactions 32, 147–153.

Ventura, J. A., Kim, D., 2003. Parallel machine scheduling with earliness-tardiness penalties and additional resource constraints. Computers and Operations Research 30, 1945–1958.

Villa, F., Vallada, E., Fanjul-Peyro, L., 2018. Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. Expert Systems With Applications 93, 28–38.

Zeidi, J., MohammadHosseini, S., 2015. Scheduling unrelated parallel machines with sequence-dependent setup times. International Journal of Advanced Manufacturing Technology 81, 1487–1496.

Zheng, X., Wang, L., 2016. A two-stage adaptive fruit fly optimization algorithm for unrelated parallel machine scheduling problem with additional resource constraints. Expert Systems with Applications 65, 28–39.