Pergamon

*Mathl. Comput. Modelling* Vol. 24, No. 9, pp. 11–19, 1996

0895-7177/96 $15.00 + 0.00

**PII: S0895-7177(96)00150-1**

# A Local Search Heuristic for Unrelated Parallel Machine Scheduling with Efficient Neighborhood Search

N. PIERSMA AND W. VAN DIJK
Econometric Institute, Erasmus University Rotterdam
P.O. Box 1738
NL-3000 DR Rotterdam, The Netherlands
piersma@opres.few.eur.nl

*(Received December 1995; accepted February 1996)*

**Abstract**—The parallel machine scheduling problem with unrelated machines is studied where the objective is to minimize the maximum makespan. In this paper, new local search algorithms are proposed where the neighborhood search of a solution uses the "efficiency" of the machines for each job. It is shown that this method yields better solutions and shorter running times than the more general local search heuristics.

**Keywords**—Unrelated parallel machine scheduling, Local search.

## 1. INTRODUCTION

Consider $m \geq 2$ parallel machines and $n$ jobs. Each job needs to be scheduled on exactly one machine, and every machine can process at most one job at a time. No preemption is allowed and there are no release times and due dates. The $R||C_{\max}$ scheduling problem is to find a schedule that minimizes the makespan, i.e., the time necessary to process all jobs. Therefore, the order in which the jobs are processed on a machine is not important, and a feasible schedule can be denoted by the assignment of each job to a machine.

Let $p_{ij}$ denote the processing time of job $j$ on machine $i$. In general, one can distinguish between the following three cases:

1. identical machines: $p_{ij} = p_{1j}$ for all $i$ and $j$;
2. uniform machines: $p_{ij} = p_j/v_i$ for all $i$ and $j$, where $v_i$ is the speed of machine $i$, and $p_j$ is the processing time at unit speed;
3. unrelated machines: $p_{ij}$ arbitrary for all $i$ and $j$.

Denote the minimum makespan by $C^*_{\max}$. We will concentrate on the case of unrelated machines, where the processing time of each job can differ on every machine. The problem is well known to be NP-hard, even in the simplest case of two identical machines [1].

Exact algorithms for the problem were developed by Van der Velde [2] and Horowitz and Sahni [3]. However, the computational requirements for these exact algorithms become very large for more than five machines and 50 jobs.

Much research effort has been put into designing fast and efficient approximation algorithms with good theoretical performance bounds. For an overview of the existing methods, see [4]. Some algorithms are based on simple list scheduling rules, like the earliest completion time heuristic of Ibarra and Kim [5]. The methods with the best theoretical performance bounds are the

LP-based heuristics by Potts [6] and Lenstra, Shmoys and Tardos [7] and the efficient assignment algorithms by Davis and Jaffe [8].

Hariri and Potts [9] already showed that the approximation methods, despite of their theoretical guarantees, are outperformed by simple iterative local improvement algorithms. Glass, Potts and Shade [10] studied a number of local search heuristics for $R||C_{\max}$. These local search methods are often quite general and do not exploit the structure of the problem to which they are applied. We will introduce an improved iterative local improvement algorithm, based on the iterative local improvement algorithm by Hariri and Potts and on the ideas by Davis and Jaffe. The novelty of the algorithm consists of an efficient search of the neighborhoods. We will show that the algorithm outperforms the general local search methods in both solution quality and running time. Next, we will show that a Taboo search algorithm with the efficient neighborhood search strategy also performs better than the general local search algorithms.

In the next section, the basic ideas behind the new improvement method are exhibited and an outline of the algorithm is presented. In Section 3, the new method is compared primarily to the local search heuristics formulated by Glass, Potts, and Shade and to the simple iterative improvement algorithm formulated by Hariri and Potts.

Some alternative ideas, such as the efficient Taboo search algorithm that have been tested are discussed in Section 4, and the last section contains some final remarks.

## 2. ITERATIVE DESCENT USING THE STRUCTURE OF $R||C_{\max}$

The local search algorithm consists of the following elements.

ALGORITHM 2.1. LOCAL SEARCH ALGORITHM.

**Starting point**

Generate a initial feasible schedule $s$ and compute its maximum makespan.

**Neighborhood search**

Select a feasible schedule $g$ in the neighborhood of $s$ and compute its maximum makespan.

**Acceptance test**

Decide whether or not to move from schedule $s$ to schedule $g$.

If the move is accepted, continue with schedule $g$.

Otherwise, continue with schedule $s$.

**Termination test**

Decide whether to stop the algorithm. If the algorithm is terminated, then output the current schedule and its maximum makespan.

Otherwise, return to the Neighborhood search.

The major decisions when applying this type of problem is the choice of the starting solution and the neighborhood search method. We will use a simple starting schedule and the well-known relocation and two-exchange neighborhoods to select alternative schedules. However, the main difference with the existing local search methods is the search direction, where we will use the structure of the $R||C_{\max}$ problem by means of the following concept [8].

Let $m_j = \min_{1 \le i \le m} p_{ij}$ be the smallest processing time of job $j$ and define the *efficiency* of machine $i$ for job $j$ by

$$\text{eff}(i,j) = \frac{m_j}{p_{ij}}.$$

The efficiency thus indicates a preference for assigning a job to a machine by means of a value in the interval $(0,1]$. Notice that a job has the smallest processing time on a machine where it has efficiency one. Let $s_{ij}$ be the decision variable that takes on the value 1 if job $j$ is scheduled on machine $i$ and 0 otherwise. Let a feasible schedule be denoted by $s = (s_{ij})_{i,j}$ and let $\mathcal{S}$ be the

set of all feasible schedules. We can formulate the $R||C_{\max}$ problem such as to find

$$C^*_{\max} = \min_{s \in S} \max_{1 \le i \le m} \sum_{j=1}^{n} \frac{m_j}{\text{eff}(i,j)} s_{ij}.$$

Since the $m_j$ are fixed constants, it seems best to assign each job to a machine on which it has efficiency one. However, in this schedule, all the jobs may be assigned to the same machine yielding an unbalanced workload. The assignment of the jobs relies thus on a trade off between a high efficiency of the jobs assigned to a machine and the workload of the machines. We will formulate a neighborhood search method that first considers schedules where every job is assigned to the machine such that the corresponding efficiency is as large as possible and then changes this schedule using the efficiencies in nonincreasing order.

To generate a starting schedule, we use a greedy approach (further denoted by GR) that is a simple on-line list scheduling procedure also known as the minimum processing time or shortest processing time heuristic.

STARTING POINT: GR HEURISTIC

> Consider the jobs in any sequence. Assign each job to the machine on which it has minimal processing time. When there is more than one machine with the smallest processing time, choose the machine with the smallest makespan in the partial schedule. Use the smallest index rule when there is still a choice between several machines.

Thus, the starting point is a feasible schedule $s$ where every job is assigned to the machine on which it has efficiency one. However, the schedule may be highly unbalanced. The neighborhood of the starting point is then searched for a more balanced schedule in the following way.

NEIGHBORHOOD SEARCH: EFF DESCENT METHOD

There are two neighborhoods that are considered for solution $s$:

$N_R(s) = \{g \in \mathcal{S} : g$ can be obtained from $s$ by reassigning one job from a machine with maximum makespan to another machine$\}$

$N_I(s) = \{g \in \mathcal{S} : g$ can be obtained from $s$ by interchanging the machine assignment of a pair of jobs$\}$

The search for an alternative schedule $g$ is performed first in neighborhood $N_R(s)$ and then in neighborhood $N_I(s)$.

Searching $N_R(s)$

1. Choose machine $m_{\max}$ as the machine with the smallest index among the machines with maximum makespan. Define the set $J_{\max}$ of the jobs assigned to machine $m_{\max}$. Also define the set $E_{\max} = \{(i,j) : i \in \{1,\ldots,m\}, i \neq m_{\max}$ and $j \in J_{\max}\}$ of all other possible assignments of the jobs assigned to machine $m_{\max}$.
2. When the set $E_{\max}$ is empty stop. Otherwise, select job $j_r$ and machine $i_r$ such that

$$\text{eff}(i_r, j_r) = \max_{(i,j) \in E_{\max}} \text{eff}(i,j).$$

   When there are multiple choices for $i_r$ and $j_r$, choose the ones with the smallest index.
3. When the new schedule is accepted goto step 1. Otherwise remove the point $(i_r, j_r)$ from the set $E_{\max}$ and goto step 2 with the current schedule.

Searching $N_I(s)$

1. Let $M_1$ be the ordered set that contains the machine indexes in nonincreasing order of the makespans (In case of ties, use the largest index rule). Also define the set $M_2$ that contains the machine indexes in nondecreasing makespan sequence. (In case of ties, use the smallest index rule). Let $m_1$ be the first element of $M_1$, and $m_2$ the first element of $M_2$.

2. Define the sets $J_1$ and $J_2$ of jobs assigned to the machine $m_1$ and $m_2$.
3. Sort the jobs in $J_1$ in nonincreasing order of the efficiencies of the jobs on machine $m_2$. Also sort the jobs in $J_2$ in nonincreasing order of the efficiencies of the jobs on machine $m_1$.
4. Set $k = 1$ and $j = 0$. Assume EXCHANGE = FALSE
   While NOT EXCHANGE and $k < |J_1| + 1$ do
   Begin
   j:=j+1.
   If the schedule $g$ with $k$ and $j$ interchanged is accepted then set EXCHANGE = TRUE. Else set $j := j + 1$
   If $j = |J_2|$ then set $k := k + 1$ and $j := 0$
   End
   If EXCHANGE=FALSE then goto step 5.
   If EXCHANGE=TRUE then goto step 1.
5. Choose for $m_2$ the next element in $M_2$. If $m_2 = m_1$ then choose for $m_1$ the next machine index in $M_1$ and let $m_2$ be the first element of the set $M_2$. If $m_1$ is the first element of the set $M_2$ then stop.

ACCEPTANCE TEST

For solutions $g \in N_R(s)$ the acceptance test is as follows.
Compare the maximum makespan $C_{\max}(s)$ of schedule $s$ with the maximum makespan of machine $i_r$ in schedule $g$ where job $j_r$ is reassigned to machine $i_r$. Move to schedule $g$ if the makespan of machine $i_r$ in schedule $g$ is smaller than $C_{\max}(s)$. Otherwise continue with schedule $s$.

For the solutions $g \in N_I(s)$, the acceptance test is the following.
Let $g$ be the schedule that is generated by exchanging job $j$ and $k$ in the schedule $s$. When the largest makespan of the machines that process job $j$ and $k$ in schedule $g$ is smaller than the largest makespan of the machines processing job $j$ and $k$ in schedule $s$, we move to schedule $g$. Otherwise continue with schedule $s$.

TERMINATION TEST

First the neighborhood $N_R(s)$ is searched for new schedules. When a new schedule $g$ is accepted, the neighborhood $N_R(g)$ is searched for new schedules. When none of the schedules in $N_R(s)$ is accepted, we search the neighborhood $N_I(s)$. If a schedule $g \in N_I(s)$ is accepted, the neighborhood $N_I(g)$ is searched. When none of the schedules in $N_I(g)$ is accepted the algorithm terminates.

Our local search method, further denoted by GR/EFF, is a simple iterative descent method that differs from the descent method by Hariri and Potts in the way the neighborhoods are searched and in the starting solution. Using the structure of the schedules, we believe that a better local optimum can be obtained with less computational effort. Observe that in the case of identical machines, the jobs have efficiency one on every machine and that in the case of uniform machines all jobs have efficiency $v_i/(\max_{1\leq k\leq m} v_k)$ on machine $i$. The method is thus not expected to perform well for identical or uniform machine scheduling problems, but very well for unrelated machines.

Notice that the greedy heuristic has a running time of $O(nm)$ and a worst case ratio of $m$, and that each neighborhood search has a running time of $O(m^2n^2)$. For bounded input (i.e., all $p_{ij} < \infty$), the running time of the algorithm is bounded because the algorithm only accepts a new schedule when a makespan becomes smaller and terminates when there exists no improved schedule in the neighborhood of the current schedule.

# 3. RESULTS

## 3.1. Experimental Design

A series of tests is performed using experimental design by Glass, Potts, and Shade [10]. The tests are coded in Turbo Pascal 7.0 on a 486DX-66 microcomputer. The problem sizes are 2,3,5,10, or 25 machines with 50, 100, 150, and 200 jobs, and 50 machines with 50 and 100 jobs.

There are four structures for the processing times.

TP = 1 No correlation between the machines or the jobs, $p_{ij}$ continuous uniformly distributed within the interval $[0, 1]$.

TP = 2 No correlation between the machines or the jobs, $p_{ij}$ discrete uniformly distributed within the interval $[1, 100]$.

TP = 3 The jobs are correlated, $p_{ij}$ discrete uniformly distributed within the interval $[\beta_j+1, \beta_j+20]$ and $\beta_j$ discrete uniformly distributed within the interval $[1, 100]$.

TP = 4 The machines are correlated, $p_{ij}$ discrete uniformly distributed within the interval $[\alpha_i + 1, \alpha_i + 20]$ and $\alpha_i$ discrete uniformly distributed within the interval $[1, 100]$.

## 3.2. Test Results

First, we compared the solutions derived with the GR/EFF method with optimal solutions. For small problems ($n \leq 200$, $m = 2, 3, 4$), for which we calculated the optimal schedule using a simple branch and bound method, the average relative deviation percentage to the optimal solution is investigated:

$$\text{ardp} = 100 * \frac{C_{\max}^{\text{GR/EFF}} - C_{\max}^{*}}{C_{\max}^{*}},$$

where we take the average over 10 problem instances for every problem type and size.

For almost every test problem, the GR/EFF algorithm has an average relative deviation percentage less than 1%, with an average running time of less than 0.7 seconds.

Second, we have compared the GR/EFF algorithm to another simple descent algorithm, called the ECT/SD algorithm. This algorithm uses the ECT-approximation-method to compute a feasible schedule. This schedule will then be used as a start for a simple descent algorithm (SD) in the second phase, where other schedules are selected by job index in the relocation and the interchange neighborhood (Hariri and Potts). The tests are performed on five problem instances for every size and type of problem. The best known solution value is taken as a replacement of the optimal solution value when this optimum is not known.

Table 1 shows that the GR/EFF algorithm has a smaller average relative deviation percentage for almost every problem instance. This difference in the average quality of solution is especially noticeable for problem instances where the ratio $n/m$ is large ($> 3$) and where $m$ is large ($> 10$). If we compare the average running times of both algorithms (see Table 2), we see that the GR/EFF algorithm becomes faster as the number of jobs becomes larger. The use of the GR-solution ($O(mn)$) instead of the ECT-solution ($O(mn^2)$) seems to be favourable, the second phase of the GR/EFF method will make up for the poor starting solution fast enough. However, for problem type TP = 4, the running time of the GR/EFF method becomes very large. On the extreme, the machine correlation results in all machines being assigned to one machine, then a balancing between this machine and one other machine is performed, etc. For large $m$, the running time of the algorithm thus becomes very large. When we apply the GR/EFF heuristic to problem structures uncommon to the $R||C_{\max}$ problem, the heuristic does not perform well. The heuristic thus indeed uses the structure of the $R||C_{\max}$ problem to find a solution.

Third, we have compared the GR/EFF algorithm to the local search methods described by Glass, Potts and Shade, that is, simulated annealing, Taboo search and genetic algorithms. We gave all the local search algorithms a maximum running time of 100 seconds because the GR/EFF method only exceeds this running time once (TP = 4, $m = 25$, $n = 200$).

Table 1. Average relative deviation percentages for the GR/EFF heuristic (left) and for the ECT/SD algorithm (right).

| | | Average relative deviation percentage | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $m$ | $n$ | TP = 1 | | TP = 2 | | TP = 3 | | TP = 4 | |
| | | GR EFF | ECT SD | GR EFF | ECT SD | GR EFF | ECT SD | GR EFF | ECT SD |
| 3 | 50 | 0.54 | 2.51 | 0.61 | 2.15 | 0.32 | 1.48 | 0.21 | 0.92 |
| | 100 | 0.31 | 0.43 | 0.26 | 1.63 | 0.12 | 0.77 | 0.08 | 0.92 |
| | 150 | 0.25 | 0.98 | 0.32 | 0.69 | 0.06 | 0.44 | 0.18 | 0.94 |
| | 200 | 0.17 | 0.57 | 0.12 | 0.64 | 0.06 | 0.44 | 0.11 | 0.99 |
| 5 | 50 | 1.04 | 8.05 | 2.20 | 4.76 | 0.10 | 2.00 | 0.26 | 1.46 |
| | 100 | 1.09 | 2.57 | 0.73 | 3.28 | 0.09 | 1.36 | 0.50 | 1.70 |
| | 150 | 0.21 | 2.42 | 0.30 | 2.91 | 0.02 | 1.04 | 0.30 | 1.17 |
| | 200 | 0.10 | 2.54 | 0.00 | 2.24 | 0.02 | 0.84 | 0.19 | 1.59 |
| 10 | 50 | 6.86 | 13.21 | 7.78 | 15.47 | 1.20 | 3.88 | 1.98 | 4.55 |
| | 100 | 1.03 | 8.55 | 1.45 | 9.94 | 0.11 | 2.70 | 1.24 | 2.71 |
| | 150 | 1.09 | 6.51 | 0.12 | 7.04 | 0.00 | 1.78 | 1.50 | 1.81 |
| | 200 | 0.39 | 3.48 | 0.11 | 4.87 | 0.00 | 1.47 | 0.49 | 1.45 |
| 25 | 50 | 4.13 | 5.59 | 2.35 | 14.02 | 4.74 | 7.97 | 1.56 | 3.96 |
| | 100 | 10.21 | 16.83 | 4.16 | 19.43 | 1.25 | 3.35 | 0.88 | 2.52 |
| | 150 | 8.48 | 12.51 | 0.00 | 10.11 | 0.45 | 2.99 | 1.48 | 1.06 |
| | 200 | 2.28 | 7.67 | 0.52 | 10.51 | 0.33 | 2.35 | 0.66 | 1.94 |
| 50 | 50 | 5.25 | 5.25 | 10.16 | 10.16 | 0.00 | 0.00 | 1.53 | 5.55 |
| | 100 | 8.26 | 8.16 | 6.67 | 6.11 | 3.00 | 4.04 | 0.84 | 3.17 |

Table 2. Average running time for the GR/EFF heuristic (left) and for the ECT/SD algorithm (right).

| | | Average running time (sec) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $m$ | $n$ | TP = 1 | | TP = 2 | | TP = 3 | | TP = 4 | |
| | | GR EFF | ECT SD | GR EFF | ECT SD | GR EFF | ECT SD | GR EFF | ECT SD |
| 3 | 50 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 | 0.1 |
| | 100 | 0.2 | 0.2 | 0.2 | 0.3 | 0.2 | 0.2 | 0.7 | 0.3 |
| | 150 | 0.3 | 0.6 | 0.4 | 0.5 | 0.3 | 0.6 | 2.6 | 0.5 |
| | 200 | 0.5 | 0.9 | 0.5 | 0.9 | 0.7 | 0.9 | 5.4 | 0.9 |
| 5 | 50 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.5 | 0.1 |
| | 100 | 0.3 | 0.3 | 0.3 | 0.3 | 0.4 | 0.3 | 3.5 | 0.3 |
| | 150 | 0.5 | 0.7 | 0.6 | 0.7 | 0.8 | 0.7 | 9.9 | 0.8 |
| | 200 | 0.9 | 1.3 | 1.1 | 1.3 | 1.2 | 1.3 | 25.4 | 1.3 |
| 10 | 50 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 | 0.2 | 1.7 | 0.1 |
| | 100 | 0.4 | 0.5 | 0.5 | 0.5 | 0.6 | 0.6 | 10.1 | 0.6 |
| | 150 | 0.8 | 1.2 | 1.0 | 1.2 | 1.5 | 1.2 | 43.9 | 1.2 |
| | 200 | 1.4 | 2.1 | 1.4 | 2.1 | 2.2 | 2.2 | 81.5 | 2.2 |
| 25 | 50 | 0.2 | 0.3 | 0.3 | 0.3 | 1.0 | 0.4 | 3.6 | 0.3 |
| | 100 | 0.6 | 1.3 | 0.9 | 1.3 | 2.1 | 1.4 | 28.9 | 1.3 |
| | 150 | 1.6 | 2.8 | 1.9 | 2.9 | 3.7 | 3.0 | 92.4 | 2.9 |
| | 200 | 2.4 | 5.1 | 3.8 | 5.1 | 6.1 | 5.3 | 178.8 | 5.1 |
| 50 | 50 | 0.4 | 0.7 | 0.4 | 0.7 | 6.2 | 0.7 | 5.4 | 0.7 |
| | 100 | 1.0 | 2.6 | 1.6 | 2.6 | 11.6 | 3.2 | 51.8 | 2.6 |

The results are reported in Table 3. The GR/EFF algorithm seems to be the best algorithm for almost all problem instances with job correlation in the processing times (TP = 3). For the problem instances TP = 1 and TP = 2, the GR/EFF algorithm seems to be the better algorithm when the number of jobs is large ($\geq 150$).

The GR/EFF method produces one local optimum whose quality is competitive to that of the solution of simulated annealing, Taboo search and a genetic algorithm, while the last three algorithms are designed to evaluate more than one local optimum. In the next section, we will consider a number of alternatives among which a Taboo search algorithm of the GR/EFF algorithm.

Table 3. The GR/EFF compared to local search algorithms.
Left: GR/EFF average relative deviation percentage
Middle: BEST average relative deviation percentage
Right: Name BEST algorithm
GA = Genetic algorithm
SA = Simulated annealing
TS = Taboo search

| $m$ | $n$ | TP = 1 | | | TP = 2 | | | TP = 3 | | | TP = 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | a.r.d.p. | | name | a.r.d.p. | | name | a.r.d.p. | | name | a.r.d.p. | | name |
| | | GR EFF | BST | BST | GR EFF | BST | BST | GR EFF | BST | BST | GR EFF | BST | BST |
| 3 | 50 | 0.54 | 0.01 | GA | 0.61 | 0.00 | SA | 0.32 | 0.20 | GA | 0.21 | 0.00 | GA |
| | 100 | 0.31 | 0.14 | GA | 0.26 | 0.11 | GA | 0.12 | GR | EFF | 0.08 | 0.00 | GA |
| | 150 | 0.25 | GR | EFF | 0.32 | 0.18 | TS | 0.06 | GR | EFF | 0.18 | 0.03 | GA |
| | 200 | 0.17 | GR | EFF | 0.12 | GR | EFF | 0.06 | GR | EFF | 0.11 | GR | EFF |
| 5 | 50 | 1.04 | 0.09 | GA | 2.20 | 0.00 | GA | 0.10 | GR | EFF | 0.26 | 0.00 | GA |
| | 100 | 1.09 | 0.67 | TS | 0.73 | 0.64 | TS | 0.09 | GR | EFF | 0.50 | 0.03 | GA |
| | 150 | 0.21 | GR | EFF | 0.30 | GR | EFF | 0.02 | GR | EFF | 0.30 | 0.02 | GA |
| | 200 | 0.10 | GR | EFF | 0.00 | GR | EFF | 0.02 | GR | EFF | 0.19 | GR | EFF |
| 10 | 50 | 6.86 | 1.52 | GA | 7.78 | 0.00 | GA | 1.20 | GR | EFF | 1.98 | 0.00 | GA |
| | 100 | 1.03 | GR | EFF | 1.45 | GR | EFF | 0.11 | GR | EFF | 1.24 | 0.17 | GA |
| | 150 | 1.09 | GR | EFF | 0.12 | GR | EFF | 0.00 | GR | EFF | 1.50 | 0.08 | GA |
| | 200 | 0.39 | GR | EFF | 0.11 | GR | EFF | 0.00 | GR | EFF | 0.49 | 0.38 | GA |
| 25 | 50 | 4.13 | 0.00 | TS | 2.35 | 0.00 | TS | 4.74 | 1.04 | TS | 1.56 | 0.62 | SA |
| | 100 | 10.21 | 2.73 | TS | 4.16 | GR | EFF | 1.25 | GR | EFF | 0.88 | 0.50 | SA |
| | 150 | 8.48 | 1.20 | TS | 0.00 | GR | EFF | 0.45 | GR | EFF | 1.48 | 0.95 | TS |
| | 200 | 2.28 | GR | EFF | 0.52 | GR | EFF | 0.33 | GR | EFF | 0.66 | GR | EFF |
| 50 | 50 | 5.25 | 0.00 | TS | 10.16 | 0.00 | TS | 0.00 | GR | EFF | 1.53 | 0.84 | SA |
| | 100 | 8.26 | 0.00 | TS | 6.67 | 5.56 | TS | 3.00 | 0.33 | SA | 0.84 | GR | EFF |

# 4. ALTERNATIVE METHODS

## 4.1. Taboo Search Using the Structure of $R||C_{\max}$

We tested a simple Taboo search algorithm that uses the efficient neighborhood search structure for the parallel scheduling problem. In this algorithm, the exchange and relocate neighborhoods are combined into one neighborhood $N$. When no improved schedule is accepted in the neighborhood $N(s)$ of the current schedule $s$, we select the best solution found during the search of $N(s)$.

The alternative schedules in the exchange and relocate neighborhood $N(s)$ are checked for membership of a Taboo list. The Taboo list is of traditional length ($L = 7$) and consists of the objective values of the last seven schedules that were accepted. An alternative $g$ in the neighborhood $N(s)$ of the current schedule $s$ is Taboo if the value of $g$, $C_{\max}(g)$ is in the Taboo list. However, the Taboo list is overruled by the following aspiration function. Let $W(\ )$ be the number of machines with maximum makespan $C_{\max}(\ )$. We will overrule a Taboo for schedule $g$ if

$$W(g) \leq W(s),$$

that is, if the number of machines in schedule $g$ with maximum makespan $C_{\max}(g)$ is not larger than the number of machines in the current schedule $s$ with makespan $C_{\max}(s)$. The idea is to make room for future replacements and exchanges.

Table 4 shows that the solution quality of the Taboo search heuristic with efficient search is better than traditional Taboo search. The Taboo search heuristic with efficient search is also compared to the GR/EFF method, where the Taboo search runs for 100 seconds. In most cases, efficient Taboo search found the best solution. Thus, also for the Taboo local search method, the efficient search of the neighborhoods results into better quality solutions.

Table 4. Average relative deviation percentages for the GR/EFF heuristic (left) and the TABOO GR/EFF algorithm (right).

| | | Average relative deviation ($r$%) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $m$ | $n$ | TP = 1 | | TP = 2 | | TP = 3 | | TP = 4 | |
| | | GR EFF | TABOO | GR EFF | TABOO | GR EFF | TABOO | GR EFF | TABOO |
| 3 | 50 | 0.54 | 0.19 | 0.61 | 0.34 | 0.32 | 0.07 | 0.21 | 0.00 |
| | 100 | 0.31 | 0.10 | 0.26 | 0.10 | 0.12 | 0.08 | 0.08 | 0.00 |
| | 150 | 0.25 | 0.25 | 0.32 | 0.09 | 0.06 | 0.04 | 0.18 | 0.10 |
| | 200 | 0.17 | 0.12 | 0.12 | 0.04 | 0.06 | 0.04 | 0.11 | 0.07 |
| 5 | 50 | 1.04 | 0.55 | 2.20 | 0.59 | 0.10 | 0.00 | 0.26 | 0.16 |
| | 100 | 1.09 | 0.47 | 0.73 | 0.32 | 0.09 | 0.02 | 0.50 | 0.46 |
| | 150 | 0.21 | 0.04 | 0.30 | 0.11 | 0.02 | 0.00 | 0.30 | 0.28 |
| | 200 | 0.10 | 0.02 | 0.00 | 0.00 | 0.02 | 0.00 | 0.19 | 0.06 |
| 10 | 50 | 6.86 | 2.95 | 7.78 | 1.88 | 1.20 | 0.00 | 1.98 | 0.45 |
| | 100 | 1.03 | 0.00 | 1.45 | 0.63 | 0.11 | 0.00 | 1.24 | 0.61 |
| | 150 | 1.09 | 0.00 | 0.12 | 0.00 | 0.00 | 0.00 | 1.50 | 1.42 |
| | 200 | 0.39 | 0.01 | 0.11 | 0.00 | 0.00 | 0.00 | 0.49 | 0.49 |
| 25 | 50 | 4.13 | 0.00 | 2.35 | 0.00 | 4.74 | 0.76 | 1.56 | 0.61 |
| | 100 | 10.21 | 0.58 | 4.16 | 0.00 | 1.25 | 0.00 | 0.88 | 0.88 |
| | 150 | 8.48 | 1.90 | 0.00 | 0.00 | 0.45 | 0.06 | 1.48 | 1.48 |
| | 200 | 2.28 | 0.12 | 0.52 | 0.00 | 0.33 | 0.09 | 0.66 | 22.29 |
| 50 | 50 | 5.25 | 0.00 | 10.16 | 0.00 | 0.00 | 0.00 | 1.53 | 0.57 |
| | 100 | 8.26 | 2.22 | 6.67 | 1.77 | 3.00 | 1.77 | 0.84 | 0.57 |

## 4.2. Alternatives of the GR/EFF Algorithm

We have also investigated several alternatives of the GR/EFF algorithm.

Because the GR algorithm is a fast, on-line algorithm with a poor quality of solution, we have analyzed the ECT/EFF algorithm, where the initial schedule is constructed using the ECT heuristic. The tests showed that the average relative deviation percentage of the ECT/EFF algorithm is larger than that of the GR/EFF algorithm. It seems to be essential to start with a schedule where every job is assigned to a machine on which it has efficiency one, rather than to start with a schedule where the workload is more balanced. The running time of the ECT/EFF algorithm shows to be larger, especially for the problem instances with uncorrelated processing times (TP = 1, 2) and with job correlated processing times (TP = 3).

The second idea is to use the efficiency assignment heuristic of Davis and Jaffe to construct a starting feasible schedule. This heuristic does take into account the efficiency of a machine for a job, but results into a more balanced schedule. However, the tests on the resulting algorithm EFF/EFF show that the GR heuristic should be preferred as a starting point. The running time of the EFF/EFF algorithm is larger and the solution quality is worse than that of the GR/EFF algorithm in almost all cases. We conclude that the GR/EFF algorithm cannot benefit from a more balanced starting schedule.

We also tried to make the improvement-phase faster by considering only machines for the relocation and interchange of jobs that have an efficiency of at least 0.4 (like Davis and Jaffe do in their efficiency assignment heuristic). It turns out that this structure gives no improvement in the average running time, and serious worse average relative deviation percentages.

## 5. CONCLUSIONS

In this paper, we report on the results of a local improvement method and a Taboo search method with efficient search for the parallel machine problem with unrelated machines. We showed that these methods have a better performance than their counterparts that do not search the neighborhood efficiently. In particular, the local improvement algorithm GR/EFF has a running time that is comparable to the descent algorithm of Hariri and Potts (less than 10 seconds for most test problems) and yields solutions that are of comparable quality to the local search heuristics that have user defined running times (100 seconds in our test cases).

The Taboo search algorithm with efficient neighborhood search should only be used to obtain high quality solutions, since the running time is user defined (100 seconds in our test cases). Notice that also a Simulated Annealing algorithm can be constructed that uses efficient search of the neighborhoods.

It is worthwhile to use the structure of the problem in a local search method for the $R||C_{\max}$ scheduling problem. We claim that the performance of improvement and local search methods for other combinatoral optimization problems will also benefit from the incorporation of problem specific features. Recently proposed methods in scheduling theory [11,12] support this view. Future research should, therefore, pay attention to the special features of combinatorial problems that can be used to improve the performance of local search methods.

## REFERENCES


1. R.M. Karp, Reducibility among combinatorial problems, In *Complexity of Computer Computations* (Edited by R.E. Miller and J.W. Thatcher), pp. 85–103, Plenum Press, New York, (1972).
2. S.L. van der Velde, Duality based algorithms for scheduling unrelated parallel machines, *ORSA Journal on Computing* **5**, 192–205 (1993).
3. E. Horowitz and S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, *J. Assoc. Comput. Mach.* **23**, 317–327 (1976).
4. E.G. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, Sequencing and scheduling: Algorithms and complexity, In *Handbooks in OR & MS*, Vol. 4, Chapter 9 (Edited by S.C. Graves, A.H.G. Rinnooy Kan and P.H. Zipkin), Elsevier Science Publishers B.V., Amsterdam, (1993).
5. O.H. Ibarra and C.E. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors, *Journal of the Association for Computing Machinery* **24**, 280–289 (1977).
6. C.N. Potts, Analysis of a linear programming heuristic for scheduling unrelated parallel machines, *Discrete Applied Mathematics* **10**, 155–164 (1985).
7. J.K. Lenstra, D.B. Shmoys and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming* **46**, 259–271 (1990).
8. E. Davis and J.M. Jaffe, Algorithms for scheduling tasks on unrelated processors, *Journal of the Association for Computing Machinery* **28**, 721–736 (1981).
9. A.M.A. Hariri and C.N. Potts, Heuristics for scheduling unrelated parallel machines, *Computers and Operations Research* **18**, 323–331 (1991).
10. C.N. Potts, C.A. Glass and P. Shade, Unrelated parallel machine scheduling using local search, *Mathl. Comput. Modelling* **20** (2), 41–52 (1994).
11. H.R. Lourenco and M. Swijnenburg, Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem, Technical Report, Centro de Investigacao Operacional, Faculdade de Ciencias da Universidade de Lisboa, (1995).
12. H.R. Lourenco, Job-shop scheduling: Computational study of local search and large-step optimization methods, *European Journal of Operational Research* **83**, 347–364 (1995).