A Tabu Search for the Parallel Machines Job-Shop Scheduling Problem

Paper ID: —-

Tadeu Zubaran
Federal University of Rio Grande do Sul
Porto Alegre, RS - Brazil
tkzubaran@inf.ufrgs.br

Marcus Ritt
Federal University of Rio Grande do Sul
Porto Alegre, RS - Brazil
mrpritt@inf.ufrgs.br

ABSTRACT

In job-shop scheduling a set of jobs has to be scheduled on a set of machines, where each job needs to be processed on the machines in a given order. In the parallel machines job-shop scheduling problem, each machine is replaced by a stage, which consists of k identical parallel machines, and the jobs must be processed in each stage in a given order, but can choose any of the parallel machines in each stage.

In this paper we propose a tabu search for solving the parallel machines job-shop scheduling problem. Different from existing approaches, the tabu search works with solutions that are unaware of the parallel machines, and only the evaluation procedure generates a parallel schedule. This reduces the search space, and makes the heuristic simpler. The proposed tabu search improves existing algorithms significantly both in running time and solution quality.

KEYWORDS. Job shop scheduling. Parallel machines. Tabu search.

Metaheurísticas. Otimização combinatória.

1. Introduction

Shop scheduling models represent scheduling problems often occurring in industrial settings. They are abstract representation of such problems so that we ignore the details, but consider the important decisions that occur in the plant floor. In the *Job-Shop Scheduling Problem* (JSP) we have m machines $M_1, \ldots, M_m$, n jobs $J_1, \ldots, J_n$ and a set of nm operations $O = \{O_1, \ldots, O_{nm}\}$.

Each operation $o \in O$ is associated to a job $\iota(o)$, to which it belongs, and to a machine $\mu(o)$, where it will be executed. Each machine can process only one job at time and needs p_o units of time to finish the execution of the operation o. The jobs have to be processed without preemption, so once a machine begins to execute an operation it must finish it, and there are unlimited buffers for operations waiting for their machine to get available. Each job can be processed by only one machine at any instant, so no two operations of the same job can be processed at the same time. No job recirculation is allowed, which means that each job has exactly one operation associated with each machine. A job may not require a machine, which is modelled by an operation with zero processing time for that particular machine. In this paper we focus on the *makespan* objective function which is the time we complete the execution of the last operation.

The JSP is an NP-hard combinatorial optimization problem [10], and several efforts to develop exact and heuristic solvers for the JSP were proposed in the literature. A relatively small instance proposed by Muth and Thompson [18] with ten jobs and machines was not solved for more than twenty years. The branch-and-bound algorithm by Carlier and Pinson [8] is the first that managed to solve it optimally. Akers Jr [2] proposed a geometrical way to exactly solve the particular case of the JSP with two machines. There are many other exact algorithms for the JSP in the literature, mostly branch-and-bound algorithms. Some well known branch-and-bound algorithms for JSP are those by Applegate and Cook [3] and Brucker et al. [6]. More recently Gromicho et al. [15] proposed a dynamic programming exact algorithm to solve the JSP.

Exact solvers usually can not solve JSP instance with twenty or more jobs and machines in a timely manner, so many efforts were focused on developing heuristic solvers. Adams et al. [1] propose a constructive heuristic for the JSP called the Shifting Bottleneck Procedure which schedules the machines utilizing the branch-and-bound algorithm for a one-machine problem with release and holding times developed by Carlier [7]. Balas and Vazacopoulos [5] propose a guided local search that uses the Shifting Bottleneck Procedure, and propose a new neighbourhood for the JSP. Gonçalves and Resende [14] extended the graphical method of Akers Jr [2] to the general case of the JSP as an heuristic, since it no longer guarantees the optimal solution. The algorithm uses the graphical method with a biased random-key genetic algorithm. Nowicki and Smutnicki [19] is a seminal work which develops a Tabu search algorithm with a neighbourhood for the JSP. The neighbourhood was later named N5. Nowicki and Smutnicki [20] improved the algorithm, and Watson et al. [22] show that the N5 neighbourhood produces good results with other metaheuristics.

In order to better capture the intricacies of real applications several modifications on the base model of JSP were proposed. The partial job-shop imposes only a partial order on the operations of a job instead of a total order. A flexible job-shop introduces the possibility that an operation may be executed on more than one machine with different processing times depending on the chosen machine. The job-shop with process planning introduces more than one possible order for the job routing.

In this paper we address the *Job-Shop Scheduling Problem with Parallel Machines* (JSP-PM) introduces k parallel identical replicas of each machine. The JSP is NP-Hard in the strong sense Garey et al. [9] and since the JSP is a particular case of JSP-PM with $k = 1$, JSP-PM also is strongly NP-Hard. Unlike the JSP the literature on JSP-PM is scarce. Rossi and Boschi [21] provide a set of instances for the PSP-PM and propose a hybrid heuristic algorithm which combines a genetic algorithm and an ant colony optimization technique to solve the problem. Gholami and Sotskov [11] and Gholami and Sotskov [12] both propose similar models but the first one has an

associated release time for each job, while the second uses a different number of replications for each machine.

2. Formal definition of the problem

The JSP can be described in a *disjunctive graph* (DG) model. Let $G = (V, C \dot\cup D)$ be a graph with vertex set $V = O \cup \{0, *\}$ which consists of all operations O and two artificial vertices 0 and $*$. Vertex 0 represents the start of the processing, and vertex $*$ the end of the processing. The arc set is $C \dot\cup D$, partitioned into a set of *conjunctive arcs* C and a set of *disjunctive arcs* D. Conjunctive arcs model given precedence relations between operation, disjunctive arcs model alternatives in the processing order. We assume that the operations of job J_j are $O_{(j-1)m+1}, \ldots, O_{jm}$ given in order of their execution. Then for the JSP we have

$$\begin{aligned} C &= \{(O_o, O_{o+1}\} \mid o \in [(j-1)m+1, jm), j \in J\} \\ &\quad \cup \{(0, O_o) \mid o = (j-1)m+1, j \in J\} \cup \{(O_o, *) \mid o = jm, j \in J\}, \\ D &= \{(O_o, O_p\} \mid \mu(o) = \mu(p)\}. \end{aligned}$$

A solution to the JSP is described by a selection of disjunctive arcs $S \subseteq D$, such that S contains either (O_o, O_p) or (O_p, O_o) for each pair of operations with $\mu(O_o) = \mu(O_p)$, i.e. defines a total order on the machines, and such that the graph $G' = (V, C \cup S)$ is acyclic.

The disjunctive graph model leads naturally to a mathematical formulation of the JSP, where we introduce a binary variable $x_{op} \in \{0, 1\}$ for each pair of operations with $\mu(O_o) = \mu(O_p)$, such that $x_{op} = 1$ when operation O_o precedes operation o_p on their common machine. Introducing further starting times $y_o \in \mathbb{R}$ for each operation $o \in O$, and an auxiliary variable $C_{\max} \in \mathbb{R}$ representing the makespan, i.e. the completion time of the artificial operation $*$, we obtain the integer linear program

$$\begin{aligned}
\textbf{minimize} \quad & C_{\max} && && (1)\\
\textbf{subject to} \quad & C_{\max} \geq y_o + p_o, && \forall o \in O, && (2)\\
& y_p \geq y_o + p_o && \forall (O_o, O_p) \in C, && (3)\\
& y_o \geq y_p + p_p - M x_{op} && \forall (O_o, O_p) \in D, && (4)\\
& y_p \geq y_o + p_o - M(1 - x_{op}) && \forall (O_o, O_p) \in D, && (5)\\
& x_{op} \in \{0, 1\}, && \forall (O_o, O_p) \in D, && (6)\\
& y_o, C_{\max} \in \mathbb{R}, && \forall o \in O. && (7)
\end{aligned}$$

Here constraint (2) defines the maximum completion time, constraint (3) forces operations with fixed precedences to start after their predecessor terminated, constraints (4) and (5) defines the order of the disjunctive operations depending on the chosen order by the x variables. In this constraint M is large constant. It can be set to $M = \sum_{o \in O} p_o$, for example. Finally, (6) and (7) define the domains of the variables.

When we introduce parallel machines, $\mu(o)$ now only defines the stage on which the operation has to be executed. In this case we have to decide additionally to which machine an operation be assigned. For the case of k parallel machines for each of the m stages, we can introduce additional decision variables $s_{oi} \in \{0, 1\}$ such that $s_{oi} = 1$ when operation O_o is executed on the

	Operation							
	1st		2nd		3rd		4th	
Job	μ	p	μ	p	μ	p	μ	p
J_1	3	6	4	8	2	4	1	5
J_2	2	14	4	9	1	3	3	16
J_3	3	6	4	13	2	5	1	20

Table 1: Example of an instance of the JSP with three jobs and four machines.

ith parallel machine on its stage. This leads to the model

$$\textbf{minimize} \quad C_{\max} \tag{8}$$

$$\textbf{subject to} \quad \sum_{i\in[k]} s_{oi} = 1 \qquad \forall o \in O, \tag{9}$$

$$s_{oi} + s_{pi} \leq x_{op} + x_{po}, \qquad \forall (O_o, O_p) \in D, \tag{10}$$

$$y_p \geq y_o + p_o - M(3 - s_{oi} - s_{pi} - x_{op}) \qquad \forall (O_o, O_p) \in D, \tag{11}$$

$$s_{oi} \in \{0, 1\}, \qquad \forall o \in O, i \in [k], \tag{12}$$

$$\text{constraints (2),(3),(5),(6).} \tag{13}$$

In this model, the new constraints (9) make sure that each operation is assigned to a single machine on its stage, and constraint (10) forces precedences among operations on the same stage, when they have been assigned to the same machine. Note that, different from the model for the JSP, it is possible that $x_{op} = x_{po} = 0$ for some pairs of operations when they have been assigned to different machines. Constraint (11) forces non-overlapping execution of operations assigned to the same machine. Constraint (12) defines the domains of the new variables s_{oi}.

In this paper we are interested in JSP-PM instances which a fixed replication factor k. Such an instance is obtained from an instance of the JSP as follows. Each job J_j from the instance is replicated k times, which the same processing times and the same machine (resp. stage) order. Each machine of the JSP is converted into a stage with k parallel machines. Note that for these instances, a solution of the corresponding JSP leads to a solution of the JSP-PM, where the k replicated jobs execute in parallel in the k replicated machines. Thus, the optimal solution of such a JSP-PM is at most the optimal solution of the corresponding JSP. However, the optimal solution of the JSP-PM can be less. Figure 1 gives an example of an instance of the JSP, such that the corresponding JSP-PM with $k = 2$ has a shorter makespan, as shown in the Gantt charts in Figure 1.

3. Scheduling for the JSP-PM

We present a novel way to deduce the schedule for JSP-PM which does not include the replicated machines explicitly in the DG representation. We use the usual DG representation of JSP ignoring the replicated machines, but adapt the procedure that computes the schedule and makespan. Given a complete orientation of the DG we derive the associated schedule starting the execution of each operation as soon as all preceding operations, according to the DG, have been scheduled.

Our algorithm processes all operations in topological order, as defined by the DG. This can be done in time $O(nm)$. Each operation $o \in O$ will be assigned to one of the k machines of its stage $\mu(o)$ and be scheduled on this machine. The assignment and starting time depends on the completion time of its job predecessor and on the occupation of k machines of its corresponding stage.

During the scheduling procedure when we process operation $o \in O$ all the preceding operations of the corresponding job $\iota(o)$ have been scheduled at stages different from $\mu(o)$, and all

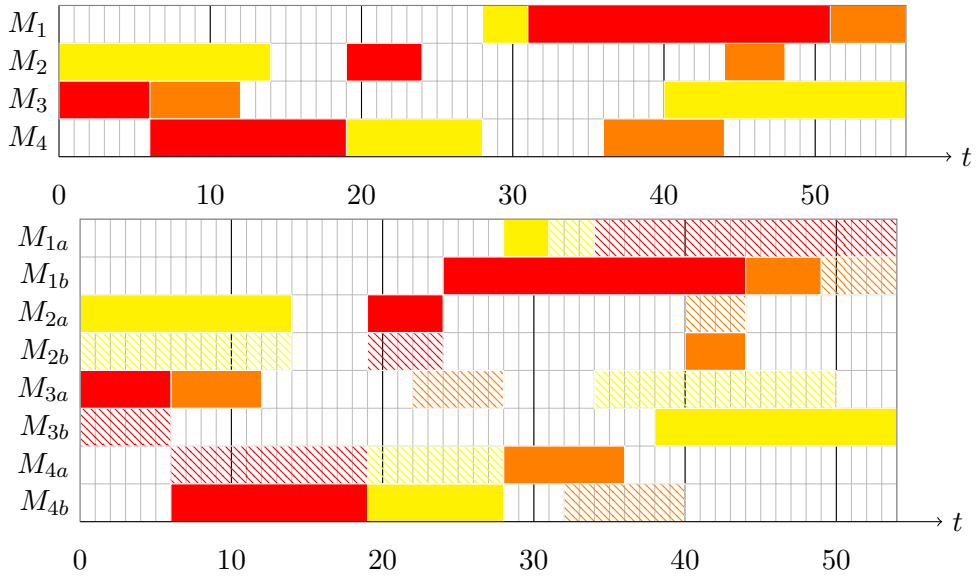

Figure 1: Top: Gantt chart of the optimal solution with $C_{\max} = 56$ of an instance of the JSP with three jobs and four machines. Bottom: Gantt char of the optimal solution with $C_{\max} = 54$ of the corresponding instance with $k = 2$ parallel machines and the double number of jobs. The replicated jobs have the same color, but are hatched. The optimal makespan of the parallel instance is less than the optimal makespan of the base instance.

the preceding operations at the same stage $\mu(o)$ have being scheduled at one of the k machines of stage $\mu(o)$. Given an operation o and its job predecessor p, we call the completion time of operation p in the current partial schedule the *job head* $h_J(o)$ of operation o. We call the completion time of the last operation scheduled at the ith machine at stage $\mu(o)$ the ith *machine head* $h^i_M(o)$ of operation o.

To schedule operation o, we check if there exists a machine i such that $h_J(o) \geq h^i_M(o)$. If this is the case, we can schedule operation o at time $h_J(o)$ without delay. Among all non-delay machines i which satisfy $h_J(o) \geq h^i_M(o)$ we assign the operation to the one of largest machine head $h^i_M(o)$. In this way operation o will start as soon as its job predecessor finishes at the machine with the highest current completion time. If no machine satisfies $h_J(o) \geq h^i_M(o)$ the set of non-delay machines is empty and we choose the machine i with the lowest machine head and schedule the operation at time $h^i_M(o)$. In this case the schedule will have idle time between operation o and its job predecessor and will start as soon as any of the available machines at its stage is free.

This procedure generates only non-delay schedules, but does not guarantee to generate active schedules. It is possible to generate any non-delay schedule with this procedure with the proper ordering of the DG and at least one of the optimal schedules is a non-delay schedule. Therefore it is always possible to represent at least one optimal solution for any instance of JSP-PM in this way.

4. Tabu Search for the Parallel Machines Job-Shop Scheduling Problem

Tabu search is meta-heuristic combining local search with a short-term memory proposed by Glover [13]. Starting from an initial solution it repeatedly passes to the best neighboring solution. If a local optimum is reached, this neighbor will be worse. To avoid returning immediately to the local optimum, attributes of previously visited solutions are stored in a so-called *tabu list* and declared *tabu* for a number of iterations, the *tabu tenure*. The modified local search is allowed to accept only non-tabu neighbors. In this way cycling is avoided. To not to miss good solutions often the tabu rules allow exceptions defined by *aspiration criteria*. A common aspiration criterion is to allow accepting tabu neighbors if they improve the incumbent, i.e. the current best solution.

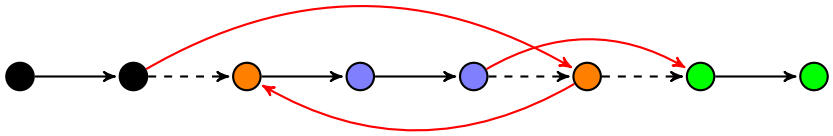

Figure 2: Example of a backward shift.

We propose a Tabu search for JSP-PM. In the following subsections we explain the neighborhood structure and the details of the search, including the generation of an initial solution, the tabu list, and the elite set, an intensification mechanism we use.

4.1. Neighbourhood

Let S_o and C_o be the starting and completion time, respectively, of operation o in a schedule. Given a non-delay schedule there exists at least one permutation of a subset of the operations $\{o_1, o_2, ..., o_x\}$ such that o_1 starts at time 0, o_x finishes at the makespan of the schedule, and $C_{o_i} = S_{o_{i+1}}$ for all $i \in [1, x)$. Such a sequence of operations is called a critical path. Any two adjacent operations (o_i, o_{i+1}) in a critical path are either executed on the same machine or belong to the same job. Moreover, in the usual DG representation for the JSP they are also directly connected by a conjunctive or disjunctive arc. With our scheduling scheme this is necessarily true only if they belong to the same job, otherwise o_i may be any of the preceding operations of o_{i+1} in the DG that share the same machine.

The neighbourhood we propose depends on a critical path. It consists of all the states generated by a *backward shift* of two adjacent operations in the critical path that share the same machine. The second operation is moved right before the first one in the DG. An example of a backward shift is shown in Figure 2. The rightmost orange operation is moved right before the leftmost one. To achieve this the black arcs remain the same, the dashed arcs are removed, and the red arcs are inserted. This may generate invalid states because the resulting DG may contain a cycle. In such cases we attribute makespan ∞ to the solution since it does not represent a valid schedule.

4.2. Tabu Search

Algorithm 1 shows the pseudo-code of the proposed tabu search. The algorithm is deterministic and has six parameters (I, Δ, L, T, C and R) which will be explained below.

The Insa procedure generates the initial solution. It is a constructive heuristic which inserts one operation at a time in the schedule. It start by inserting all operations of the job with the largest total processing time. Thus, the resulting partial schedule of each machine contains the single corresponding operation of this job. The remaining operations are processing in order of non-increasing processing times. Each operation is inserted into the current partial schedule of its corresponding machine, such that the length of the longest path passing through the current operation is minimal.

The *neighbourhood search procedure* NSP evaluates and selects one of the neighbours at each iteration. The tabu list contains pair of operations, which cannot be shifted. Each time a move is performed, the inverse operation is added to the tabu list with a tabu tenure of T. The NSP tests all neighbours of the current solution and chooses the next solution with the following priority:

a) The best non-tabu move that improves the best solution found so far.

b) The best tabu move that improves the best solution found so far.

c) The best non-tabu move.

d) The oldest tabu move.

Algorithm 1 Tabu Search

Input: parameters $I_{\max}$, Δ, L, T, C, and R.
$S^* = \text{Insa}()$
$I = I_{\max}$
add S^* to elite set E
while $E \neq \emptyset$ **do**
  get the next solution S from E
  $i = 0$
  $Jump = false$
  while $Jump$ is false **do**
    $i = i + 1$
    $S = \text{NSP}(S)$
    if $C_{\max}(S) < C_{\max}(S^*)$ **then**
      $I = I_{\max}; i = 0$
      update the incumbent S^*
      insert S^* into the elite list E
    end if
    if $Cycle(C, R)$ or $i > I$ **then**
      $I = I - \Delta$
      $Jump = true$
    end if
  end while
end while
Return S^*

If we select the oldest tabu move, the algorithm artificially forwards the number of iterations performed in the tabu list until a move is available.

For intensification the tabu search uses an elite list as a long-term memory. The elite list contains the L best solutions found during the search. For each solution in the elite list, we maintain a list of neighbors that have been examined already. Each time a new best solution is found it is inserted into the elite list E along with the associated tabu list and the neighbours that were not chosen by the NSP. If the list already contains L solutions, the oldest element of the list, namely the solution, candidate and tabu list, is discarded.

The tabu search starts at the best neighbor of the initial solution. It runs with a budget of a most I iterations, which is renewed whenever the incumbent improves. Otherwise after I iterations or when a cycle has been detected, the current solution is abandoned, and the tabu search continues from the next solution from the elite list. If the elite list is empty, the search stops. The budget I is initially $I_{\max}$ and is reduced by Δ whenever a solution is abandoned. If the incumbent improves, the budget is reset to $I_{\max}$.

In order to avoid cycling, the search includes a simple cycle detector with parameters C and R. If the makespan of the last C solutions is repeated R times then the search also abandons the current solution and continues from the next solution in the elite list.

5. Computational results

In this section we report experimental results with the proposed tabu search and compare them to results from the literature. We have implemented the proposed tabu search in C++. It has been compiled with the GNU g++ version 5.3.1. All experiments have been done on a PC with an 8-core AMD FX-8150 processor and 32 GB of main memory. In the experiments only one core has been used. The tabu search has been evaluated on a set of 30 instances derived from a set of 15 instances for the JSP proposed by Lawrence [16]. The number of jobs n and the number of machines m

Table 2: Characteristics of the instances used in the computational experiments.

Inst.	n	m	Inst.	n	m	Inst.	n	m
la01	10	5	la06	15	5	la11	20	5
la02	10	5	la07	15	5	la12	20	5
la03	10	5	la08	15	5	la13	20	5
la04	10	5	la09	15	5	la14	20	5
la05	10	5	la10	15	5	la15	20	5

Table 3: Parameter setting used in the experimental tests.

Maximum number of iterations without improvement I	2500
Reduction of the number of iterations after backtracking Δ	400
Size of the elite list L	5
Number of makespans to test for cycle C	100
Number of repeats to configure a cycle R	2
Tabu tenure T	8

of these instances are shown in Table 2. The 30 instances for the JSP-PM have been obtained by introducing $k = 2$ and $k = 3$ parallel machines, and replicating each job by the same amount in each of the 15 instances.

The chosen parameter settings for the tabu search can be seen in Table 3. The maximum number of iterations without improving I, the reduction of the number of iterations after backtracking Δ, the size of the elite list L, as well as the two parameters C and R associated with the cycle detector have been fixed after some preliminary tests. The tabu tenure T has been calibrated by an iterative F-Race [4]. We have used the R package irace [17]. An iterative F-Race samples the parameter space, runs the algorithm with different parameter settings, and identifies the best performing parameter settings by a non-parametric Friedman test with post hoc tests. For the racing we have chosen the six instances, la06 and la07 with one machine ($k = 1$), la01 and la11 with two parallel machine ($k = 2$), and la02 and la12 with three parallel machines ($k = 3$) and a budget of 3000 evaluations. The initial parameter range for the tabu tenure has been fixed at $[4, 15]$. The racing found the optimal value $T = 8$. This value has been used in the tests below.

In the literature there are no better solutions for the parallel instances than the best known values for JSP, therefore the best known values reported are the best known values for the corresponding JSP instances, which are proven to be optimal solutions for the case without replicated machines.

In Table 4 we see the results of the experiments. The table reports for each of the instances the number of replicated machines k, the best known value ("BKV"), the best makespan $C_{\max}$ found and the total computation time in seconds ($t(s)$) for the Hybrid Genetic algorithm and Ant Colony optimization proposed by Rossi and Boschi [21] (HGA) and the proposed Tabu search (Tabu). For clarity we do not replicate the BKV for $k = 3$, since it is always the same as $k = 2$.

Rossi and Boschi [21] have conducted their experiments on a PC with an AMD Athlon 2800 processor. Based on scores from the PassMark benchmark, a conservative estimate is that our computing environment is about a factor of 2.5 faster. We have adjusted the running time reported by Rossi and Boschi [21] accordingly to make them comparable to our running times.

Additionally, Table 5 reports the average relative deviations in percent ("Dev.") and the average running times in seconds over all instances, and the instances with $k = 2$ and $k = 3$ replicated machines separately. This table is an indicative of the scalability of both algorithms regarding the number of parallel machines.

Table 4: Comparison of the performance of the Tabu search and HGA.

			HGA		Tabu search	
Name	k	BKV	$C_{\max}$	$t(s)$	$C_{\max}$	$t(s)$
la01	2	666	**666**	73.20	**666**	0.52
	3		677	144.40	**669**	1.29
la02	2	655	688	88.40	**665**	0.60
	3		712	145.20	**658**	2.68
la03	2	597	626	116.00	**606**	1.09
	3		673	99.20	**630**	1.15
la04	2	590	611	124.80	**598**	0.95
	3		629	241.20	**597**	1.97
la05	2	593	**593**	0.00	**593**	0.39
	3		**593**	77.20	**593**	0.64
la06	2	926	**926**	0.00	**926**	0.99
	3		936	169.60	**926**	2.00
la07	2	890	894	44.00	**890**	1.84
	3		922	480.80	**890**	5.24
la08	2	863	**863**	5.20	**863**	1.16
	3		871	182.40	**864**	1.48
la09	2	951	**951**	0.00	**951**	1.06
	3		**952**	243.60	953	2.36
la10	2	958	**958**	0.00	**958**	0.96
	3		**958**	177.20	**958**	1.77
la11	2	1222	**1222**	0.80	**1222**	4.20
	3		1239	887.60	**1222**	7.30
la12	2	1039	**1039**	0.80	**1039**	2.01
	3		1049	990.80	**1039**	4.41
la13	2	1150	**1150**	0.00	**1150**	2.76
	3		1163	924.40	**1150**	6.41
la14	2	1292	**1292**	0.00	**1292**	2.14
	3		**1292**	170.80	**1292**	4.54
la15	2	1207	1246	144.00	**1207**	8.35
	3		1283	812.80	**1222**	22.35

6. Conclusions

The tabu search clearly outperforms HGA in both time and quality of the found solutions. From the 30 proposed instances the Tabu search achieved the same result or better than HGA in 29, being surpassed only in instance la09 with 3 machine replications by one time unit. We obtained strictly better results than the HGA in 16 instances, 11 of them for the instances with $k = 3$ replicated machines.

The execution time of the tabu search is considerably lower than that of the HGA in most of the instances. In average the tabu search is more than a factor 10 faster for the instances with $k = 2$ replicated machines, and more than 80 times faster for the instances with $k = 3$ replicated machines. The better scalability in relation to the number of replicated machines of the tabu search is probably due to the strategy of not changing the DG representation with the number of replicated machines, which maintains the search space size for the tabu search independent of the number of replicated machines k.

We were able to show that the proposed tabu search can generate good results in a timely

Table 5: Average deviations from Bkv and average execution times.

	HGA		Tabu	
k	Dev. (%)	$t(s)$	Dev. (%)	$t(s)$
All	2.08	211.48	0.45	3.15
2	1.14	39.81	0.29	1.93
3	3.01	383.15	0.61	4.37

manner combining a representation that ignores the replicated machines with a greedy strategy to schedule such machines. The main contribution of this work is the scheduling strategy for the replicated machines, combined with the new neighbourhood that uses an adapted critical path. The results generated by the proposed tabu search are good, both in time and quality of the generated solutions. We emphasize the scalability of the heuristic with an increasing number of parallel machines, as shown in Table 5. The average relative deviation from the best known solutions and the processing times for the instances with three replicated machines is about the double of that of the instances with two replicated machines.

Future work can focus on larger instances both in the number of jobs, distinct machines and replicated machines, since the base JSP instances for the proposed instance set are small. Moreover, since the jobs are replicated in the instance set, they have a symmetry that is not explicitly exploited by the algorithm. One may study both the efficacy of the algorithm in instances without such a symmetry as well as identify and exploit such symmetries explicitly when they arise. The literature is also rich with works that combine the routing flexibility of the jobs with the parallel machines flexibility. It seems probable that an approach similar to this one will work well for such problems.

References

[1] Joseph Adams, Egon Balas, and Daniel Zawack. “The shifting bottleneck procedure for job shop scheduling”. In: *Manag. Sci.* 34.3 (1988), pp. 391–401.

[2] Sheldon B Akers Jr. “A graphical approach to production scheduling problems”. In: *Oper. Res.* 4.2 (1956), pp. 244–245.

[3] David Applegate and William Cook. “A computational study of the job-shop scheduling problem”. In: *ORSA J. Comput.* 3.2 (1991), pp. 149–156.

[4] P. Balaprakash, M. Birattari, and T. Stützle. “Improvement strategies for the F-race algorithm: Sampling design and iterative refinement”. In: *HM 2007*. Ed. by T. Bartz-Beielstein et al. Vol. 4771. LNCS. 2007, pp. 108–122.

[5] Egon Balas and Alkis Vazacopoulos. “Guided Local Search with Shifting Bottleneck for Job Shop Scheduling”. In: *Management Science* 44.2 (1998), pp. 262–275.

[6] Peter Brucker, Bernd Jurisch, and Bernd Sievers. “A branch and bound algorithm for the job-shop scheduling problem”. In: *Discrete Appl. Math.* 49.1 (1994), pp. 107–127.

[7] Jacques Carlier. “The one-machine sequencing problem”. In: *European Journal of Operational Research* 11.1 (1982), pp. 42 –47. ISSN: 0377-2217.

[8] Jacques Carlier and Eric Pinson. “An algorithm for solving the job-shop problem”. In: *Manag. Sci.* 35.2 (1989), pp. 164–176.

[9] M. R. Garey, D. S. Johnson, and R. Sethi. “The complexity of flowshop and jobshop scheduling”. In: *Math. Oper. Res.* 1.2 (1976), pp. 117–129.

[10] Michael R Garey and David S Johnson. “Computers and intractability: a guide to the theory of NP-completeness. 1979”. In: *San Francisco, LA: Freeman* (1979).

[11] Omid Gholami and Yuri N Sotskov. "A fast heuristic algorithm for solving parallel-machine job-shop scheduling problems". In: *The International Journal of Advanced Manufacturing Technology* 70.1-4 (2014), pp. 531–546.

[12] Omid Gholami and Yuri N Sotskov. "Solving parallel machines job-shop scheduling problems by an adaptive algorithm". In: *Int. J. Prod. Res.* 52.13 (2014), pp. 3888–3904.

[13] Fred Glover. "Tabu search for nonlinear and parametric optimisation (with links to genetic algorithms)". In: *Discrete Appl. Math.* 49 (1994), pp. 231–255.

[14] José Fernando Gonçalves and Mauricio GC Resende. "An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling". In: *International Transactions in Operational Research* 21.2 (2014), pp. 215–246.

[15] Joaquim A.S. Gromicho et al. "Solving the job-shop scheduling problem optimally by dynamic programming". In: *Computers and Operations Research* 39.12 (2012), pp. 2968 – 2977. ISSN: 0305-0548.

[16] S. Lawrence. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*. Tech. rep. Pittsburgh: Graduate School of Industrial Administration, Carnegie Mellon University, 1984.

[17] Manuel López-Ibáñez et al. *The irace package, Iterated Race for Automatic Algorithm Configuration*. Tech. rep. TR/IRIDIA/2011-004. IRIDIA, Université libre de Bruxelles, 2011.

[18] John F Muth and Gerald Luther Thompson. *Industrial scheduling*. Prentice-Hall, 1963.

[19] Eugeniusz Nowicki and Czeslaw Smutnicki. "A Fast Taboo Search Algorithm for the Job Shop Problem". In: *Management Science* 42.6 (1996), pp. 797–813.

[20] Eugeniusz Nowicki and Czeslaw Smutnicki. "An Advanced Tabu Search Algorithm for the Job Shop Problem". In: *J. of Scheduling* 8.2 (Apr. 2005), pp. 145–159. ISSN: 1094-6136.

[21] Andrea Rossi and Elena Boschi. "A hybrid heuristic to solve the parallel machines job-shop scheduling problem". In: *Advances in Engineering Software* 40.2 (2009), pp. 118–127.

[22] Jean-Paul Watson, Adele E Howe, and L Darrell Whitley. "Deconstructing Nowicki and Smutnicki's i-TSAB tabu search algorithm for the job-shop scheduling problem". In: *Comput. Oper. Res.* 33.9 (2006), pp. 2623–2644.