Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

S.L. van de Velde

Duality-based algorithms for scheduling unrelated parallel machines

---

Department of Operations Research, Statistics, and System Theory Report BS-R9010 May

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Duality-Based Algorithms for Scheduling Unrelated Parallel Machines

S.L. van de Velde
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam
The Netherlands

We consider the following parallel machine scheduling problem. Each of n independent jobs has to be scheduled on one of m unrelated parallel machines. The processing of job J_j on machine M_i requires a positive processing time p_{ij}. The objective is to find an assignment of jobs to machines so as to minimize the maximum job completion time. The objective of this paper is to design practical algorithms for this $\mathcal{NP}$-hard problem. We present optimization and approximation algorithms, in which the notion of duality plays a key role. The optimization algorithm is capable of solving quite large problems within reasonable time limits. The approximation algorithm is based upon a novel concept for iterative local search, in which the search direction is guided by dual multipliers.

1980 Mathematics Subject Classification (Revision 1985): 90B35.
Key Words & Phrases: parallel machine scheduling, unrelated parallel machines, maximum completion time, linear programming, duality, approximation algorithm, branch-and-bound, duality-based heuristic search.
Note: This paper has been submitted for publication.

1. Introduction

We consider the following machine scheduling problem. There are m parallel machines available for processing n independent jobs. Each of these machines can handle at most one job at a time. The processing of job J_j $(j = 1, \ldots, n)$ on machine M_i $(i = 1, \ldots, m)$ requires a positive integral processing time p_{ij}. Each job has to be scheduled on one of the machines and has to be processed without interruption. A *schedule* is an assignment of each of the jobs to exactly one machine. The length of the schedule, also referred to as the *makespan*, is the maximum job completion time, which is equal to the maximum machine completion time. The objective is to find a schedule of minimum length.

This problem may arise in the context of computer system scheduling, where the machines are processors of a distributed computing environment with varying capabilities across the tasks. Another application is found in the area of flexible manufacturing systems, in which a cluster of parallel machines form a single or bottleneck stage in the production process.

In case $p_{ij} = p_j$ for each J_j and M_i, the machines are said to be *identical*. If $p_{ij} = p_j / s_i$, where s_i denotes the speed of machine M_i, the machines are *uniform*. In the general case, the machines are *unrelated*. Following the notation of Graham, Lawler, Lenstra, and Rinnooy Kan (1979), we refer to these problems as $P \mid\mid C_{\max}$, $Q \mid\mid C_{\max}$, and $R \mid\mid C_{\max}$, respectively.

Since $P2 \mid\mid C_{\max}$ is already $\mathcal{NP}$-hard, there does not exist a polynomial-time algorithm for $R \mid\mid C_{\max}$ unless $\mathcal{P} = \mathcal{NP}$. The traditional problem is to balance solution quality with running

time: an optimal solution may only be found at the expense of an exponential amount of computation time, while a polynomial-time algorithm cannot be guaranteed to produce the optimal solution.

Two attempts have been made to solve $R||C_{\max}$ to optimality. Stern (1976) presents a branch-and-bound algorithm, and Horowitz and Sahni (1976) develop a dynamic programming procedure. In both cases, no computational results are reported.

Much research effort has been invested in the development of approximation algorithms with a guaranteed accuracy. An approximation algorithm that never delivers a schedule length of more than ρ times the optimal length is referred to as a ρ-approximation algorithm. In other words, the approximation algorithm has *worst-case performance ratio* ρ. Ibarra and Kim (1977) and Davis and Jaffe (1981) propose various approximation algorithms with worst-case performance ratios that increase with the number of machines. For *fixed m* (i.e., the number of machines is specified as part of the problem type and not of the problem instance), Horowitz and Sahni (1976) give a fully polynomial approximation scheme, which has time and space complexity $O(nm(nm/(\rho-1))^{m-1})$. A *polynomial approximation scheme* is a family of algorithms that contains for any $\rho > 1$ a ρ-approximation algorithm with a running time that is bounded by a polynomial in the problem size; this running time may depend on ρ. A family of algorithms is called a *fully polynomial approximation scheme* if it contains for any $\rho > 1$ a ρ-approximation algorithm for which the running time is bounded by a polynomial in the problem size as well as in $1/(\rho-1)$.

Potts (1985) presents a 2-approximation algorithm. The running time of this algorithm is polynomial only for fixed m, but the space required is polynomial in m. For the two-machine case, Potts shows that the worst-case ratio can even be improved to $(1+\sqrt{5})/2$. The algorithm is a two-phase procedure, in which linear programming is used in the first phase to assign at least $n-m+1$ jobs and complete enumeration is applied in the second phase to schedule the at most $m-1$ remaining jobs. Lenstra, Shmoys, and Tardos (1987) use Potts' algorithm as the basis for a 2-approximation algorithm that is polynomial in m. They also present a polynomial approximation scheme for a fixed number of machines, where the space required is bounded by a polynomial in the problem size and $\log(1/(\rho-1))$. In addition, they prove a notable negative result: unless $\mathcal{P} = \mathcal{NP}$, there exists no polynomial ρ-approximation algorithm for any $\rho < \frac{3}{2}$.

There are two papers that consider $R||C_{\max}$ from an empirical point of view. De and Morton (1981) present several hybrid list scheduling algorithms and perform a large-scale computational testing. From our experience, however, we found that their algorithms display a significant proportional deviation from the optimal solution. In the spirit of Potts' 2-approxiation algorithm, Hariri and Potts (1990) propose several two-phase heuristics that use linear programming in the first phase to schedule at least $n-m+1$ jobs and another heuristic to schedule the remaining jobs. They also consider several constructive heuristics that are used in conjunction with iterative local improvement procedures.

In spite of the considerable attention that the $R||C_{\max}$ problem has received, there is still a lack of practical algorithms and computational insight. This paper addresses this issue. We are concerned with methods that solve $R||C_{\max}$ satisfactorily from a practical standpoint. We develop a branch-and-bound algorithm that is capable of solving relatively large problems to optimality within reasonable time limits. In addition, we propose a new approximation

algorithm that produces outstanding results. Both algorithms make extensive use of the notion of duality in min-max integer linear programming. Furthermore, we will discuss the relationships between these duality-based algorithms and the linear programming based approximation algorithm of Potts for the case $m = 2$.

The organization of this paper is as follows. In Section 2, we give a mathematical programming formulation of the $R \mid\mid C_{\max}$ problem and we examine the properties of its dual. In Section 3, we discuss the ingredients of the approximation algorithm, which is based upon the dual. A complete description of the branch-and-bound algorithm is given in Section 4, while some computational results are presented in Section 5. Conclusions are found in Section 6.

2. Minimizing makespan and its dual problem

The $\mathcal{NP}$-hardness of the $R \mid\mid C_{\max}$ problem justifies and motivates the development of approximative and enumerative algorithms. The concern of this section is to compute a lower bound, necessary to apply branch-and-bound, from the dual of the linear programming formulation of $R \mid\mid C_{\max}$. In addition, we will show that the lower bound computation can almost be integrated with the search for a good approximate solution.

Evidently, there is an optimal solution in which the jobs are processed without any inserted idle time. In addition, the ordering of the jobs on the respective machines is irrelevant for the length of the schedule. Therefore, we are actually looking for an *assignment* of jobs to machines. Accordingly, we introduce assignment variables x_{ij} ($i = 1, \ldots, m$, $j = 1, \ldots, n$) that take the value 1 if J_j is scheduled on M_i, and 0 otherwise. If we let C_i denote the completion time of machine M_i, we have $C_i = \Sigma_{j=1}^{n} p_{ij} x_{ij}$. The maximum value of the machine completion times, denoted by $C_{\max}$, is then the length of the schedule.

The $R \mid\mid C_{\max}$ problem, hereafter referred to as problem P, is to determine values x_{ij} that minimize

$$C_{\max} \tag{P}$$

subject to

$$\sum_{j=1}^{n} p_{ij} x_{ij} \leqslant C_{\max}, \qquad i = 1, \ldots, m, \tag{1}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j = 1, \ldots, n, \tag{2}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, m,\ j = 1, \ldots, n. \tag{3}$$

Conditions (1) in this formulation ensure that the completion time of each machine is less than or equal to the length of the schedule, while conditions (2) guarantee that each job is completely assigned. Conditions (3) ensure that each job is scheduled on *exactly one* machine, thereby precluding preemption.

In the remainder of the paper $v(\cdot)$ denotes the optimal value for problem $\cdot$.

A common strategy for lower bound computation is to identify a set of hard constraints, the relaxation of which makes the problem more agreeable. For instance, replacing the integrality constraints (3) with the weaker conditions $x_{ij} \geqslant 0$ ($i = 1, \ldots, m$, $j = 1, \ldots, n$) yields the *linear programming relaxation* $\bar{\mathrm{P}}$, which is efficiently solvable. We choose to apply the more involved

technique of *surrogate relaxation*. The central idea is to replace a set of troublesome constraints by a single condition that is a weighted aggregation of these constraints. In our application, we aggregate the constraints (1). We therefore introduce a vector of *surrogate multipliers* $\lambda = (\lambda_1, \ldots, \lambda_m) \geqslant 0$ with $\lambda_i > 0$ for at least one i ($i = 1, \ldots, m$), and replace the m conditions with

$$\sum_{i=1}^{m} \lambda_i \sum_{j=1}^{n} p_{ij} x_{ij} \leqslant C_{\max} \sum_{i=1}^{m} \lambda_i, \tag{1a}$$

or, equivalently,

$$C_{\max} \geqslant \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_i p_{ij} x_{ij} / \sum_{i=1}^{m} \lambda_i. \tag{1b}$$

The surrogate relaxation problem, hereafter referred to as problem S_λ, is then to determine $v(S_\lambda)$, which is the minimum of

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_i p_{ij} x_{ij} / \sum_{i=1}^{m} \lambda_i \tag{S_λ}$$

subject to

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j = 1, \ldots, n, \tag{2}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, m, \; j = 1, \ldots, n. \tag{3}$$

PROPOSITION 1. Problem S_λ provides a lower bound on $v(P)$, since any solution that satisfies (1) also satisfies (1b) (but not necessarily vice versa). Therefore, we have that $v(S_\lambda) \leqslant v(P)$ for any vector $\lambda \geqslant 0$.

PROPOSITION 2. Problem S_λ is solvable in $O(nm)$ time by assigning each job J_j to the machine M_h for which $\lambda_h p_{hj} = \min_{1 \leqslant i \leqslant m} \lambda_i p_{ij}$.

Note that $v(S_\lambda) = \Sigma_{j=1}^{n} \min_{1 \leqslant i \leqslant m} \lambda_i p_{ij} / \Sigma_{i=1}^{m} \lambda_i$. We refer to $\lambda_i p_{ij}$ as the dual processing time of J_j on M_i. The conditions (3) in the surrogate relaxation problem can be replaced with the conditions $x_{ij} \geqslant 0$ ($i = 1, \ldots, m$, $j = 1, \ldots, n$). Problem S_λ is said to have the *integrality property*, since it can be solved as a linear programming problem.

PROPOSITION 3. Any solution to S_λ, for any $\lambda \geqslant 0$, is also a feasible solution to the primal problem P. This is true, since constraints (2) enforce the assignment of each job to exactly one machine. The approximate solution value is given by $\max_{1 \leqslant i \leqslant m} C_i(\lambda)$, where $C_i(\lambda)$ denotes the completion time of M_i in the solution to problem S_λ.

PROPOSITION 4. The objective value $v(S_\lambda)$ is a *convex* combination of the machine completion times. This implies that $\min_{1 \leqslant i \leqslant m} C_i(\lambda) \leqslant v(S_\lambda) \leqslant \max_{1 \leqslant i \leqslant m} C_i(\lambda)$.

Consider the following example where eight jobs need to be scheduled on three machines

with the processing times given in Table 1. Let $\lambda = (1,1,1)$ be the vector of surrogate multi-

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
M_1	6	3	10	12	11	14	8	6
M_2	10	∞	15	6	6	11	14	7
M_3	11	9	14	14	∞	10	10	9

TABLE 1. Processing time matrix.

pliers. The multipliers are such that S_λ is solved by scheduling each job on the machine with the smallest processing time. This produces the schedule as represented by the *Gantt-chart* of Figure 1. The initial choice $\lambda = (1,1,1)$ results in an elementary lower bound: it is the sum of the minimum processing times divided by the number of machines. Therefore, the lower bound is $v(S_\lambda) = 18\frac{1}{3}$, and the length of the schedule is 33.

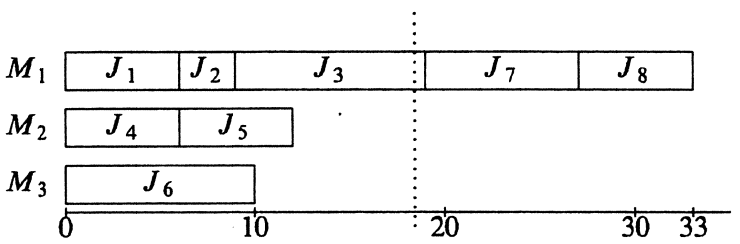

FIGURE 1. Gantt-chart for $\lambda = (1,1,1)$.

Naturally, we have a particular interest in finding the vector of surrogate multipliers that produces the largest lower bound. In fact, this is the *surrogate dual problem*, hereafter referred to as problem S, and it is defined as

$$v(S) = \max\{v(S_\lambda) \mid \lambda \geqslant 0\}. \tag{S}$$

PROPOSITION 5. Since S_λ possesses the integrality property, we have $v(\bar{P}) = v(S)$: the surrogate dual yields the same lower bound as the linear programming relaxation $\bar{P}$ (Greenberg and Pierskalla, 1970, Karwan and Rardin, 1979).

This result can also be derived in the following direct way. Geoffrion (1974) points out that it is feasible to take the dual of a linear programming problem with respect to only a portion of the constraints. We assert that doing so for problem $\bar{P}$ with respect to conditions (1) yields exactly the dual problem S, since the conditions (3) in problem S_λ can be replaced with $x_{ij} \geqslant 0$ $(i = 1,\ldots,m, j = 1,\ldots,n)$.

PROPOSITION 6. The surrogate dual problem $\max\{v(S_\lambda) \mid \lambda \geqslant 0\}$ is a convex programming problem, since $v(S_\lambda)$ is a linear non-differentiable concave function in λ (see Fisher, 1981).

Proposition 6 implies that any *local* optimum to problem S is also *global*. In addition, we know that the optimum is found in a point of non-differentiability. Hence, problem S can be solved

by an *ascent direction* technique. This is an iterative procedure that requires a *direction* and a *step size* by which the current vector λ should be perturbed in order to improve on the value $v(S_\lambda)$. Since $v(S_\lambda)$ is a concave function in λ, we may restrict ourselves to perturbing one component of λ per iteration. If no such direction can be found, then we have identified an optimal solution.

Solving problem S instead of problem $\bar{P}$ is motivated by two arguments. First, the ascent direction technique we propose generates a series of vectors of multipliers, and each of them has an associated lower bound. As we will see, the procedure requires only $O(mn)$ time per iteration. Second, each vector induces a feasible primal solution. It turns out that the quest of finding the vector of multipliers that solves problem S concurs with finding vectors that provide satisfactory approximate solutions.

We define machine M_i, $i = 1, \ldots, n$, to be *overloaded* in the solution of problem S_λ if $C_i(\lambda) > v(S_\lambda)$. Conversely, machine M_i is *underloaded* if $C_i(\lambda) < v(S_\lambda)$. In the remainder, the superscript t counts the number of iterations in the ascent direction procedure. For brevity, however, we denote $C_i(\lambda^t)$ by C_i^t. The basis of the ascent direction procedure is constituted by the next theorem.

THEOREM 1. *Consider problem* $S_{\lambda'}$ *and its solution. Identify a machine* M_h *with* $C_h^t > v(S_{\lambda'})$. *Let* U_h^t *be the set of jobs scheduled on* M_h *in the optimal solution of problem* $S_{\lambda'}$, $\Delta_h = \min_{1 \leq i \leq m,\, i \neq h,\, J_j \in U_h^t} (\lambda_i^t p_{ij} / p_{hj} - \lambda_h^t)$, *and* $\lambda^{t+1} = (\lambda_1^t, \ldots, \lambda_h^t + \Delta_h, \ldots, \lambda_m^t)$. *If* $\Delta_h > 0$, *then* $v(S_\lambda^{t+1}) > v(S_\lambda^t)$.

PROOF. The value Δ_h is the smallest perturbation of λ_h^t such that some job J_j currently scheduled on the overloaded machine M_h can equally well be scheduled on some other machine M_g. In view of Proposition 2, this implies that $(\lambda_h^t + \Delta_h)p_{hj} = \lambda_g^t p_{gj}$. In addition, we have $C_h^{t+1} = C_h^t - p_{hj}$ and $C_g^{t+1} = C_g^t + p_{gj}$. Hence, we have

$$v(S_\lambda^{t+1}) = \frac{\sum_{i=1}^{m} \lambda_i^{t+1} C_i^{t+1}}{\sum_{i=1}^{m} \lambda_i^{t+1}} = \frac{(\lambda_h^t + \Delta_h)(C_h^t - p_{hj}) + \lambda_g^t (C_g^t + p_{gj}) + \sum_{i=1;\, i \neq g,h}^{m} \lambda_i^t C_i^t}{\Delta_h + \sum_{i=1}^{m} \lambda_i^t}$$

$$= \frac{\Delta_h C_h^t + \sum_{i=1}^{m} \lambda_i^t C_i^t}{\Delta_h + \sum_{i=1}^{m} \lambda_i^t} = \frac{\Delta_h C_h^t + \dfrac{\sum_{i=1}^{m} \lambda_i^t C_i^t}{\sum_{i=1}^{m} \lambda_i^t} \sum_{i=1}^{m} \lambda_i^t}{\Delta_h + \sum_{i=1}^{m} \lambda_i^t} > \frac{(\Delta_h + \sum_{i=1}^{m} \lambda_i^t) v(S_\lambda^t)}{\Delta_h + \sum_{i=1}^{m} \lambda_i^t} = v(S_\lambda^t).$$

□

THEOREM 2. *Consider problem* S_{λ^t} *and its solution. Identify a machine* M_h *with* $C_h^t < v(S_{\lambda^t})$. *Let* $\overline{U}_h^t$ *be the set of jobs not scheduled on* M_h *in the optimal solution of problem* S_{λ^t}, $\Delta_h = \min_{1 \leq i \leq m;\, i \neq h,\, J_j \in \overline{U}_h^t} (\lambda_i^t p_{ij}/p_{hj} - \lambda_h^t)$, *and* $\lambda^{t+1} = (\lambda_1^t, \ldots, \lambda_h^t + \Delta_h, \ldots, \lambda_m^t)$. *If* $\Delta_h < 0$, *then* $v(S_\lambda^{t+1}) > v(S_\lambda^t)$.

PROOF. Analogous to the proof of Theorem 1. □

The iterative application of Theorem 1 and Theorem 2 is an ascent direction procedure to solve problem (S). Evidently, such a procedure solves problem (S), since it always identifies an ascent direction and a corresponding step size, if one exists.

Given an initial multiplier λ^1, it takes $O(mn)$ time to solve problem S_{λ^1}. If we create for each machine M_i a list with elements $\lambda_i p_{ij}$ $(j = 1, \ldots, n)$, sorted in non-decreasing order, then it takes only $O(mn)$ time to compute an ascent direction, its associated step size, and the solution to problem S_{λ^2}. The relevant lists are updated in only constant time. The ascent direction procedure is not a polynomial-time algorithm, since there is no guarantee that the number of iterations is polynomially bounded.

Let us reconsider our example and the solution of S_{λ^1} with $\lambda^1 = (1,1,1)$. Machine M_1 has here maximum completion time. According to Theorem 1, λ_1^1 has to be increased by $\Delta_1 = \frac{1}{6}$, which makes it possible to move J_8 to M_2. Doing so, we obtain a schedule with makespan 27 and $v(S_{\lambda^2}) = 19.1$ (see Figure 2). Machine M_1 has still the largest overload; if we increase λ_1

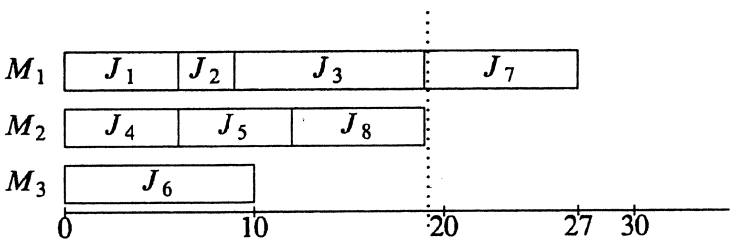

FIGURE 2. Gantt-chart for $\lambda = (\frac{7}{6}, 1, 1)$.

to $\frac{5}{4}$, then we have also the option to schedule J_7 on M_3. Note that for the new value of λ job J_8 must be executed on M_2. The associated schedule has makespan 20 and $v(S_{\lambda^3}) = 19.3$ (see Figure 3). Since all processing times are assumed to be integral, the optimal makespan must be integral as well. Hence, we have found an optimal primal solution. Note that we have not yet solved the dual problem; M_2 is underloaded and a nonzero stepsize can be computed. If we decrease λ_2 by $\frac{1}{10}$, then J_6 can equally well be scheduled on M_2. Accordingly, we get $\lambda^4 = (\frac{5}{4}, \frac{9}{10}, 1)$ and $v(S_{\lambda^4}) = 19\frac{44}{139}$. This is the optimal vector of multipliers, as it is impossible to compute a nonzero stepsize: J_6 can be scheduled both on M_2 and M_3, and J_8 both on M_2 and M_1. Therefore, the schedule in Figure 3 represents only one of the optimal dual solutions.

At this point, we discuss the relationship with Potts' 2-approximation algorithm for the case $m = 2$. For an arbitrary number of machines, the first step in Potts' algorithm is to solve problem $\overline{P}$, the linear programming relaxation problem of $R \mid\mid C_{\max}$. This provides us with an assignment in which at most $m - 1$ jobs are split over two or more machines and in which at

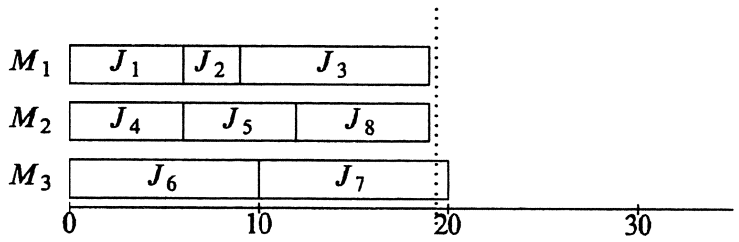

FIGURE 3. Gantt-chart for $\lambda = (\frac{5}{4}, 1, 1)$.

least $n - m + 1$ jobs are assigned to exactly one machine. The jobs that have been assigned to exactly one machine are retained as a partial schedule. The split jobs are assigned so as to minimize the makespan, given the partial schedule. Since $v(\bar{P}) \leqslant v(P)$, the length of the partial schedule is no more than $v(P)$. The scheduling of the split jobs proceeds by complete enumeration; this adds at most $v(P)$ to the length of the partial schedule. Hence, the resulting schedule has a makespan at most twice the optimal makespan. Since $\bar{P}$ is solvable in polynomial time and complete enumeration for at most $m - 1$ split jobs requires $O(m^m)$ time, the procedure is polynomial for fixed m. It is easy to verify that the split jobs in the solution to $\bar{P}$ correspond to the jobs in the solution to S for which the minimum dual processing time is attained by more than one machine. In our example, J_6 and J_7 would be the split jobs in the solution to problem $\bar{P}$.

In the case $m = 2$, the linear programming relaxation is solvable in $O(n)$ time (Gonzalez, Lawler and Sahni, 1990). Moreover, there is at most one fractional job. The solution generated by Potts' algorithm then concurs with the best primal solution found by solving problem S through the ascent direction procedure.

3. DUALITY-BASED HEURISTIC SEARCH

The approximation algorithm we propose conveys the idea that a near-optimal dual solution induces a good primal solution. In that sense, we need to develop a scheme that generates a series of promising dual multipliers. As seen from the example, the search for such multipliers might be integrated into the ascent direction method. The ascent direction method as such, however, is too restrictive for our purpose. From computational experience, it appeared that it generally solves the dual in only a few iterations. In contrast, we need a scheme that allows us to browse quickly through various near-optimal solutions for problem S. On that ground, the approximation algorithm differs on two counts from the ascent direction method. First, we always select the machine for multiplier adjustment that has the largest load. From a primal point of view, this choice is easily justified: one of the jobs scheduled on this machine must be rescheduled in order to lower the completion time that induces the current makespan. Second, we increase its multiplier more than would be strictly necessary to enforce a change in the schedule. Let machine M_h be such that $C_h^t = C_{\max}^t$, where $C_{\max}^t = \max_{1 \leqslant i \leqslant m} C_i^t$ and let U_h^t denote the set of jobs currently scheduled on M_h. Then we compute

$$\delta_h = \text{min2}_{1 \leqslant i \leqslant m, J_j \in U_h^t} (\lambda_i^t p_{ij} / p_{hj} - \lambda_h^t),$$

where min2 denotes the second minimum of these values. Note that $\Delta_h \leqslant \delta_h$. If we put

$\lambda^{t+1} = (\lambda_1^t, \ldots, \lambda_h^t + \delta_h, \ldots, \lambda_m^t)$, then we enforce that some J_k leaves M_h and goes to some other machine M_g, and that another job can equally well be scheduled on M_h as on some other machine. Nonetheless, we keep this second job scheduled on M_h. The next step is to compute $C_{\max}^{t+1}$. We have no guarantee that the rescheduling of J_k induces an improved schedule: we can have either $C_{\max}^{t+1} \leqslant C_{\max}^t$ or $C_{\max}^{t+1} > C_{\max}^t$. The latter occurs if $C_g^{t+1} = C_g^t + p_{gk} > C_{\max}^t$. Hence, the approximation algorithm is equipped with a mechanism that accepts deteriorations of the makespan. We repeat this process for the machine with the largest load, and store the best solution on the way. Notice that we must put an upper bound on the number of iterations, since this procedure does not have any convergence properties. Below we give a stepwise description of the algorithm, in which *maxiter* is some prespecified number of maximum iterations and *ub* is the currently best solution value.

APPROXIMATION ALGORITHM

Step 1. Put $\lambda^1 \leftarrow (1, \ldots, 1)$, $t \leftarrow 1$, $ub \leftarrow \infty$, and solve S_{λ^1}.

Step 2. Determine M_h with $C_h^t = C_{\max}^t$. If $C_h^t < ub$, put $ub \leftarrow C_h^t$ and store the current schedule. Compute δ_h, and identify a job J_k and a machine M_g such that $\lambda_g^t p_{gk} / p_{hk} = \min_{1 \leqslant i \leqslant m,\, i \neq h,\, J_j \in U_h^t} (\lambda_i^t p_{ij} / p_{hj})$. Put $t \leftarrow t+1$.

Step 3. Put $C_h^{t+1} \leftarrow C_h^t - p_{hk}$, $C_g^{t+1} \leftarrow C_g^t + p_{gk}$, $\lambda^{t+1} \leftarrow (\lambda_1^t, \ldots, \lambda_h^t + \delta_h, \ldots, \lambda_m^t)$, and solve $S_{\lambda^{t+1}}$. If $t < maxiter$, go to Step 2, else stop.

In the remainder, we refer to the approximation algorithm described above as the *duality-based approximation algorithm*, and to the particular strategy employed as *duality-based heuristic search*. Note that the approximation algorithm applied to the example goes through the same steps as described in Section 2.

There is a wide variety in heuristic search strategies that are all applicable to the parallel machine scheduling problem. Most of them have in common that they adjust the current schedule somewhat to try to improve on its value. Let σ be some arbitrary schedule and let σ_{jk} be the schedule that can be obtained from σ by swapping J_j and J_k $(j \neq k)$. We then define the so-called *single pairwise interchange neighborhood* for σ as the set N_σ that comprises the schedules σ_{jk} for all $j = 1, \ldots, n-1$, $k = j+1, \ldots, n$. Suppose M_h is such that $C_h(\sigma) = C_{\max}(\sigma)$, where $C_h(\sigma)$ and $C_{\max}(\sigma)$ denote the completion time of M_h and the maximum machine completion time in σ, respectively. Let (J_j, J_k) be a pair of jobs such that J_j is scheduled on M_h and J_k on some other machine M_g, $(g \neq h)$, for which we have

$$C_g + p_{gj} - p_{gk} < C_g, \text{ and } C_h - p_{hj} + p_{hk} < C_h.$$

If we interchange J_j and J_k, that is, we put J_j on M_g and J_k on M_h, then we reduce the makespan. In other words, we have identified a schedule $\sigma_{jk} \in N_\sigma$ with $C_{\max}(\sigma_{jk}) < C_{\max}(\sigma)$. This process can be repeated until no further improvement is found. *Iterative local improvement procedures* are based upon these concepts. The main danger is to get stuck in a relatively poor local optimum, from which no escape is possible. The traditional policy to avoid this pitfall is to use multiple schedules as starting points in order to obtain multiple local minima. Hopefully, one of these local minima is then a satisfactory approximate solution. Some refinements have been developed, among which *simulated annealing* and *tabu search* take prominent places.

Simulated annealing (see e.g. Van Laarhoven and Aarts, 1987) leaves the possibility open to travel from one local optimum to another. This is achieved by accepting deteriorations of the objective value with a probability that is a decreasing function of running time. Tabu search (Glover, 1989, De Werra and Herz, 1989) is much similar to simulated annealing, but provides a deterministic mechanism to accept deteriorations. The willingness to accept deteriorations unconditionally marks the duality-based search technique as described above from simulated annealing, tabu search, and general iterative local improvement schemes.

Anticipating on the implementation and the evaluation of the duality-based approximation algorithm in Section 5.2, however, we will consider two versions of the algorithm. On the one hand, we evaluate the duality-based algorithm on its own, and on the other hand we evaluate the algorithm in conjunction with an iterative local improvement procedure of the type we described. For the latter case, we only submitted the best solution to the improvement procedure. It appeared that the duality-based algorithm in conjunction with the iterative local improvement procedure produced excellent results. Apparently, the duality-based approximation algorithm succeeds in finding an attractive starting solution, the neighborhood of which can be further explored by the iterative local improvement procedure.

4. The branch-and-bound algorithm

The first step in the branch-and-bound algorithm is to solve problem S and to compute the optimal vector of dual multipliers $\lambda^* = (\lambda_1^*, \ldots, \lambda_m^*)$ through the ascent direction procedure described in Section 2. On the way, we store the best approximate primal solution. We also try to find an improved upper bound through the duality-based approximation algorithm and the constructive heuristics presented by De and Morton, Ibarra and Kim, and Davis and Jaffe. The implementation of this process is described in Section 5. The vector λ^* plays an important rule in the growth and truncation of the search tree.

4.1. Initial reductions

The size of an instance may be reduced by a simple reduction test, which is common in linear programming theory. It can be conducted for any vector of multipliers, but success is most likely for λ^*.

Theorem 3. *If for a given vector of multipliers* $\lambda = (\lambda_1, \ldots, \lambda_m)$, *we have for some* J_k *and* M_h *that*

$$(\lambda_h p_{hk} - \min_{1 \leq i \leq m} \lambda_i p_{ik}) / \sum_{i=1}^{m} \lambda_i > UB - v(S_\lambda) - 1,$$

where UB is a given upper bound on $v(\mathrm{P})$, *then* $x_{hk} = 0$ *in any schedule with* $C_{\max} < UB$, *if such a schedule exists.*

Proof. Suppose there is a schedule with makespan less than UB, and yet with J_k scheduled on M_h. Solving the surrogate relaxation problem S_λ under the additional constraint $x_{kl} = 1$ gives the lower bound LB with

$$LB = \frac{\lambda_h p_{hk} + \sum_{j=1; j \neq k}^{n} \min_i \lambda_i p_{ij}}{\sum_{i=1}^{m} \lambda_i} = \frac{(\lambda_h p_{hk} - \min_i \lambda_{ik}) + \sum_{j=1}^{n} \min_i \lambda_i p_{ij}}{\sum_{i=1}^{m} \lambda_i} > UB - 1,$$

which is clearly a contradiction. □

4.2. The search tree

A node at level k of the search tree corresponds to a partial schedule with a specific assignment of the jobs $J_1, \ldots, J_k$. Each node at level k ($k = 1, \ldots, n-1$) has at most m descendant nodes: one node for the assignment of job J_{k+1} to machine M_i, for $i = 1, \ldots, m$, respectively. We note that the jobs and machines have been reindexed in compliance with the branching rule that we discuss in the next subsection. The algorithm we propose is of the 'depth-first' type. We employ an *active node* search: at each level we choose one node to branch-from, thereby adding some job to the partial schedule. We backtrack if we reach the bottom of the tree or if the active node can be discarded on the basis of the lower bound computed or on the basis of reductions made.

4.3. Branching rule

The dual processing times $\lambda_i^* p_{ij}$, ($i = 1, \ldots, m$, $j = 1, \ldots, n$) serve as a guideline to structure the branch-and-bound tree. Define $\gamma_j = \min 2_{1 \leqslant i \leqslant m} \lambda_i^* p_{ij} - \min_{1 \leqslant i \leqslant m} \lambda_i^* p_{ij}$, where min2 denotes the second minimum of the dual processing times. As Potts and Lenstra et al. pointed out, it follows from linear programming theory that, in case of the optimal dual multiplier, we have for at least $m-1$ jobs that $\gamma_j = 0$; in the dual solution these jobs can equally well be scheduled on at least two machines. On the other hand, some jobs will have a relatively large value γ_j, with $0 < \gamma_j < UB - v(S) - 1$. From a dual point of view, a large value γ_j is an indication for the existence of some optimal primal solution with J_j scheduled on the machine with minimum dual processing time. After all, in view of Theorem 3, there is some value UB that fixes J_j to this machine. In order to enhance the possibility that we find improved values UB soon, we wish, therefore, to enumerate first the various configurations in which the jobs with a comparatively large value γ_j are assigned to their most likely machines. This is obtained if we reindex the jobs in order of non-decreasing values γ_j. As a result of our branching rule, the jobs associated with the upper levels of the search tree are then assigned to their most likely machines.

In addition, at each level k ($k = 1, \ldots, n-1$) the descendant nodes are branched from in order of non-decreasing values $\lambda_i^* p_{i,k+1}$ ($i = 1, \ldots, m$). In this fashion, we obtain the same effect as above: we first consider the schedules with jobs assigned to machines according to minimum dual processing times.

By sorting jobs and machines with the help of the dual processing times we try to reduce the search as much as possible. Suppose we find an improved upper bound which entails additional reductions. If we have created the tree as indicated, then the most newly fathomed nodes have not been considered yet.

Note that the first complete schedule encountered in the tree solves the dual problem. Furthermore, implicit enumeration of the bottom $m-1$ levels of the tree concurs with Potts' 2-approximation algorithm. The $n-m+1$ jobs with $\gamma_j > 0$ are assigned according to the linear programming solution (i.e., according to minimum dual processing times), and the remaining configurations are enumerated.

4.4.Fathoming

Here we describe in detail the various rules by which nodes can be fathomed. Surprisingly enough, it appeared from computational experience that it is not worthwile to recompute the lower bound $v(S)$ in each node of the tree. Rather, we use the optimal vector of dual multipliers $(\lambda_1^*, \ldots, \lambda_m^*)$ from the root node for lower bound computation in the descendant nodes. Suppose the values z_{ij} $(i = 1, \ldots, m, j = 1, \ldots, k)$ record the current partial schedule at level k of the tree. That is, $z_{ij} = 1$ if J_j has been assigned to M_i, and $z_{ij} = 0$ otherwise. Let $v(S_\lambda \cdot, k)$ denote the lower bound value for problem $S_\lambda \cdot$ subject to $x_{ij} = z_{ij}$ for $i = 1, \ldots, m$, $j = 1, \ldots, k$. Then we have

$$v(S_\lambda \cdot, k) = v(S_\lambda \cdot) + \sum_{j=1}^{k} (\sum_{i=1}^{m} (\lambda_i^* p_{ij} z_{ij} - \min_{1 \leqslant i \leqslant m} \lambda_i^* p_{ij})) / \sum_{i=1}^{m} \lambda_i^*.$$

Note that $v(S_\lambda \cdot, k) \geqslant v(S_\lambda \cdot) = v(S)$. Hence, a node at level k that assigns J_k to machine M_h can be fathomed if

$$(\lambda_h^* p_{hk} - \min_{1 \leqslant i \leqslant m} \lambda_i^* p_{ik}) / \sum_{i=1}^{m} \lambda_i^* > UB - v(S_\lambda \cdot, k-1) - 1. \tag{F1}$$

Note that this test requires constant time per node of the tree. In addition, the corresponding node can be fathomed if

$$\sum_{j=1}^{k-1} p_{hj} z_{hj} + p_{hk} > UB - 1. \tag{F2}$$

The third test tries to establish whether the current partial schedule is dominated by another partial schedule for the same k jobs. Suppose we have some job J_l $(1 \leqslant l \leqslant k-1)$ that is currently scheduled on M_i for which

$$p_{il} > p_{ik} \text{ and } p_{hl} < p_{hk}. \tag{F3}$$

Interchanging J_l and J_k would reduce the load of both M_i and M_h. The current partial can then be discarded, since there is at least one optimal schedule in which there is no pair of jobs for which property (F3) holds.

Conditions similar to (F2) can be checked for every job J_j $(j = k+1, \ldots, n)$. In case there is a job J_l $(k+1 \leqslant l \leqslant n)$ for which

$$\sum_{j=1}^{k} p_{ij} z_{ij} + p_{il} > UB - 1 \quad \text{for each } M_i,\ i = 1, \ldots, m, \tag{F4}$$

we fathom the node, too. Similarly, if condition (F4) applies to some J_l $(k+1 \leqslant l \leqslant n)$ for all machines M_i $(i = 1, \ldots, m)$ but one, we can assign J_l to this machine. Subsequently, we can possibly carry out additional assignments, which enhances the likelihood that the node can be closed on account of (F1), (F2), (F3), or (F4).

In addition, we try to identify a machine M_h $(1 \leqslant h \leqslant m)$ for which

$$\sum_{j=1}^{l} p_{hj} z_{hj} + p_{hl} > UB - 1$$

for each J_l, $l = k+1, \ldots, n$. In this case, M_h can be ignored for the assignment of any

remaining job. Therefore, we can discard the node if

$$\left[\sum_{j=1}^{k}\sum_{i=1;i\neq h}^{m}\lambda_i^* p_{ij}z_{ij} + \sum_{j=k+1}^{n}\min_{1\leqslant i\leqslant m;\, i\neq h}\lambda_i^* p_{ij}\right] / \sum_{i=1;i\neq h}^{m}\lambda_i^* > UB - 1.$$

5. Computational experiments

Both algorithms have been coded in the computer language C; the experiments were conducted on a Compaq-386 personal computer.

The algorithms were tested on a broad range of instances with n and m varying from 20 to 200 and from 2 to 20, respectively, giving rise to 80 combinations altogether. The processing times were generated from the uniform distribution [10,100]. For each combination of n and m we considered 10 instances.

5.1. The branch-and-bound algorithm

For the branch-and-bound algorithm we put an upper bound of 100,000 nodes; computation for a particular instance was discontinued at this limit. In Table 2 we present for each combination the number of unresolved problems. An empty cell indicates that the branch-and-bound algorithm was not run, as it could be expected from adjacent cells or initial computations that most of the instances would be prematurely terminated. Table 3 shows the average number of nodes explored. The average for a particular combination of n and m is computed by aggregating the number of nodes for each of its corresponding instances and dividing the sum by 10, the total number of instances for each combination. Note that prematurely terminated instances contribute 100,000 nodes each to the average number of nodes. Table 4 presents the average computation time for the branch-and-bound algorithm in addition to the running time for the heuristics and the duality-based approximation algorithm. The time spent on unfinished instances is included, too. The average computation time for a particular combination is computed in a similar fashion as the average number of nodes.

The general impression is that instances with a few machines are relatively easy. In addition, the required effort to solve a problem seems to increase more with the number of machines than with the number of jobs. However, surprising exceptions to this statement form the instances with $m \geqslant 12$ and $n \leqslant 40$. Note, in addition, that the 100,000-node limit on the branch-and-bound algorithm is quite arbitrary, as it induces distinct *time* limits across the instances. For example, it appears that instances with $m = 20$ and $n = 50$ or 60 are solvable within about ten thousand nodes on the average, but requiring approximately 5 minutes running time. From Table 4, however, one can easily form some idea which instances are within reach with, say, one minute of computation time.

Significant deviations from the averages may occur. For example, a single instance for the combination $n = 30$ and $m = 15$ accounts for the remarkably high figure in Table 3 and considerable computation time in Table 4. Finally, it is not unconceivable that the performance of the algorithm is enhanced by fine-tuning the algorithm to particular instances. For example, it may turn out to be worthwile to recompute the optimal vector of dual multipliers in each node of the tree after all for large values of n and m. Even in that case, however, such instances are not solvable within reasonable time limits.

$n\downarrow\ m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	1	2	0	0	0
50	0	0	0	0	1	5	-	-	-	0
60	0	0	0	0	1	-	-	-	-	1
80	0	0	0	2	-	-	-	-	-	-
100	0	0	0	3	-	-	-	-	-	-
200	0	1	0	-	-	-	-	-	-	-

TABLE 2. Number of unresolved problems out of 10 for each cell.

$n\downarrow\ m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	16	46	68	203	180	75	33	37	11	0
30	31	90	340	752	434	1440	784	145	4784	64
40	37	170	615	4488	10149	6786	23936	3800	192	342
50	59	171	1188	6133	16022	48202	-	-	-	5848
60	68	358	1127	12715	27942	-	-	-	-	10669
80	85	1232	3386	37110	-	-	-	-	-	-
100	132	2503	5198	28116	-	-	-	-	-	-
200	330	12245	14274	-	-	-	-	-	-	-

TABLE 3. Average number of nodes.

$n\downarrow\ m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	1	1	1	1	1	1	1	1	1	1
30	1	1	1	2	2	9	6	2	43	2
40	1	1	2	12	39	39	214	63	3	10
50	1	1	3	16	57	285	-	-	-	204
60	1	1	3	33	105	-	-	-	-	373
80	1	3	8	96	-	-	-	-	-	-
100	1	6	12	87	-	-	-	-	-	-
200	3	40	52	-	-	-	-	-	-	-

TABLE 4. Average computation time in seconds.

5.2. The duality-based approximation algorithm

In the implementation of the duality-based approximation algorithm we have put *maxiter* $= n \cdot m$. Note furthermore that cycling may occur. This happens, for instance, under the following conditions: J_j is scheduled on M_1, but can equally well be assigned to M_2, and we have $C_1^t = C_{\max}^t$, and $C_2^{t+1} = C_{\max}^{t+1}$. In such a situation, J_j would oscillate between M_1 and M_2. The procedure is discontinued once this phenomenon is detected.

The duality-based approximation algorithm was compared with the constructive heuristics of De and Morton (1980), Ibarra and Kim (1977), Davis and Jaffe (1981), and with Potts' 2-approximation algorithm (Potts, 1985). Note that the two-phase heuristics presented by Hariri and Potts (1990) are dominated by Potts' 2-approximation algorithm. It turned out that the constructive heuristics have a very erroneous behavior. For instance, the De and Morton heuristic, which takes the best result from 10 underlying heuristics, produces solutions which deviate 27.0% on the average from the best solution found. We have therefore treated the constructive heuristics as a single algorithm by only considering the best result.

In Table 5, we present the average proportional deviation for the best schedule generated by the constructive heuristics from the optimal solution, or if this is not available, from the best known solution. In the latter case, brackets have been placed around the figures. Table 6 shows the same information for the duality-based approximation algorithm.

As a whole, the duality-based approximation algorithm shows a better performance than the constructive heuristics, which behave poorly. This certainly applies to instances with a larger number of machines. The performance of the constructive heuristics can easily be enhanced by submitting them to an iterative local improvement scheme. Therefore, the schedules generated by the constructive heuristics should merely be seen as initial solutions that serve as input for some iterative local improvement procedure.

Hence, each schedule generated by the constructive heuristics was submitted to the iterative local improvement procedure that tries to reduce makespan by job interchanges. The procedure has been described in detail in Section 3. In contrast, only the best schedule generated by the duality-based approximation algorithm was submitted to the improvement procedure.

In Tables 7, 8, and 9 we present the results for the constructive heuristics, Potts' 2-approximation algorithm, and the duality-based approximation algorithm after local improvement, respectively. The sign '*' behind an entry in these tables indicates that the corresponding algorithm has the best average performance for the associated instances. From Table 7 it can been seen that the iterative local improvement technique is powerful for the constructive heuristics in case of few machines or jobs. However, its power deteriorates with increasing number of machines, as only two machines at a time are involved in job interchanges. In that sense, it is hard to find a way to an attractive local neighborhood, even in case of multiple start solutions. Generally, the running time, which seems to be increasing with n, is quite modest: instances up to $n = 100$ require only one or two seconds, while in case of $n = 200$ approximately 10 seconds of computation time are required. Because the job interchanges stay limited to two machines at a time, the number of machines hardly seems to play a role in the computation time.

Potts' 2-approximation algorithm was embedded in the branch-and-bound algorithm as described in Section 4 so as to assign the $m-1$ fractional jobs by impicit enumeration, given the partial schedule for the other $n-m+1$ jobs. The branch-and-bound algorithm was

$n\downarrow$ $m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	2.9	8.0	6.8	12.8	19.0	20.3	7.6	22.6	8.9	5.4
30	2.2	6.3	8.2	18.0	18.8	25.5	22.5	23.1	14.6	13.0
40	3.0	6.5	10.8	14.6	13.3	(27.5)	(27.4)	25.6	28.6	19.0
50	2.0	7.2	12.0	12.1	(19.3)	(23.4)	(17.2)	(18.4)	(14.7)	32.8
60	1.4	6.0	10.3	10.5	(15.9)	(14.6)	(17.2)	(15.4)	(16.9)	(42.5)
80	1.8	4.6	8.4	(11.1)	(13.4)	(11.4)	(16.9)	(17.5)	(24.7)	(20.5)
100	2.8	3.3	7.2	(9.7)	(11.1)	(15.0)	(19.8)	(18.3)	(19.8)	(21.2)
200	0.7	(2.4)	3.9	(4.2)	(5.0)	(8.2)	(15.5)	(17.0)	(15.5)	(24.5)

TABLE 5. Proportional deviation for the constructive heuristics.

$n\downarrow$ $m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	4.2	5.4	6.6	10.5	11.3	16.4	15.2	14.6	7.4	3.0
30	1.5	4.9	6.0	9.8	8.9	14.7	16.7	21.0	14.0	15.2
40	1.9	4.2	3.5	9.0	8.3	(10.0)	(19.0)	14.4	13.5	19.3
50	1.6	3.3	4.9	7.4	(6.5)	(8.3)	(4.1)	(1.9)	(2.5)	18.1
60	1.2	1.1	4.1	5.5	(5.0)	(4.0)	(1.8)	(1.0)	(1.8)	(24.3
80	1.4	2.3	2.9	(3.5)	(2.4)	(1.9)	(2.1)	(2.8)	(2.7)	(4.5)
100	2.3	2.3	2.4	(3.6)	(1.8)	(2.2)	(1.9)	(1.9)	(0.8)	(1.4)
200	0.4	(1.5)	1.1	(1.1)	(1.2)	(1.8)	(3.3)	(1.3)	(3.3)	(0.6)

TABLE 6. Proportional deviation for the duality-based approximation algorithm.

adjusted at two points: we omitted dominance rule (F3) and and we initially put $UB = \infty$. Condition (F3) is useful if our aim is to find an optimal solution, but might cut off good approximate solutions. As a result of this, it occasionally happened that Potts' algorithm took more time than the optimization algorithm. It is surprising that the final solution was rarely improved by the local improvement procedure, which was applied to all the jobs. The computational effort for the algorithm was modest and seemed to increase more with the number of machines than with the number of jobs. For instances up to $m = 12$, it took one or two seconds, but for $m = 15$ and $m = 20$ it required about 15 to 20 seconds on the average. The instance $n = 20$ and $m = 20$ was not run because Potts' algorithm would require explicit enumeration of almost the entire state space.

As can be seen from the number of '*' signs in Table 9, the duality-based approximation algorithm has the best performance on the average. Note that the entries for $m = 2$ are identical for the duality-based algorithm and Potts' algorithm. It is remarkable that the duality-based

$n\downarrow\ m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	0.0*	1.0*	3.0*	7.1	8.3*	10.4	4.9*	11.1	1.9*	5.1
30	0.1*	1.2*	3.1	4.4*	7.6	13.3*	15.0	17.4*	11.0	9.2
40	0.2*	1.3	1.7*	3.8*	7.9	(13.6)	(15.4)	17.8	22.2	14.7*
50	0.3*	1.1*	2.7*	5.2	(7.8)	(11.6)	(5.0)	(6.9)	(7.0)	20.4
60	0.2*	1.0	3.1	3.8	(5.9)	(4.8)	(2.5)	(7.4)	(10.6)	(32.8)
80	0.1*	0.9	2.3	(2.8)	(1.9)	(1.9)	(4.7)	(6.7)	(12.5)	(12.4)
100	2.5	0.7	1.9	(2.9)	(1.7)	(2.1)	(6.1)	(6.1)	(6.1)	(10.6)
200	0.2	(0.6)	1.1	(0.8)	(1.3)	(1.8)	(3.5)	(4.1)	(3.5)	(8.7)

TABLE 7. Proportional deviation for the constructive heuristics after iterative local improvement.

$n\downarrow\ m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	1.7	3.0	5.4*	9.1	11.0	8.9*	10.8	14.2	4.8	-
30	0.2	2.7*	4.6	7.9	6.9	13.4	12.7*	22.0	10.4*	3.7*
40	0.4	2.1	2.8	5.3	8.7	(13.6)	(17.6)	19.5	20.6	15.1
50	0.4	1.6	3.3	5.6	(7.6)	(11.8)	(8.6)	(6.6)	(6.2)	21.6
60	0.3	2.6	2.8	5.1	(6.7)	(1.9)	(3.1)	(10.1)	(4.2)	(37.0)
80	0.1*	1.9	2.2	(5.7)	(4.0)	(3.4)	(9.5)	(7.2)	(10.3)	(12.4)
100	2.3*	1.7	1.9	(4.4)	(3.1)	(2.6)	(3.9)	(3.9)	(9.3)	(10.6)
200	0.1*	(0.8)	1.0	(1.7)	(2.4)	(3.9)	(5.0)	(6.3)	(5.0)	(14.7)

TABLE 8. Proportional deviation for Potts' 2-approximation algorithm after iterative local improvement.

$n\downarrow\ m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	1.7	1.0*	5.4	5.4*	9.8	14.5	14.0	10.8*	7.4	3.0*
30	0.2	2.6	3.9	5.2	6.7*	14.2	15.0	19.7	11.3	14.8
40	0.4	1.0*	2.8	5.2	4.4*	(9.3)*	(15.2)*	13.9*	12.0*	16.4
50	0.4	1.5	2.3*	4.1*	(4.2)*	(7.2)*	(1.3)*	(0.0)*	(1.2)*	17.0*
60	0.3	0.5*	2.0*	2.7*	(3.5)	(1.0)*	(0.8)*	(0.8)*	(0.9)*	(22.8)*
80	0.1*	0.7*	1.0*	(1.9)*	(1.8)*	(0.6)*	(0.6)*	(2.2)*	(1.6)*	(4.5)*
100	2.3*	0.6*	1.1*	(2.2)*	(0.8)*	(0.9)*	(0.7)*	(0.7)*	(0.3)*	(0.7)*
200	0.1*	(0.5)*	0.7*	(0.3)*	(0.1)*	(0.8)*	(1.1)*	(0.4)*	(1.2)*	(0.0)*

TABLE 9. Proportional deviation for the duality-based approximation algorithm after iterative local improvement.

approximation algorithm performs considerably better than Potts' algorithm in spite of their close relationship. Perhaps more information can be drawn from Table 10, which presents the number of times (out of 10) that the duality-based approximation algorithm produced the best or equally best solution. It appears that the algorithm performs remarkably well if m and n are large; apparently, these instances are mostly beyond the reach of the iterative local improvement procedure and Potts' 2-approximation algorithm. In a sense, the duality-based approximation algorithm and the branch-and-bound algorithm are supplementary: the latter is applicable to instances for which the former yields to other algorithms. The running time is about a factor two more than the running time of the constructive heuristics and Potts' approximation algorithm, but it seems to be comparable or less in the extreme combinations with $n = 200$ or $m = 20$.

$n \downarrow$ $m \rightarrow$	2	3	4	5	6	8	10	12	15	20
20	1	7	5	7	4	4	3	6	7	9
30	7	4	4	4	7	6	4	7	4	4
40	5	6	3	3	8	9	4	7	9	5
50	7	4	5	7	7	6	6	10	8	9
60	4	8	7	6	8	4	5	9	8	9
80	2	5	7	6	9	6	8	6	9	7
100	9	4	5	6	5	6	8	8	9	9
200	6	7	6	7	9	7	8	8	8	10

TABLE 10. Number of times (out of 10) the duality-based approximation algorithm performed at least as good as the other approximation algorithms.

6. CONCLUSIONS

The $R \mid\mid C_{max}$ problem is a highly practical scheduling problem for which we have proposed a branch-and-bound and an approximation algorithm. The branch-and-bound algorithm manages to solve relatively large instances to optimality within reasonable time limits. The approximation algorithm is based upon a simple, intuitively appealing but effective idea for local search: heuristic duality-based search in conjunction with iterative local improvement. Measured from computational experiments, it performs best in comparison with other approximation algorithms for instances that are beyond the reach of an optimization algorithm.

ACKNOWLEDGEMENT

The author would like to thank Jan Karel Lenstra for his helpful comments on an earlier draft of this paper.

REFERENCES

E. DAVIS AND J.M. JAFFE (1981). Algorithms for scheduling tasks on unrelated parallel processors. *Journal of the Association of Computing Machinery 28*, 721-736.

P. DE AND T.E. MORTON (1980). Scheduling to minimize makespan on unequal parallel processors. *Decision Sciences 11*, 586-603.

M.L. FISHER (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science 27*, 1-18.

A.M. GEOFFRION (1974). Duality in nonlinear programming: a simplified applications-oriented development. *SIAM Review 13*, 1-37.

F. GLOVER (1989). Tabu search - Part I. *ORSA Journal on Computing 1*, 190-206.

T. GONZALEZ, E.L. LAWLER, AND S. SAHNI (1990). Optimal preemptive scheduling of two unrelated processors. To appear in *ORSA Journal on Computing*.

H.J. GREENBERG, W.P. PIERSKALLA (1970). Surrogate mathematical programming. *Operations Research 18*, 924-939.

R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA AND A.H.G. RINNOOY KAN (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annual Discrete Mathematics 5*, 287-326.

A.M.A HARIRI AND C.N. POTTS (1990). *Heuristics for scheduling unrelated parallel machines*, Working Paper, Faculty of Mathematical Studies, University of Southampton.

E. HOROWITZ AND S. SAHNI (1976). Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the Association of Computing Machinery 23*, 317-327.

O.H. IBARRA AND C.G. KIM (1977). On heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the Association of the Computing Machinery 24*, 280-289.

M.H. KARWAN AND R.L. RARDIN (1979). Some relationships between Lagrangian and surrogate duality in integer programming. *Mathematical Programming 17*, 320-334.

P.J.M. VAN LAARHOVEN AND E.H.L. AARTS (1987). *Simulated Annealing: Theory and Applications*, Reidel, Dordrecht.

J.K. LENSTRA, D.B. SHMOYS AND E. TARDOS (1990). Approximation algorithms for scheduling unrelated parallel machines. To appear in *Mathematical Programming*.

C.H. PAPADIMITRIOU AND K. STEIGLITZ (1982). *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

C.N. POTTS (1985). Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics 10*, 155-164.

H.I. STERN (1976). *Minimizing makespan for independent jobs on nonidentical machines - an optimal procedure*, Working Paper 2/75, Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva.

D. DE WERRA AND A. HERTZ (1989). Tabu search techniques: a tutorial and an application to neural networks. *OR Spektrum 11*, 131-141.