Performance guarantees of local search for multiprocessor scheduling

Performance Guarantees of Local Search for Multiprocessor Scheduling

Petra Schuurman
Hagenkampweg Noord 13, 5616 TC Eindhoven, The Netherlands, petraschaakt@xs4all.nl
Tjark Vredeveld
Maastricht University, Department of Quantitative Economics, P.O.Box 616, 6200 MD Maastricht, The Netherlands, t.vredeveld@ke.unimaas.nl

Increasing interest has recently been shown in analyzing the worst-case behavior of local search algorithms. In particular, the quality of local optima and the time needed to find the local optima by the simplest form of local search has been studied. This paper deals with worst-case performance of local search algorithms for makespan minimization on parallel machines. We analyze the quality of the local optima obtained by iterative improvement over the jump, swap, multi-exchange, and the newly defined push neighborhoods. Finally, for the jump neighborhood we provide bounds on the number of local search steps required to find a local optimum.

Key words: production-scheduling: multiple machine; approximation heuristics; local search; analysis of algorithms
History: Accepted by William Cook, Area Editor for Design and Analysis of Algorithms; received September 2003; revised December 2004; accepted May 2005.

---

1. Introduction

Local search methods exhibit good empirical behavior, but little is known about their worst-case performance. In this paper, we analyze the quality of several local optima from such a worst-case perspective. Moreover, for the simplest form of local search, we make some remarks about the time needed to obtain a local optimum. The problems under consideration are multiprocessor scheduling problems, where we are given a set of n jobs, $J_1, \ldots, J_n$, each of which has to be processed without preemption on one of m machines, $M_1, \ldots, M_m$. A machine can process at most one job at a time and all jobs and machines are available at time 0. The objective we consider is *makespan minimization*, i.e., we want the last job to complete as early as possible. The time, p_{ij}, it takes for a job J_j to be fully processed on a machine M_i depends on the machine environment:

- Identical parallel machines, denoted by P: a job has the same processing time on all machines, i.e., $p_{ij} = p_j$, where p_j is a positive integer.

- Uniform parallel machines, denoted by Q: the machines have positive integral speeds, $s_1, \ldots, s_m$, and each job has a given positive integral processing requirement p_j; the processing time is $p_{ij} = p_j/s_i$.

- Unrelated parallel machines, denoted by R: the time it takes to process job J_j on machine M_i is dependent on the machine as well as the job; p_{ij} is a positive integer.

If the number of machines is part of the input, Graham et al. (1979) denote these problems by $P\|C_{\max}$, $Q\|C_{\max}$, and $R\|C_{\max}$. If the number of machines is a constant m, then they denote the problems by $Pm\|C_{\max}$, etc.

Even the simplest case, $P2\|C_{\max}$, is NP-hard (Garey and Johnson 1979), so we search for approximate solutions. If an algorithm is guaranteed to deliver a solution that has value at most ρ times the optimal solution value, we call it a *ρ-approximation* algorithm. ρ is called the *worst-case performance guarantee*. For $P\|C_{\max}$ and $Q\|C_{\max}$, Hochbaum and Shmoys (1987, 1988) develop so-called polynomial-time approximation schemes, i.e., for each $\epsilon > 0$, there exists a polynomial-time $(1+\epsilon)$-approximation algorithm. For $R\|C_{\max}$, Lenstra et al. (1990) present a polynomial-time 2-approximation algorithm; they also prove that there does not exist a polynomial-time $(\frac{3}{2} - \epsilon)$-algorithm for $\epsilon > 0$, unless P=NP.

A way to find approximate solutions is through *local search*. Local search methods iteratively search through the set of feasible solutions. Starting from an initial solution, a local search procedure moves from a feasible solution to a neighboring solution until some stopping criteria are met. The choice of a suitable neighborhood function has an important influence on the performance of local search. We analyze the performance of local search for the jump, the swap, the multi-exchange, and the newly defined push neighborhood from a worst-case perspective.

The simplest form of local search is *iterative improvement*, also called local improvement or, in the case of minimization problems, descent algorithms. This method iteratively chooses a better solution in the neighborhood of the current solution and it stops when no better solution is found; we say that the current solution is a *local optimum*.

We analyze the quality of some local optima and the time needed to find some of them. Previous results on worst-case analysis of local search include the following. In the area

of facility location we refer to Korupolu et al. (2000), Charikar and Guha (1999), Chudak and Williamson (2005), Pál et al. (2001), and Arya et al. (2004). In Hurkens and Schrijver (1989), Arkin and Hassin (1998), and De Bontridder et al. (2003), set-packing problems are considered and local search algorithms for the metric labeling problem are studied in Boykov et al. (1999) and Gupta and Tardos (2000). Lu and Ravi (1992) and Brüggemann et al. (2003) consider some spanning-tree problems, and Vredeveld and Lenstra (2003) show that the simplest form of local search can be arbitrarily bad for a generalized form of graph coloring. Improvements by local search of the Goemans and Williamson (1995) solution for the max-cut problem are given in Feige et al. (2002) and for complexity aspects we refer to Ausiello and Protasi (1995). In the area of scheduling, the first worst-case analysis of local search is Finn and Horowitz (1979), who give a guarantee on the quality of what we call jump optimal solutions and they also claim a bound on the time needed to obtain such a local optimum by iterative improvement. Hurkens and Vredeveld (2003) present an example showing that this bound is incorrect. Brucker et al. (1996, 1997) give the correct time bound.

The rest of this paper is organized as follows. In the following section we discuss the neighborhoods and in Section 3 we establish performance guarantees for the various local optima and scheduling problems. In Section 4, we remark on the running time for iterative improvement to obtain the local optima, and finally we make some concluding remarks.

2. Neighborhoods

Before discussing the neighborhoods, we first describe our representation of a schedule. As the sequence in which the jobs are processed does not influence the makespan of a schedule for a given assignment of the jobs to the machines, we represent a schedule by such an assignment. This is equivalent to a partitioning of the set of jobs into m disjoint subsets $\mathcal{J}_1, \ldots, \mathcal{J}_m$, where $\mathcal{J}_i$ is the set of jobs scheduled on M_i. The *load* of a machine is the total processing time of its jobs. A *critical machine* is a machine with maximum load.

The first neighborhood that we consider is the *jump* neighborhood, also known as the *move* neighborhood: we select a job J_j and a machine M_i on which job J_j is not scheduled. The neighbor is obtained by moving J_j to M_i, as shown in Figure 1. We say that we are in a *jump-optimal solution* if no jump decreases the makespan or the number of critical machines without increasing the makespan.

For a *swap neighbor*, we select two jobs, J_j and J_k, scheduled on different machines. A

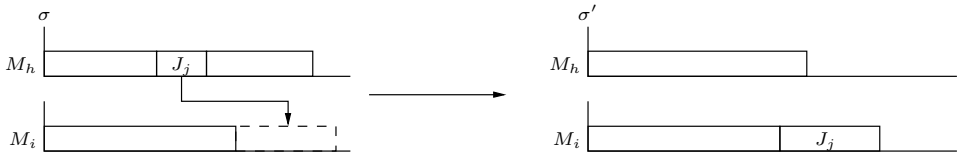

Figure 1: Jump

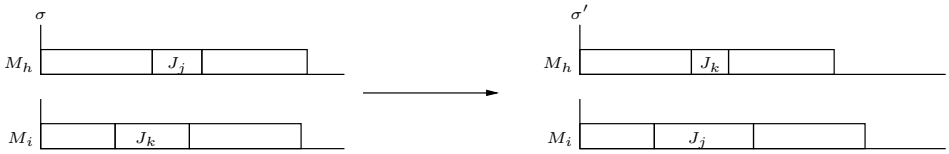

Figure 2: Swap

swap neighbor is formed by interchanging the machine allocations of the jobs (Figure 2). If all jobs are scheduled on the same machine, then no swap neighbor exists; therefore, we define the swap neighborhood as one that consists of all possible jumps and all possible swaps. A *swap-optimal solution* is one in which no swap or jump decreases the makespan or the number of critical machines without increasing the makespan.

Based on Ahuja et al. (2001), Frangioni et al. (2004) define so-called multi-exchange neighborhoods. For a *cyclic-exchange neighbor*, we select a sequence of jobs $C = (J_{j_1}, \ldots, J_{j_k})$ such that all jobs are assigned to different machines. The neighbor is obtained by moving job J_{j_l} to the machine on which $J_{j_{l+1}}$ is currently scheduled ($l = 1, \ldots, k-1$) and moving J_{j_k} to the machine of J_{j_1}. For a *path-exchange neighbor* we are given a sequence $P = (J_{j_1}, \ldots, J_{j_k}, M_i)$ of jobs and one machine M_i such that no two jobs are scheduled on the same machine and none of them is assigned to machine M_i. The neighbor is obtained in the same way as in the cyclic exchange neighborhood, except that job J_{j_k} is moved to machine M_i. Note that jump and swap are special cases of the path and cycle exchanges in which the length of the sequence is restricted to two. If there are only two machines, these multi-exchange neighborhoods are equivalent to the jump and swap neighborhood, due to the restriction that all jobs in the sequence need to be scheduled on different machines. In contrast to the jump and swap neighborhood, the number of path and cyclic-exchange neighbors can grow exponentially with respect to the problem size. Moreover, Frangioni et al. showed that deciding whether there exists an improving cyclic or path-exchange neighbor is NP-complete. We say that a schedule is *multi-exchange optimal* if no cyclic or path exchange decreases the makespan or the number of critical machines without increasing the makespan.

As we will show in the next section, the above-defined neighborhoods have no constant performance guarantee for uniform machines. Therefore, we introduce a *push* neighborhood, for which any local optimum is at most a factor $2-\frac{2}{m+1}$ of optimal for the problem on uniform parallel machines. The push neighborhood is a form of variable-depth search, introduced by Kernighan and Lin (1970) for graph partitioning and the traveling-salesman problem (Lin and Kernighan 1973). A push is a sequence of jumps.

Starting with a schedule $\sigma = (\mathcal{J}_1, \ldots, \mathcal{J}_m)$ having makespan $C_{\max}(\sigma)$, a push is initiated by selecting a job J_k on a critical machine and a machine M_i to which to move it. We say that J_k *fits* on M_i if $\sum_{J_j \in \mathcal{J}_i : p_{ij} \geq p_{ik}} p_{ij} + p_{ik} < C_{\max}(\sigma)$. If J_k does not fit on any machine, then it cannot be pushed. If, after moving J_k to M_i, the load of this machine is at least as large as the original makespan, i.e., if $\sum_{J_j \in \mathcal{J}_i} p_{ij} + p_{ik} \geq C_{\max}(\sigma)$, then we iteratively remove the smallest job from M_i until the load of M_i is less than $C_{\max}(\sigma)$. The removed jobs are gathered in a queue. We now have a queue of pending jobs and a partial schedule that has lower makespan or fewer critical machines. If the queue is non-empty, then the largest job in the queue is removed and moved to some machine on which it fits, in the same way as the first job was pushed. Thus, if necessary, we allow some smaller jobs to be removed. If the largest job in the queue does not fit on any machine, then we say that the push is *unsuccessful*. We repeat the procedure of moving the largest job in the queue to a machine until the queue is empty or we have determined that the push is unsuccessful. When pushing all jobs on the critical machines is unsuccessful, we are in a *push-optimal solution*.

We illustrate a push in the following example.

Example 1 Consider $m = 3$ identical machines with $n = 8$ jobs. The processing times are $p_1 = 8$, $p_2 = p_3 = p_4 = 6$, $p_5 = p_6 = 5$, $p_7 = 3$, and $p_8 = 2$. The starting schedule is $\sigma = (\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3)$, with $\mathcal{J}_1 = \{J_2, J_5, J_6\}$, $\mathcal{J}_2 = \{J_1, J_7, J_8\}$, and $\mathcal{J}_3 = \{J_3, J_4\}$, and has makespan $C_{\max}(\sigma) = 16$ (Figure 3a). In the first step of the push, we select job J_6, with $p_6 = 5$, to be pushed onto machine M_2. Note that J_6 does not fit on M_3. When moving J_6 to M_2, jobs J_8 and J_7 have to be removed from M_2 (Figure 3b). At this point, we have a partial schedule $\sigma' = (\mathcal{J}'_1, \mathcal{J}'_2, \mathcal{J}'_3)$, $\mathcal{J}'_1 = \{J_2, J_5\}$, $\mathcal{J}'_2 = \{J_1, J_6\}$, $\mathcal{J}'_3 = \{J_3, J_4\}$, and a queue of pending jobs containing J_7 and J_8. In the next step, we remove J_7 from the queue and move it to M_1 and then we move J_8 to M_3. After moving J_8, the queue is empty and we have a new schedule $\bar{\sigma} = (\bar{\mathcal{J}}_1, \bar{\mathcal{J}}_2, \bar{\mathcal{J}}_3)$, with $\bar{\mathcal{J}}_1 = \{J_2, J_5, J_7\}$, $\bar{\mathcal{J}}_2 = \{J_1, J_6\}$, and $\bar{\mathcal{J}}_3 = \{J_3, J_4, J_8\}$, which has makespan $C_{\max}(\bar{\sigma}) = 14$ (Figure 3c).

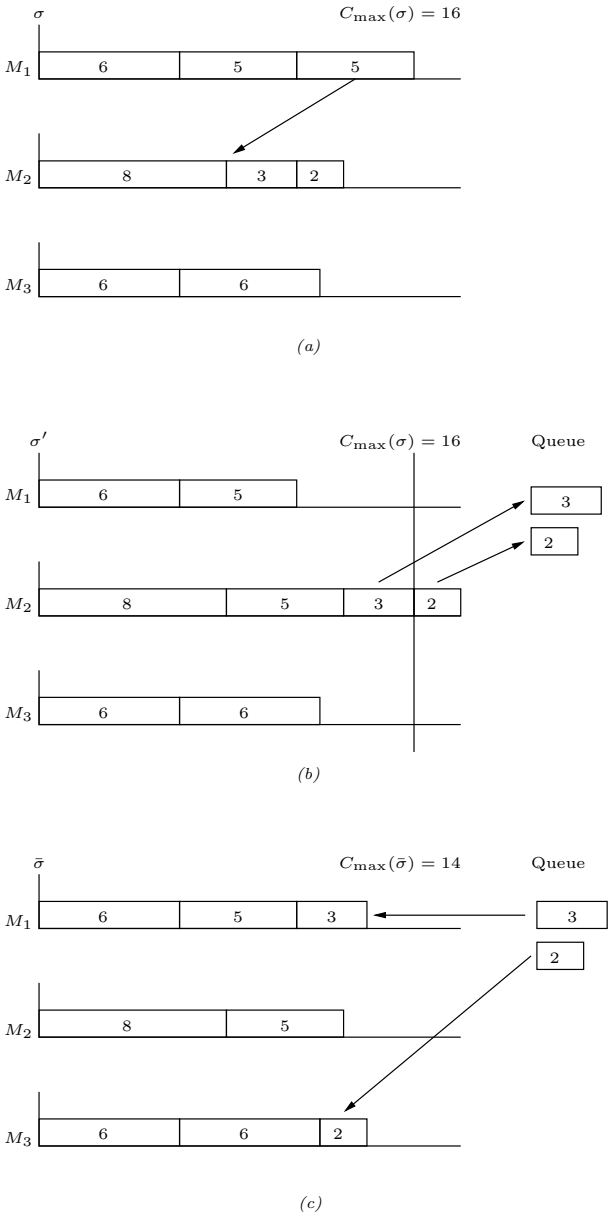

Figure 3: Push

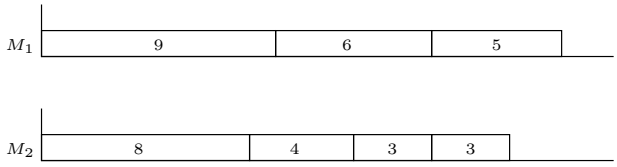

Figure 4: Push Optimal Schedule

A push-optimal solution is given in Example 2. As the schedule in this example is not swap-optimal, it shows that a push-optimal solution is not necessarily swap-optimal. Of course, as a push is a sequence of one or more jumps, a push-optimal solution is jump optimal.

Example 2 In Figure 4, we give an example for $m = 2$ identical machines. There are $n = 7$ jobs, $J_1, \ldots, J_7$, and the processing times are $p_1 = 9$, $p_2 = 8$, $p_3 = 6$, $p_4 = 5$, $p_5 = 4$, and $p_6 = p_7 = 3$. The schedule $\sigma = (\mathcal{J}_1, \mathcal{J}_2)$, with $\mathcal{J}_1 = \{J_1, J_3, J_4\}$ and $\mathcal{J}_2 = \{J_2, J_5, J_6, J_7\}$, is push-optimal. When trying to push J_1, jobs J_7, J_6, and J_5 are moved to the queue of pending jobs. Then jobs J_5 and J_6 are moved to machine M_1, resulting in a partial schedule with a load of 18 for M_1 and of 17 for M_2. The queue of pending jobs consists of job J_7 of length $p_7 = 3$. As $18 + 3 \geq 20 = C_{\max}(\sigma)$ and $17 + 3 \geq 20 = C_{\max}(\sigma)$, J_7 does not fit on M_1 as well as on M_2. Hence, pushing J_1 is unsuccessful. In the same manner, we see that pushing J_3 or J_4 is also unsuccessful. The schedule σ can be improved by swapping e.g. J_1 and J_2, and thus it is not swap-optimal.

By our way of defining a push, we know that when moving a job J_k, only jobs smaller than J_k can be removed from the machine. Hence, during one push, at most n jobs need to be moved. As one move can straightforwardly be implemented such that it needs $\mathcal{O}(n)$ time, a push requires $\mathcal{O}(n^2)$ elementary operations. If we use appropriate data structures, like binary heaps for the queue of pending jobs and for the list of machines and doubly linked lists for the jobs, and if we select the machine to which to move a job in a greedy manner, a push neighbor can be found in $\mathcal{O}(n \log n)$ time.

Note that, as we always take the largest job from the queue, the push neighborhood is defined only for scheduling problems where the largest job is defined unambiguously. Therefore, in the case of unrelated parallel machines, a push is not well defined.

Table 1: Guarantees on Ratio Between Value of Local Optimum and Optimal Value; † Upper Bound due to Finn and Horowitz (1979), ‡ Upper Bound due to Cho and Sahni (1980)

	jump	swap and multi-exchange	push
$P2\|\|C_{\max}$	$\frac{4}{3}^{\dagger}$	$\frac{4}{3}^{\dagger}$	$\frac{8}{7}$
$P\|\|C_{\max}$	$2-\frac{2}{m+1}^{\dagger}$	$2-\frac{2}{m+1}^{\dagger}$	UB $=\frac{4}{3}-\frac{1}{3m}$ LB $=\frac{4m}{3m+1}$
$Q2\|\|C_{\max}$	$\frac{1+\sqrt{5}}{2}^{\ddagger}$	$\frac{1+\sqrt{5}}{2}^{\ddagger}$	$\frac{\sqrt{17}+1}{4}$
$Q\|\|C_{\max}$	$\frac{1+\sqrt{4m-3}}{2}^{\ddagger}$	$\frac{1+\sqrt{4m-3}}{2}^{\ddagger}$	UB $=2-\frac{2}{m+1}$ LB $=\frac{3}{2}-\epsilon$
$R2\|\|C_{\max}$	LB $=\frac{p_{\max}}{C^*_{\max}}$	LB $=n-1$	undefined
$R\|\|C_{\max}$	LB $=\frac{p_{\max}}{C^*_{\max}}$	LB $=\frac{p_{\max}}{C^*_{\max}}$	undefined

3. Performance Guarantees

In this section, we establish performance guarantees for the various local optima and scheduling problems, given in Table 1. "UB $=\rho$" denotes that ρ is a performance guarantee and "LB $=\rho$" denotes that the performance guarantee cannot be less than ρ; "ρ" denotes that UB $=\rho$ and LB $=\rho$. For the unrelated-parallel-machines cases, we use $p_{\max}=\max_{i,j} p_{ij}$.

In the following subsection we prove the performance guarantees for the identical-parallel-machines cases, and in the subsequent two subsections we consider the cases of uniform and unrelated parallel machines, respectively. The value of an optimal schedule is denoted by $C^*_{\max}$, and $C^{\mathrm{J}}_{\max}$, $C^{\mathrm{S}}_{\max}$, $C^{\mathrm{M}}_{\max}$, and $C^{\mathrm{P}}_{\max}$ denote the makespans of respectively a jump-optimal, swap-optimal, multi-exchange-optimal, and push-optimal schedule, respectively.

3.1 Identical Parallel Machines

Recall that in the identical-parallel-machine environment, all machines need the same amount of time for a job. Hence the processing time of a job J_j on a machine M_i is $p_{ij}=p_j$. Finn and Horowitz (1979) showed the following result for the jump neighborhood.

Theorem 1 (Finn and Horowitz 1979) *A jump-optimal schedule for $P\|C_{\max}$ has makespan at most $2-\frac{2}{m+1}$ times the optimal solution value.*

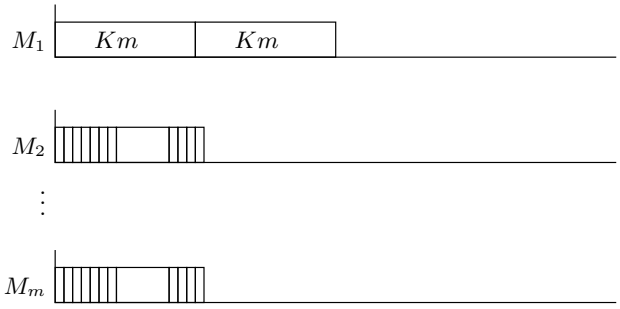

Figure 5: Swap Optimal Schedule

Corollary 1 *A swap- or multi-exchange-optimal schedule for $P||C_{\max}$ has makespan at most $2 - \frac{2}{m+1}$ times the optimal makespan.*

The bounds given in Theorem 1 and Corollary 1 are tight, as can be seen in the following example.

Example 3 For given $K > 1$, consider the instance in which there are m machines and $n = 2 + (m-1)(Km+1)$ jobs. The processing times of the jobs are $p_1 = p_2 = Km$ and $p_j = 1$ $(j = 3, \ldots, n)$. The optimal makespan is obtained by a schedule in which jobs J_1 and J_2 are processed on different machines and the other jobs are divided over the machines such that the loads of any two machines differ by at most 1. The optimal makespan is $C^*_{\max} = K(m+1)+1$.

In Figure 5, a swap- and multi-exchange-optimal schedule is given: jobs J_1 and J_2 are both processed on machine M_1 and the other jobs are equally divided over machines $M_2, \ldots, M_m$. It is easy to see that this schedule is multi-exchange-optimal as the only improvement can be obtained by moving J_1 or J_2. However, moving J_1 or J_2 to any other machine does not decrease the makespan, as by definition of the neighborhood no two jobs scheduled on the same machine may be moved. The value of this schedule is at least $C^{\mathrm{S}}_{\max} = C^{\mathrm{M}}_{\max} = 2Km = (\frac{2m}{m+1+1/K})C^*_{\max}$ and for large values of K, the ratio $C^{\mathrm{M}}_{\max}$ / $C^*_{\max}$ is close to $2 - \frac{2}{m+1}$. For a jump-optimal schedule, we can even remove one job from each of the machines $M_2, \ldots, M_m$. The value of the optimum is then $C^*_{\max} = K(m+1)$ and the jump-optimal schedule remains $C^{\mathrm{J}}_{\max} = 2Km = (2 - \frac{2}{m+1})C^*_{\max}$.

For a push-optimal schedule, we can prove a better performance guarantee.

Theorem 2 *A push-optimal schedule for the problem on identical parallel machines is at most $\frac{4}{3} - \frac{1}{3m}$ times the optimal makespan.*

Proof: Consider a push-optimal schedule with makespan $C^{\mathrm{P}}_{\max}$, and consider pushing a job which is currently scheduled on a critical machine. This leads to an unsuccessful push. Hence, there will be a partial schedule, $(\mathcal{J}'_1, \ldots, \mathcal{J}'_m)$ and a non-empty queue of pending jobs. Let job J_k be the largest job in this queue. Then, we know that this job does not fit on any machine, i.e.,

$$L'_i + p_k \geq C^{\mathrm{P}}_{\max},$$

where $L'_i = \sum_{j \in \mathcal{J}'_i : p_j \geq p_k} p_j$. Summing this inequality over all machines leads to

$$C^{\mathrm{P}}_{\max} \leq \frac{1}{m}(\sum_i L'_i + m p_k) \leq \frac{1}{m}\sum_j p_j + \frac{m-1}{m} p_k \leq C^*_{\max} + \frac{m-1}{m} p_k. \tag{1}$$

If $p_k \leq \frac{1}{4} C^{\mathrm{P}}_{\max}$, then rewriting (1) yields:

$$C^{\mathrm{P}}_{\max} \leq \frac{4m}{3m+1} C^*_{\max} \leq \frac{4m-1}{3m} C^*_{\max}.$$

If $p_k > \frac{1}{4} C^{\mathrm{P}}_{\max}$, there are at most three large jobs, i.e., at least as large as job J_k, scheduled on each machine. We will show that in this case either the push-optimal schedule is globally optimal or there exists a machine in the optimal schedule that processes at least three large jobs. For the latter, we use $C^*_{\max} \geq 3p_k$ in (1) and obtain

$$C^{\mathrm{P}}_{\max} \leq C^*_{\max} + \frac{m-1}{m} p_k \leq \frac{4m-1}{3m} C^*_{\max}.$$

A push-optimal schedule is globally optimal if, in the partial schedule, there is a machine without any large job ($C^{\mathrm{P}}_{\max} \leq p_k \leq C^*_{\max}$) or if, in the partial schedule, there exists a machine M_i containing exactly one large job J_j that is processed in the optimal schedule on a machine containing another large job, i.e., $C^*_{\max} \geq p_j + p_k \geq L'_i + p_k \geq C^{\mathrm{P}}_{\max}$. Hence, if a push-optimal schedule is not globally optimal, then each machine in the partial schedule contains at least one large job, and the number of machines processing two or three large jobs in the optimal schedule is not more than this number for the partial schedule. As job J_k is not assigned to any machine in the partial schedule, we know by the pigeon-hole principle that in the optimal schedule there exists at least one machine processing three large jobs. □

In contrast to the jump- and swap-optimal solutions, we have no tightness guarantees. The following example shows that the performance guarantee for push-optimal schedules for identical machines cannot be less than $\frac{4m}{3m+1}$. The gap between the upper and lower bound is $\frac{m-1}{3m(3m+1)}$, which is at most $\frac{1}{45}$ for $m \geq 3$.

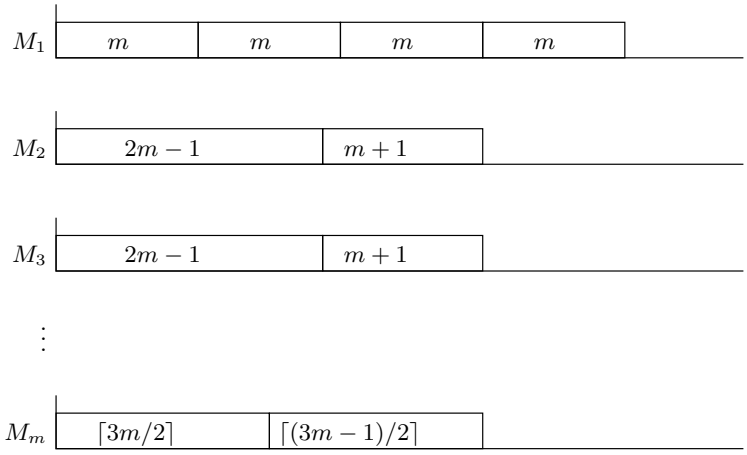

Figure 6: Push Optimal Schedule

Example 4 Consider m machines and $n = 2m+2$ jobs. The processing times of jobs $J_1, \ldots, J_{n-4}$ are $p_j = m + \lceil \frac{j}{2} \rceil$ for $j = 1, \ldots, n-4$, i.e., there are exactly two jobs of size p for $p = m+1, \ldots, 2m-1$. The processing times of the last four jobs are $p_{n-3} = p_{n-2} = p_{n-1} = p_n = m$. The optimal makespan is $C^*_{\max} = 3m+1$ and is attained by $\sigma^* = (\mathcal{J}^*_1, \ldots, \mathcal{J}^*_m)$, with $\mathcal{J}^*_1 = \{J_1, J_{n-3}, J_{n-2}\}$, $\mathcal{J}^*_2 = \{J_2, J_{n-1}, J_n\}$, and $\mathcal{J}^*_i = \{J_i, J_{n-1-i}\}$ for $i = 3, \ldots, m$.

In the schedule in Figure 6 the jobs of size $p_j = m$ are assigned to M_1 and M_i processes J_{i-1} and J_{n-i-2}, for $i = 2, \ldots, m$. This is push-optimal with makespan $C^{\mathrm{P}}_{\max} = 4m = \frac{4m}{3m+1} C^*_{\max}$.

If there are only two identical parallel machines, we can prove a tight performance guarantee of 8/7.

Theorem 3 *A push-optimal solution for $P2\|C_{\max}$ has value at most 8/7 times the optimal solution value.*

Proof: Suppose, to the contrary, that there exists a push-optimal schedule with makespan $C^{\mathrm{P}}_{\max} > \frac{8}{7} C^*_{\max}$. Let $L_i = \sum_{J_j \in \mathcal{J}_i} p_j$ be the load of machine M_i $(i = 1, 2)$. W.l.o.g. we assume that $L_1 \geq L_2$, thus $C^{\mathrm{P}}_{\max} = L_1$.

For the difference in loads of the two machines, we know that

$$L_1 - L_2 = L_1 - (\sum_j p_j - L_1) \geq 2L_1 - 2C^*_{\max} > \frac{1}{4} L_1.$$

The first inequality is due to the lower bound $C^*_{\max} \geq \frac{1}{2} \sum_j p_j$ and the second inequality is due to the assumption that $L_1 > \frac{8}{7} C^*_{\max}$. Let J_1 be the smallest job on M_1. Then, by push

optimality, we know that $p_1 \geq L_1 - L_2 > \frac{1}{4}L_1$. Hence, there are at most three jobs on M_1. We assume that M_1 processes three jobs, i.e., $\mathcal{J}_1 = \{J_1, J_2, J_3\}$, $\mathcal{J}_2 = \{J_4, \ldots, J_n\}$, and that $p_1 \leq p_2 \leq p_3$ and $p_4 \geq \cdots \geq p_n$. Let J_k be defined by the following two inequalities.

$$\begin{aligned} p_4 + \cdots + p_{k-1} + p_1 &< L_1, \\ p_4 + \cdots + p_k + p_1 &\geq L_1. \end{aligned} \tag{2}$$

Informally, this means that the smaller of job J_1 and J_k is to be removed from machine M_2 when pushing job J_1 to M_2.

We claim that our assumptions imply that $L_2 < 3p_1$ and $p_k \geq p_1$. Hence, M_2 processes at most two jobs of size greater than or equal to p_k.

To show the first claim, note that push optimality implies that $p_1 \geq L_1 - L_2 > \frac{1}{4}L_1$. Moreover, from $L_1 - L_2 > \frac{1}{4}L_1$ it also follows that $L_2 < \frac{3}{4}L_1 < 3p_1$, which shows our claim.

Second, from $L_1 - L_2 > \frac{1}{4}L_1 \geq \frac{3}{4}p_1$ we know that $L_2 + p_1 < L_1 + \frac{1}{4}p_1$ and

$$\frac{1}{4}p_1 > L_2 + p_1 - L_1 \overset{(2)}{\geq} L_2 - (p_4 + \cdots + p_k) = p_{k+1} + \cdots + p_n. \tag{3}$$

Because of push optimality, we know that $p_1 \leq p_k + \cdots + p_n$ and thus

$$p_k \geq p_1 - (p_{k+1} + \cdots + p_n) \overset{(3)}{>} \frac{3}{4}p_1. \tag{4}$$

Suppose that $p_k < p_1$, then by pushing J_1 to M_2, J_k is moved to M_1 and jobs $J_{k+1}, \ldots, J_n$ are distributed among M_1 and M_2 yielding a schedule with makespan

$$C'_{\max} \leq \max(L_1 - p_1 + p_k,\ L_2 + p_1 - p_k) \overset{(3),(4)}{<} \max(L_1, L_1 - \frac{1}{2}p_1) = L_1.$$

Thus the schedule is not push-optimal and it must be the case that $p_1 \leq p_k$.

If M_2 processes only one job of size at least p_k, i.e., $k = 4$, then the current schedule is optimal, as the sub-schedule for J_1, J_2, J_3, J_4 is optimal, because by (2) $p_1 + p_4 \geq p_1 + p_2 + p_3$.

In the case that M_2 processes two jobs of size at least p_k, i.e., $p_4 \geq p_5 = p_k$, we consider two sub-cases: $p_3 \leq p_5$ and $p_3 > p_5$. If $p_3 \leq p_5$, then the sub-schedule for $J_1, \ldots, J_5$ is optimal and $C^*_{\max} \geq C^*_{\max}[1,5] = L_1 = C^{\mathrm{P}}_{\max}$, where $C^*_{\max}[1,5]$ denotes the makespan of an optimal sub-schedule for $J_1, \ldots, J_5$.

If $p_3 > p_5$, then an optimal schedule for $J_1, \ldots, J_5$ has value $C^*_{\max}[1,5] \geq p_1 + p_2 + p_5$. As $L_2 < 3p_1$, we know that $p_4 < 2p_1$ and by push optimality we know that $p_5 + \sum_{j \geq 6} p_j \geq p_3$. Hence,

$$\frac{C^{\mathrm{P}}_{\max}}{C^*_{\max}} \leq \frac{p_1 + p_2 + p_3}{p_1 + p_2 + p_5} \leq \frac{p_1 + p_2 + p_5 + \ldots + p_n}{p_1 + p_2 + p_5} \overset{(3)}{\leq} 1 + \frac{1/4p_1}{3p_1} = \frac{13}{12} < \frac{8}{7}.$$

This contradicts the assumption that $C^{\mathrm{P}}_{\max} > \frac{8}{7}C^*_{\max}$.

If M_1 processes only two jobs, i.e., $\mathcal{J}_1 = \{J_1, J_2\}$ and $\mathcal{J}_2 = \{J_3, \ldots, J_n\}$, we can prove in a similar way that $C^*_{\max} = C^{\mathrm{P}}_{\max}$. If M_1 only processes J_1, then $C^{\mathrm{P}}_{\max} = C^*_{\max}$.

Hence, $C^{\mathrm{P}}_{\max} \leq \frac{8}{7}C^*_{\max}$. □

The bound in Theorem 3 is tight: if we set $m = 2$ in Example 4, we have a push-optimal schedule with makespan $C^{\mathrm{P}}_{\max} = \frac{8}{7} C^*_{\max}$.

3.2 Uniform Parallel Machines

In this environment, jobs have processing requirements p_j, for $j = 1, \ldots, n$, and machines have speeds s_i, for $i = 1, \ldots, m$. The processing time of job J_j on machine M_i is $p_{ij} = p_j/s_i$.

Lemma 1 of Cho and Sahni (1980) shows that the quality of a schedule for uniform parallel machines, obtained by list scheduling is no worse than $\frac{1}{2}(1 + \sqrt{4m-3})$. Using the same techniques, we are able to prove the same guarantee for jump-optimal schedules in the uniform-parallel-machines environment.

Theorem 4 (Cho and Sahni 1980) *A jump-optimal schedule for $Q\|C_{\max}$ has makespan at most $\frac{1+\sqrt{4m-3}}{2}$ times the optimal makespan.*

Corollary 2 *A swap- or multi-exchange-optimal schedule for $Q\|C_{\max}$ has makespan at most $\frac{1+\sqrt{4m-3}}{2}$ times the optimum.*

For $m \geq 3$, Cho and Sahni show a better guarantee than $\frac{1}{2}(1+\sqrt{4m-3})$ for list schedules. Example 5 shows that we cannot hope for a better guarantee for a jump-, swap-, or multi-exchange-optimal solution.

Example 5 For given integral $s > 1$, there are $n = m + 1$ jobs and $m = s^2 - s + 1$ machines, i.e., $s = \frac{1+\sqrt{4m-3}}{2}$. Job J_1 has processing requirement $p_1 = s$ and all other jobs have processing requirement $p_j = 1$ $(j = 2, \ldots, n)$. Machine M_1 has speed $s_1 = s$ and all other machines have speed $s_i = 1$ $(i = 2, \ldots, m)$. In an optimal schedule, machine M_1 processes job J_1 and one job of unit length and each of machines $M_2, \ldots, M_m$ processes exactly one job of unit length. The optimal makespan is $C^*_{\max} = 1 + \frac{1}{s}$.

In Figure 7 a multi-exchange-optimal schedule is given. The second machine processes job J_1 and machine M_1 processes all other jobs. The makespan is $C^{\mathrm{M}}_{\max} = s = \frac{s}{1+1/s} C^*_{\max}$ and for large s the ratio $C^{\mathrm{M}}_{\max} / C^*_{\max}$ is close to $s = \frac{1+\sqrt{4m-3}}{2}$. For a jump-optimal schedule,

we can even remove one of the unit-sized jobs, so that the optimum has value $C^*_{\max} = 1$ and the value of the local optimal solution remains $C^{\mathrm{J}}_{\max} = s = \frac{1+\sqrt{4m-3}}{2} C^*_{\max}$.

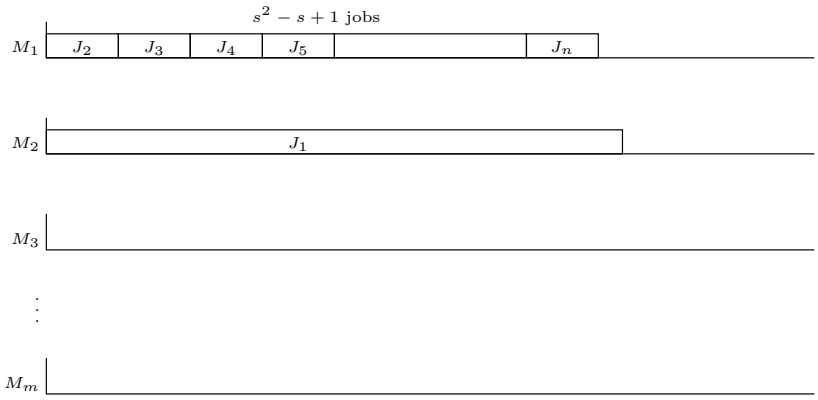

Figure 7: Swap Optimal Schedule for $Q\|C_{\max}$.

Corollary 3 *The performance guarantee of jump-, swap-, and multi-exchange-optimal schedules for $Q2\|C_{\max}$ is $\frac{1+\sqrt{5}}{2}$ and this is tight.*

Proof: The performance guarantee is a direct consequence of Theorem 4. To see that the bound is tight, consider $n = 2$ jobs with processing requirements $p_1 = \frac{1+\sqrt{5}}{2}$ and $p_2 = 1$, and with speeds $s_1 = \frac{1+\sqrt{5}}{2}$ and $s_2 = 1$ for machines M_1 and M_2 respectively. Obviously, the optimal makespan is $C^*_{\max} = 1$ and the schedule in which J_1 is processed by M_2 and J_2 is scheduled on M_1 is a jump-optimal schedule with makespan $C^{\mathrm{J}}_{\max} = \frac{1+\sqrt{5}}{2}$.

This schedule clearly is not swap-optimal. To make a swap-optimal one, we chop job J_2 of unit length into $\frac{1}{\epsilon}$ jobs with processing requirements $p_j = \epsilon$, for some $\epsilon > 0$ such that $1/\epsilon$ is integral, and we add one job of size ϵ. The optimal makespan is $C^*_{\max} = 1 + \frac{2\epsilon}{1+\sqrt{5}}$ and the schedule in which all jobs of size ϵ are scheduled on M_1 and J_1 is processed by M_2 is swap-optimal, with makespan $C^{\mathrm{S}}_{\max} = \frac{1+\sqrt{5}}{2}$. Hence, for small ϵ the ratio $C^{\mathrm{S}}_{\max}$ / $C^*_{\max}$ is close to $\frac{1+\sqrt{5}}{2}$. □

Theorem 5 *A push-optimal schedule for $Q\|C_{\max}$ has makespan at most $2 - \frac{2}{m+1}$ times the optimal solution value.*

Proof: Assume w.l.o.g. that $s_1 \geq \cdots \geq s_m$. If we consider an unsuccessful push, then there is a partial schedule, $(\mathcal{J}'_1, \ldots, \mathcal{J}'_m)$, and a queue of pending jobs. The largest job in this

queue does not fit on any machine. Let job J_k be this job and let $L'_i = \sum_{j \in \mathcal{J}'_i : p_j \geq p_k} \frac{p_j}{s_i}$ be the total processing time of the large jobs on machine M_i, i.e., at least as large as J_k. By push optimality, we know that for all $i = 1, \ldots, m$,

$$L'_i + \frac{p_k}{s_i} \geq C^{\mathrm{P}}_{\max}. \tag{5}$$

Let M_h be the slowest machine on which job J_k has a processing time that is not larger than the optimal makespan, i.e., $h = \max\{i : \frac{p_k}{s_i} \leq C^*_{\max}\}$. Thus $C^*_{\max} \geq \frac{p_k}{s_h}$. As all large jobs need to be scheduled on machines $M_1, \ldots, M_h$, a second lower bound on the optimal makespan is

$$C^*_{\max} \geq \frac{\sum_{j:p_j \geq p_k} p_j}{\sum_{i=1}^{h} s_i}.$$

By push optimality (5), we know that

$$\sum_{i=1}^{h} s_i C^{\mathrm{P}}_{\max} \leq \sum_{i=1}^{h} s_i L'_i + h p_k \leq \sum_{j:p_j \geq p_k} p_j + (h-1)p_k. \tag{6}$$

If $s_1 \geq 2s_h$, then $\sum_{i=1}^{h} s_i \geq (h+1)s_h$ and rearranging the terms in (6) yields

$$C^{\mathrm{P}}_{\max} \leq C^*_{\max} + \frac{h-1}{h+1}\frac{p_k}{s_h} \leq C^*_{\max} + \frac{h-1}{h+1} C^*_{\max} \leq \frac{2m}{m+1} C^*_{\max}.$$

The second inequality is due to our choice of h. If $s_1 \leq 2s_h$, then $\sum_{i=1}^{h} s_i \geq \frac{h+1}{2} s_1$. We may assume that $C^*_{\max} \geq \frac{2p_k}{s_1}$, as otherwise there are at most h large jobs and the push-optimal schedule is optimal. Rearranging terms in (6) yields

$$C^{\mathrm{P}}_{\max} \leq C^*_{\max} + \frac{h-1}{(h+1)/2}\frac{p_k}{s_1} \leq \frac{2m}{m+1} C^*_{\max}.$$

□

Theorem 6 *The performance guarantee of push-optimal schedules for $Q\|C_{\max}$ is at least $\frac{3}{2} - \epsilon$, for $\epsilon > 0$.*

Proof: Consider the following instance for $Q\|C_{\max}$. For given $r \in (\frac{2}{3}, 1)$, there are $m = \lceil \log_r(\frac{3r-2}{2r-1}) \rceil$ machines and $n = m + 1$ jobs. The speeds of the machines are given by $s_1 = 2$ and $s_i = r s_{i-1} + 1$, $i = 2, \ldots, m$. The processing requirements are given by $p_j = r s_j$ for $j = 1, \ldots, n-2$ and $p_{n-1} = p_n = 1$. As by our choice of m we have that $\frac{2}{s_m} \leq r$, and the optimal makespan is $C^*_{\max} \leq r$.

The schedule in Figure 8 is push-optimal: M_1 processes both jobs of size 1, and J_j is scheduled on M_{j+1} for $j = 1, \ldots, m-1$. This schedule has makespan $C^{\mathrm{P}}_{\max} = 1 \geq \frac{1}{r} C^*_{\max}$. Hence, for any $\epsilon > 0$ there exists a push-optimal schedule with $C^{\mathrm{P}}_{\max} \geq (\frac{3}{2} - \epsilon) C^*_{\max}$. □

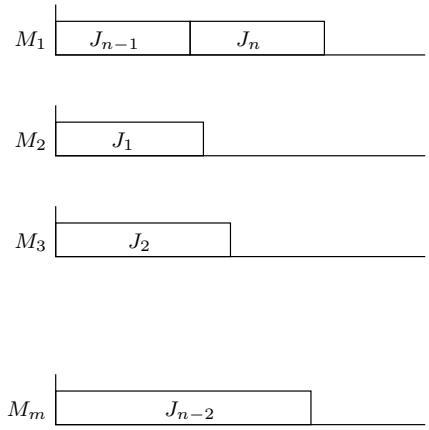

Figure 8: Push Optimal Schedule for $Q\|C_{\max}$

In the case of two uniform machines, we establish a better performance guarantee. To do so, we need the following two lemmata, in which we consider instances with only three jobs and different speeds for the two machines. We prove in Theorem 7 that a smallest worst-case instance for push for the problem of scheduling on two uniform parallel machines has exactly three jobs.

Lemma 1 *Consider an instance for $Q2\|C_{\max}$ with three jobs in which the machines do not have the same speed. Assume w.l.o.g that $p_1 \geq p_2 \geq p_3$ and $s_1 > s_2$. If, in an optimal schedule, J_1 is processed on M_1, then in any push-optimal schedule a job of size p_1 is scheduled on M_1 and this push-optimal schedule is globally optimal.*

Proof: Suppose to the contrary that there is a push-optimal schedule in which J_1 is processed on M_2. Then J_2 is scheduled on M_1, as otherwise a push is possible.

If $p_1 = p_2$, then the first part of the lemma is proved. Consider the case that $p_1 > p_2$. If M_2 is the critical machine, then J_1 can be pushed, and the schedule is not push-optimal. Therefore, M_1 is the critical machine and it processes J_2 as well as J_3. In the optimal schedule, J_1 is assigned to M_1 and the optimal makespan has value $C^*_{\max} \geq \min\{\frac{p_1+p_3}{s_1}, \frac{p_2+p_3}{s_2}\}$. As $\frac{p_1+p_3}{s_1} > \frac{p_2+p_3}{s_1} = C^{\mathrm{P}}_{\max}$ and $\frac{p_2+p_3}{s_2} > \frac{p_2+p_3}{s_1} = C^{\mathrm{P}}_{\max}$, we have $C^*_{\max} > C^{\mathrm{P}}_{\max}$, which is a contradiction. Therefore, in a push-optimal schedule, J_1 must be processed by M_1, whenever J_1 is scheduled on M_1 in an optimal schedule.

By enumerating over all possible schedules with J_1 scheduled on M_1 for the push-optimal schedule as well as the optimal schedule, it is easy to see that whenever such a schedule is push-optimal it is globally optimal. □

Lemma 2 *Consider an instance for $Q2\|C_{\max}$ with three jobs in which the machines do not have equal speed. Assume w.l.o.g. that $p_1 \geq p_2 \geq p_3$ and $s_1 > s_2$. If $C^{\mathrm{P}}_{\max} > C^*_{\max}$, then in the optimal schedule M_1 processes J_2 and J_3, and J_1 is scheduled on M_2. In a push-optimal schedule with $C^{\mathrm{P}}_{\max} > C^*_{\max}$, the machine allocation of the jobs is reversed, that is, J_1 is scheduled on M_1, and M_2 processes J_2 and J_3. This push-optimal schedule has makespan $C^{\mathrm{P}}_{\max} = \frac{p_2+p_3}{s_2}$.*

Proof: By Lemma 1, we know that whenever $C^{\mathrm{P}}_{\max} > C^*_{\max}$, in the optimal schedule J_1 is scheduled on M_2. As $\frac{p_1+p_2}{s_2} \geq \frac{p_1+p_3}{s_2} > \frac{p_2+p_3}{s_1}$, J_2 as well as J_3 is assigned to M_1 in the optimal schedule.

If, in a push-optimal schedule, J_1 is processed by M_2, then this schedule must be globally optimal. Hence, for each push-optimal schedule with $C^{\mathrm{P}}_{\max} > C^*_{\max}$, J_1 is scheduled on M_1 and the critical machine is M_2, as otherwise J_1 can be pushed. If J_2 or J_3 are also scheduled on M_1, then M_2 cannot be critical and the schedule is not push-optimal. Therefore, a push-optimal schedule with makespan $C^{\mathrm{P}}_{\max} > C^*_{\max}$ processes J_1 on M_1, and J_2 and J_3 on M_2, and as M_2 is the critical machine, $C^{\mathrm{P}}_{\max} = \frac{p_2+p_3}{s_2}$. □

Theorem 7 *A push-optimal schedule for $Q2\|C_{\max}$ has performance guarantee $\frac{\sqrt{17}+1}{4}$.*

Proof: Consider a push-optimal schedule with $C^{\mathrm{P}}_{\max} > \frac{5}{4}C^*_{\max}$. We may assume that such a schedule exists as otherwise $C^{\mathrm{P}}_{\max}/C^*_{\max} \leq \frac{5}{4} < \frac{\sqrt{17}+1}{4}$. Pushing the smallest job which is currently scheduled on the critical machine leads to an unsuccessful push. Hence, there is a largest job in the queue of pending jobs that does not fit on both machines. Let this job be J_k. Note that this job is at most as large as the smallest job on the critical machine. Because of push optimality, we now have

$$\sum_{J_j \in \mathcal{J}_i : p_j \geq p_k} \frac{p_j}{s_i} + \frac{p_k}{s_i} \geq C^{\mathrm{P}}_{\max}, \qquad i = 1, 2.$$

Thus,

$$(s_1 + s_2)C^{\mathrm{P}}_{\max} \leq \sum_{j : p_j \geq p_k} p_j + p_k \leq (s_1 + s_2)C^*_{\max} + p_k.$$

By the assumption that $C^{\mathrm{P}}_{\max} > \frac{5}{4}C^*_{\max}$, we have that $\sum_j p_j \leq (s_1 + s_2)C^*_{\max} < 4p_k$. Hence, there are at most three large jobs, i.e., at least as large as J_k.

If we remove all jobs that are smaller than J_k from the push-optimal schedule, then we still have a push-optimal schedule and the makespan has not changed, as all the jobs that

are smaller than J_k are scheduled on the non-critical machine. As the optimal makespan of the instance with only the large jobs is at most equal to the optimal makespan of the original instance, the smallest worst-case instance consists of only those (at most three) large jobs.

Any push-optimal schedule on an instance with at most two jobs is an optimal schedule, and therefore the worst-case instance for the ratio $C^{\mathrm{P}}_{\max}/C^{*}_{\max}$ consists of three jobs. Consider such a worst-case instance, and assume w.l.o.g. that $p_1 \geq p_2 \geq p_3$ and that $s_1 > s_2$. Note that if $s_1 = s_2$, then we actually have two identical parallel machines and by Theorem 3 we know that $C^{\mathrm{P}}_{\max}/C^{*}_{\max} \leq \frac{8}{7}$.

Consider a worst-case instance, and assume w.l.o.g. that $p_1 \geq p_2 \geq p_3$ and that $s_1 > s_2$. By Lemma 2, we know that in this worst-case push-optimal schedule, J_1 is scheduled on M_1, and M_2 processes J_2 and J_3. We also know that $C^{\mathrm{P}}_{\max} = \frac{p_2+p_3}{s_2}$ and $C^{*}_{\max} = \max\{\frac{p_1}{s_2}, \frac{p_2+p_3}{s_1}\}$.

By push optimality, we know that $\frac{p_1+p_3}{s_1} \geq \frac{p_2+p_3}{s_2}$, and thus $C^{\mathrm{P}}_{\max}/C^{*}_{\max}$ is bounded by

$$C^{\mathrm{P}}_{\max}/C^{*}_{\max} = \min\{\frac{s_1}{s_2}, \frac{p_2+p_3}{p_1}\} \leq \min\{\frac{p_1+p_3}{p_2+p_3}, \frac{p_2+p_3}{p_1}\}.$$

This minimum is maximal, when $\frac{p_1+p_3}{p_2+p_3} = \frac{p_2+p_3}{p_1}$. Then $(p_2+p_3)^2 = p_1^2 + p_1p_3$ and thus

$$C^{\mathrm{P}}_{\max}/C^{*}_{\max} \leq \frac{\sqrt{p_1^2+p_1p_3}}{p_1} = \sqrt{1+\frac{p_3}{p_1}}. \tag{7}$$

As $p_2 \geq p_3$, we know that $p_1^2 + p_1p_3 = (p_2+p_3)^2 \geq 4p_3^2$ and, thus $p_1 \geq \frac{\sqrt{17}-1}{2}p_3$. Using this bound in (7) yields

$$C^{\mathrm{P}}_{\max}/C^{*}_{\max} \leq \sqrt{1+\frac{2}{\sqrt{17}-1}} = \frac{\sqrt{17}+1}{4}.$$

□

In the following example, we have an instance for $Q2\|C_{\max}$ and a push-optimal schedule for which $C^{\mathrm{P}}_{\max} = \frac{\sqrt{17}+1}{4}C^{*}_{\max}$.

Example 6 Consider the following instance with three jobs: $p_1 = \frac{\sqrt{17}-1}{2}$, $p_2 = p_3 = 1$, and $s_1 = \frac{\sqrt{17}+1}{4}$ and $s_2 = 1$. In the optimal schedule, M_1 processes J_2 and J_3, and J_1 is scheduled on M_2. The optimal makespan is $C^{*}_{\max} = \frac{\sqrt{17}-1}{2}$. The schedule in which J_1 is processed by M_1 and J_2 and J_3 are scheduled on M_2 is push-optimal with makespan $C^{\mathrm{P}}_{\max} = 2$, and $C^{\mathrm{P}}_{\max}/C^{*}_{\max} = 2/(\frac{\sqrt{17}-1}{2}) = \frac{4}{\sqrt{17}-1} = \frac{\sqrt{17}+1}{4}$.

3.3 Unrelated Parallel Machines

In the unrelated-parallel-machine environment, the processing times are job- and machine-dependent, i.e., the processing time of job J_j on machine M_i is p_{ij}. The maximum processing time is $p_{\max} = \max_{i,j} p_{ij}$.

Theorem 8 *The performance guarantee of jump-optimal schedules for $R\|C_{\max}$ cannot be smaller than $p_{\max}/C^*_{\max}$*

Proof: For given $K > 1$ consider n jobs and $m = n$ machines. The processing times of the jobs are

$$p_{ij} = \begin{cases} 1 & \text{if} \quad i = j, \\ K & \text{otherwise.} \end{cases}$$

In the optimal schedule, machine M_i processes job J_i, and this schedule has makespan $C^*_{\max} = 1$. The schedule in which machine M_1 processes job J_n and machine M_i processes job J_{i-1} $(i = 2, \ldots, m)$ is jump-optimal and has makespan $C^{\mathrm{J}}_{\max} = K$. □

As the above example also is a jump-optimal schedule in the case of only two machines, we have the following corollary.

Corollary 4 *A jump-optimal schedule for $R2\|C_{\max}$ has a performance guarantee that is not better than $p_{\max}/C^*_{\max}$.*

For the environment with identical and uniform parallel machines, a jump-optimal schedule with ratio $\rho = C^{\mathrm{J}}_{\max}/C^*_{\max}$ can be converted into a swap-optimal schedule with the same ratio ρ, as in the proof of Corollary 3. For the environment with unrelated parallel machines, this is not possible. In the following two theorems we establish a lower bound on the performance guarantee for swap- and multi-exchange-optimal schedules.

Theorem 9 *A performance guarantee for a swap- or multi-exchange-optimal solution for $R\|C_{\max}$ is not better than $p_{\max}/C^*_{\max}$.*

Proof: For given $K > m$ consider m machines and $n = m + 1$ jobs. The processing times of job J_1 are

$$p_{i1} = \begin{cases} \frac{1}{2} & \text{if} \quad i = 1, \\ K + \frac{1}{2} & \text{if} \quad i \neq 1, \end{cases}$$

for job J_2 are

$$p_{i2} = \begin{cases} K & \text{if } \; i = 1, \\ 1 & \text{if } \; i = 2, \\ K + \frac{1}{2} & \text{if } \; i \neq 1, 2, \end{cases}$$

and for job J_{m+1} are

$$p_{i,m+1} = \begin{cases} \frac{1}{2} & \text{if } \; i = 1, \\ K & \text{if } \; i = 2, \\ K + \frac{1}{2} & \text{if } \; i \neq 1, 2. \end{cases}$$

All other jobs have unit length on each machine, i.e., $p_{ij} = 1$ for $j = 3, \ldots, m$ and $i = 1, \ldots, m$. The optimal schedule has makespan $C^*_{\max} = 1$.

Consider the schedule that processes job J_1 on machine M_3, J_2 on M_1, and J_{m+1} on M_2 and all other jobs arbitrarily. This schedule is swap- and multi-exchange-optimal and has makespan $C^{\mathrm{S}}_{\max} = C^{\mathrm{M}}_{\max} = K + \frac{1}{2} = p_{\max}$. To see that this schedule is multi-exchange-optimal, note that only machine M_3 is critical. Hence, to improve the schedule we need to move job J_1 to machine M_1. If job J_2 is not moved, then we still have makespan $K + \frac{1}{2}$. Otherwise, job J_2 has to be moved to machine M_2. As the multi-exchange neighborhood ensures that no two jobs are moved to the same machine, job J_{m+1}, currently scheduled at M_2, cannot be moved to machine M_1 and thus the makespan remains $K + \frac{1}{2}$. □

The example in the above proof needs at least three machines. In the case of two machines a swap-, and thus multi-exchange-optimal schedule can be as bad as $n - 1$ times the optimal makespan.

Theorem 10 *The performance guarantee of swap- and multi-exchange-optimal schedules for $R2\|C_{\max}$ is at least $n - 1$.*

Proof: Consider n jobs, where J_1 has processing times

$$p_{i1} = \begin{cases} 1 & \text{if } \; i = 1, \\ n - 1 - \frac{1}{n-1} & \text{if } \; i = 2, \end{cases}$$

and the other jobs have processing times

$$p_{ij} = \begin{cases} 1 & \text{if } \; i = 1, j > 1, \\ \frac{1}{n-1} & \text{if } \; i = 2, j > 1. \end{cases}$$

In the optimal schedule, J_1 is scheduled on M_1 and the other jobs are processed by M_2. The makespan of this schedule is $C^*_{\max} = 1$. The schedule in which J_1 is processed by M_2 and the other jobs are scheduled on M_1 is swap- and multi-exchange-optimal, having makespan $C^{\mathrm{S}}_{\max} = n - 1$. □

4. Running Time

We have focused so far on the quality of local optima with respect to three neighborhoods. In this section, we remark on the time it takes a form of iterative improvement to find jump-optimal solutions.

Theorem 11 (Brucker et al. 1996, 1997) *A jump-optimal solution for $P\|C_{\max}$ can be found by iterative improvement, using $\mathcal{O}(n^2)$ jumps.*

The proof is in Brucker et al. (1996, 1997), who show that the iterative-improvement procedure that iteratively lets a job jump from the critical machine to a machine with minimum load finds a jump-optimal solution in $\mathcal{O}(n^2)$ iterations for $P\|C_{\max}$ as well as $P2\|C_{\max}$.

The result of Brucker et al. (1996) can be improved for $P2\|C_{\max}$ by always choosing the largest job on a critical machine that can be jumped. Using this rule for choosing the job to jump, the iterative-improvement procedure finds a jump-optimal solution in $\mathcal{O}(n)$ iterations. This is stated by the following theorem.

Theorem 12 *The iterative-improvement procedure that always chooses the largest possible job on a critical machine to jump, finds a jump-optimal solution for $P2\|C_{\max}$ in $\mathcal{O}(n)$ jumps.*

Proof: Brucker et al. (1996) show that the iterative-improvement procedure produces a sequence of feasible schedules with decreasing makespan and increasing minimum load. They also show that if the critical machine after a jump is the machine that had minimum load before the jump, then the job that jumped and all jobs that are at least as large as this job cannot be moved again in the following iterations. Therefore, only smaller jobs can be moved in the following iterations and these jobs have not been jumped before, as we always take the largest possible job to jump. Hence, each job can jump at most once and after $\mathcal{O}(n)$ iterations the procedure is finished and has found a jump-optimal solution. □

To describe the iterative-improvement procedure that finds jump-optimal solutions in the case of uniform parallel machines, we need to define the *slack* of a machine, as the total amount of processing requirement that can be added to this machine such that its load does not become larger than the makespan. We denote the slack of machine M_i by $\Delta_i = s_i(C'_{\max} - L_i)$, where $L_i = \sum_{J_j \in \mathcal{J}_i} p_{ij}$ and $C'_{\max}$ is the current makespan. Note that

when no job on a critical machine has a processing requirement that is less than the maximum slack, we have found a jump-optimal solution.

Our iterative-improvement procedure iteratively lets a job jump from a critical machine to a machine with maximum slack.

Theorem 13 *The iterative-improvement procedure described above finds a jump-optimal solution for $Q\|C_{\max}$ after $\mathcal{O}(n^2m)$ jumps.*

Proof: Note that this algorithm computes a sequence of schedules with non-increasing makespan and maximum slack. We denote the values of L_i, $C'_{\max}$, Δ_i, and $\Delta = \max_{1\leq i\leq m} \Delta_i$ in an iteration t by $L_i(t)$, etc.

Consider a machine M_i that was critical in iteration t_0 and had maximum slack in iteration $t_1 > t_0$, where t_0 and t_1 are chosen such that M_i is neither critical nor has maximum slack in iterations t, for $t_0 < t < t_1$. Note that if none of the machines satisfy this condition, the algorithm is finished after $\mathcal{O}(nm)$ iterations: if a job has been moved onto m different machines, it will certainly have been moved to a machine from which it was moved before.

Let job J_j be the job that was moved in iteration t_0. Then $L_i(t_1) = L_i(t_0) - p_j/s_i = C'_{\max}(t_0) - p_j/s_i$. By monotonicity of $C'_{\max}$, we have $\Delta(t_1) = s_i(C'_{\max}(t_1) - L_i(t_1)) \leq s_i(C'_{\max}(t_0) - L_i(t_0)) + p_j = p_j$. So, $p_j \geq \Delta(t_1)$ and by monotonicity of Δ, job J_j cannot be moved.

Hence, after at most nm iterations, at least one job cannot be moved, and thus after $\mathcal{O}(n^2m)$ iterations no job can move and the algorithm terminates. □

Corollary 5 *The iterative-improvement procedure described above finds a jump-optimal solution for $Q2\|C_{\max}$ in $\mathcal{O}(n^2)$ iterations.*

As with the identical-parallel-machine case, this result can be improved for $Q2\|C_{\max}$, when we always choose the largest possible job to jump.

Theorem 14 *The iterative-improvement procedure described above that always chooses the largest possible job to jump finds a jump-optimal solution for $Q2\|C_{\max}$ in $\mathcal{O}(n)$ jumps.*

Proof: The algorithm computes a sequence of schedules with decreasing makespan and maximum slack. Assume w.l.o.g. that, after iteration t, the critical machine changes from

M_1 to M_2, and that J_j jumped in that iteration. Using the same notation as in the proof of Theorem 13, we have

$$\Delta(t+1) = s_1(C'_{\max}(t+1) - L_1(t+1)) \leq s_1(C'_{\max}(t) - (C'_{\max}(t) - \frac{p_j}{s_1})) = p_j.$$

Therefore, only jobs smaller than J_j can jump in the following iterations, and as we always choose the largest possible job to jump, these jobs have not been moved before. Thus each job jumps at most once and the algorithm finds a jump-optimal solution after $\mathcal{O}(n)$ jumps. □

Hurkens and Vredeveld (2003) extended the above theorem to the general case of uniform parallel machines, showing a running time of $\mathcal{O}(nm)$ jumps to find a jump-optimal solution if the iterative-improvement procedure always chooses the largest possible job on a critical machine to jump.

5. Concluding Remarks

The main focus of this paper was the quality of local optima with respect to four neighborhoods. We have seen that, with respect to these neighborhoods, the local optima have a constant performance guarantee for the problem of minimizing makespan on identical parallel machines. In the case of uniform parallel machines, the two basic neighborhoods and the multi-exchange one do not have a constant performance guarantee. The neighborhood based on variable-depth search provides local optima that have a constant performance guarantee.

It would be interesting to see whether we can extend the push neighborhood to the unrelated-parallel-machine environment, or whether there exists a neighborhood for this machine environment that achieves a constant, or at least job-size-independent, performance guarantee.

We also saw that only a polynomial number of iterations is needed to find a jump-optimal solution for $P\|C_{\max}$ and $Q\|C_{\max}$. It is still an open question how many iterations iterative improvement needs to find a swap- or a push-optimal solution. We conjecture that a push-optimal solution cannot be found in polynomial time through iterative improvement. For the multi-exchange neighborhood, even finding an improving neighbor is already NP-hard, and therefore it cannot be hoped that there will be a polynomial-time iterative-improvement procedure that finds a local optimum with respect to this neighborhood.

Acknowledgments

The authors thank Jan Karel Lenstra and Cor Hurkens for their useful comments, and two anonymous referees for their helpful comments to improve the exposition. Moreover, we thank one anonymous referee for pointing out the paper by Frangioni et al. (2004).

The second author was supported by the project "High performance methods for mathematical optimization" of the Netherlands Organization for Scientific Research (NWO), by the EU project AMORE grant HPRN-CT-1999-00104, and by DFG research center MATHEON "Mathematics for Key Technologies."

A preliminary version of this paper appeared as Schuurman and Vredeveld (2001). The research was done while both authors were with the Department of Mathematics and Computer Science of Eindhoven University of Technology.

References

Ahuja, R.K., J.B. Orlin, D. Sharma. 2001. New neighborhood search structures for the capacitated minimum spanning tree problem. *Mathematical Programming* **91** 71–97.

Arkin, E.M., R. Hassin. 1998. On local search for weighted k-set packing. *Mathematics of Operations Research* **23** 640–648.

Arya, V., N. Garg, R. Khandekar, A. Meyerson, K. Munagala, V. Pandit. 2004. Local search heuristic for k-median and facility location problems. *SIAM Journal of Computing* **33** 544–562.

Ausiello, G., M. Protasi. 1995. Local search, reducibility and approximability of NP optimization problems. *Information Processing Letters* **54** 73–79.

Boykov, Y., O. Veksler, R. Zabih. 1999. A new algorithm for energy minimization with discontinuities. *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer, Berlin, Germany, 205–220.

Brucker, P., J. Hurink, F. Werner. 1996. Improving local search heuristics for some scheduling problems I. *Discrete Applied Mathematics* **65** 97–122.

Brucker, P., J. Hurink, F. Werner. 1997. Improving local search heuristics for some scheduling problems II. *Discrete Applied Mathematics* **72** 47–69.

Brüggemann, T., J. Monnot, G.J. Woeginger. 2003. Local search for the minimum label spanning tree problem with bounded color classes. *Operations Research Letters* **31** 195–201.

Charikar, M., S. Guha. 1999. Improved combinatorial algorithms for the facility location and k-median problems. *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*. 378–388.

Cho, Y., S. Sahni. 1980. Bounds for list schedules on uniform processors. *SIAM Journal on Computing* **9** 91–103.

Chudak, F.A., D.P. Williamson. 2005. Improved approximation algorithms for capacitated facility location problems. *Mathematical Programming* **102** 207–222.

De Bontridder, K.M.J., B.V. Halldórsson, M.M. Halldórsson, C.A.J. Hurkens, J.K. Lenstra, R. Ravi, L. Stougie. 2003. Approximation algorithms for the test cover problem. *Mathematical Programming* **98** 477–491.

Feige, U., M. Karpinski, M. Langberg. 2002. Improved approximation of Max-Cut on graphs of bounded degree. *Journal of Algorithms* **43** 201–219.

Finn, G., E. Horowitz. 1979. A linear time approximation algorithm for multiprocessor scheduling. *BIT* **19** 312–320.

Frangioni, A., E. Necciari, M.G. Scutellà. 2004. A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. *Journal of Combinatorial Optimization* **8** 195–220.

Garey, M.R., D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.

Goemans, M.X., D.P. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* **42** 1115–1145.

Graham, R. L., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* **5** 287–326.

Gupta, A., É. Tardos. 2000. A constant factor approximation algorithm for a class of classification problems. *Proceedings of 32nd ACM Symposium on Theory of Computing*. 652–658.

Hochbaum, D.S., D.B. Shmoys. 1987. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM* **34** 144–162.

Hochbaum, D.S., D.B. Shmoys. 1988. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing* **17** 539–551.

Hurkens, C.A.J., A. Schrijver. 1989. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics* **2** 68–72.

Hurkens, C.A.J., T. Vredeveld. 2003. Local search for multiprocessor scheduling: how many moves does it take to a local optimum? *Operations Research Letters* **31** 137–141.

Kernighan, B.W., S. Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* **49** 291–307.

Korupolu, M.R., C.G. Plaxton, R. Rajaraman. 2000. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms* **37** 146–188.

Lenstra, J.K., D.B. Shmoys, É. Tardos. 1990. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming* **46** 259–271.

Lin, S., B.W. Kernighan. 1973. An effective heuristic for the traveling salesman problem. *Operations Research* **21** 498–516.

Lu, H., R. Ravi. 1992. The power of local optimization: approximation for maximum-leaf spanning tree. *Proceedings of the 13th Annual Allerton Conference on Communication, Control, and Computing*. 533–542.

Pál, M., Éva Tardos, T. Wexler. 2001. Facility location with nonuniform hard capacities. *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science*. 329–338.

Schuurman, P., T. Vredeveld. 2001. Performance guarantees of local search for multiprocessor scheduling. *Proceedings of 8th Integer Programming and Combinatorial Optimization Conference*. LNCS 2081, Springer, Berlin, Germany, 370–382.

Vredeveld, T., J.K. Lenstra. 2003. On local search for the generalized graph coloring problem. *Operations Research Letters* **31** 28–34.