Local search performance guarantees for restricted related parallel machine scheduling

Maastricht University

Diego Recalde, Cyriel Rutten,
Petra Schuurman, Tjark Vredeveld

Local search performance guarantees for restricted related parallel machine scheduling

RM/09/061

METEOR

Maastricht University School of Business and Economics
Maastricht Research School of Economics
of Technology and Organization

P.O. Box 616
NL - 6200 MD Maastricht
The Netherlands

Local search performance guarantees for restricted related parallel machine scheduling

Diego Recalde[1⋆], Cyriel Rutten[2], Petra Schuurman[3] and Tjark Vredeveld[4]

[1] Escuela Politécnica Nacional, Department of Mathematics, Ladrón de Guevara E11-253, Quito, Ecuador. E-mail: drecalde@math.epn.edu.ec
[2] Maastricht University, Department of Quantitative Economics, P.O. Box 616, 6200 MD Maastricht, The Netherlands. E-mail: c.rutten@maastrichtuniversity.nl
[3] Sterk spel, Eindhoven, The Netherlands. E-mail: petraschaakt@xs4all.nl
[4] Maastricht University, Department of Quantitative Economics, P.O. Box 616, 6200 MD Maastricht, The Netherlands. E-mail: t.vredeveld@maastrichtuniversity.nl

Abstract. We consider the problem of minimizing the makespan on restricted related parallel machines. In restricted machine scheduling each job is only allowed to be scheduled on a subset of machines. We study the worst-case behavior of local search algorithms. In particular, we analyze the quality of local optima with respect to the jump, swap, push and lexicographical jump neighborhood.

Key words: Local search, performance guarantee, restricted machines, eligibility constraints.

1 Introduction

We consider the problem of minimizing the makespan in restricted related parallel machine scheduling. In this setting, each job is only allowed to be scheduled on a subset of machines. The problem is also known as the related parallel machine scheduling problem with eligibility constraints [19, 20, 27] or as the restricted assignment model for related parallel links [1, 9, 10]. It has applications in, among others, operating systems, communication networks [17], semiconductor manufacturing [4], and the throughput management of hospital operating rooms [28].

The problem is defined as follows. Given is a set $J = \{1, ..., n\}$ of n jobs and a set $M = \{1, ..., m\}$ of m machines. Each job j needs to be scheduled on one of its *eligible machines* $\mathcal{M}_j \subseteq M$. We will refer to the family $\{\mathcal{M}_j\}$ as *eligibility sets*. We also say that job j is *allowable* on machine i if $i \in \mathcal{M}_j$. A machine $i \in M$ can process at most one job at a time, and all jobs and machines are available at time 0. Each machine $i \in M$ is characterized by a processing speed $s_i > 0$. Similarly, each job has a processing requirement $p_j > 0$. If a job j is

⋆ Partially supported by a grant of the Ecuadorian Organization for Science and Technology (SENACYT)

allowable on a machine i, then the processing time of job j on machine i, p_{ij}, equals p_j/s_i. If job j is not allowable on machine i, then p_{ij} is set to infinity. We refer to the setting as stated above as *restricted related parallel machines*. The term "restricted" refers to jobs being restricted in the sense that they are only allowed to be processed on a subset of machines. The objective is to schedule the jobs in such a way that the *makespan* is minimized, i. e., we seek the last job to be completed as early as possible. The *latency* of a machine is the ratio of the processing requirements of all jobs assigned to the machine over its speed. Then, the *makespan* equals the maximum latency over machines. In absence of the eligibility constraints, that is $\mathcal{M}_j = M$ for all jobs j, the model is known as the *(uniform) related parallel machine scheduling model*. In the special case of *restricted identical parallel machines* the processing speed of each machine equals one. Furthermore, we refer to the special case wherein the processing requirements of all jobs equal one as the *restricted related parallel machines with identical (unit-length) jobs*. Adapting the standard notation introduced by [14], the problems of minimizing the makespan on restricted identical parallel machines, restricted related parallel machines with identical jobs and restricted related parallel machines are denoted by $P|\mathcal{M}_j|C_{\max}$, $Q|\mathcal{M}_j, p_j = 1|C_{max}$ and $Q|\mathcal{M}_j|C_{\max}$, respectively, see e. g., [18, 25]. Standard scheduling problems which minimize the makespan on identical or related parallel machines are both known to be strongly NP-hard [12]. Hence, the problems with eligibility constraints are strongly NP-hard as well. One way to deal with NP-hard problems is to find approximative solutions. If an algorithm is guaranteed to deliver a solution that has a value at most ρ times the optimal solution value, we call it a *ρ-approximation* algorithm. ρ is called the *performance guarantee*.

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. A *neighborhood function* defines for each feasible solution a set of solutions which are in some sense close to it. This set is called a *neighborhood*. The choice of a suitable neighborhood function has an important influence on the performance of local search. The simplest form of local search is *iterative improvement*, also called local improvement algorithms. This method iteratively chooses a better solution in the neighborhood of the current solution and it terminates when no better solution is found; we say that the final solution is a *local optimum*.

Neighborhoods In this paper, we investigate the performance guarantees of four different neighborhoods for various restricted related parallel machines settings, namely the *jump, swap, push* and *lexicographical jump (lexjump) neighborhood*.

Before discussing the neighborhoods, we first describe our representation of a schedule. Since the order in which the jobs are processed on a machine does not influence the latency of the corresponding machine, we will represent a schedule by an assignment. An *assignment* A is uniquely determined by a partition of the set of jobs J into m disjoint subsets $J_1^A, J_2^A, \ldots, J_m^A$ where J_i^A denotes the set of jobs assigned to machine $i \in M$ in assignment A. Let $A(j) \in \mathcal{M}_j$ denote the

machine to which job j is assigned in assignment A, that is, $A(j) = i$ implies $j \in J_i^A$ and vice versa. The *load of a machine* is the total processing requirement assigned to the machine for some assignment A, i. e., $L_i^A = \sum_{j \in J_i^A} p_j$, for all $i \in M$. The *latency of a machine* is the total processing time needed by a machine to process all jobs which are assigned to it, i. e., $\Lambda_i^A = \sum_{j \in J_i^A} p_{ij} = L_i^A / s_i$. Obviously, for identical parallel machines, $\Lambda_i^A = L_i^A$ for all machines $i \in M$. A *critical machine* is a machine with maximum latency. The makespan of some given assignment A, $C_{\max}^A$, i. e., the latest completion time of a job, equals the latency of the critical machine(s). Thus, $C_{\max}^A = \max_{i \in M} \Lambda_i^A$.

The first neighborhood we consider is the *jump neighborhood*, also known as the *move neighborhood*. A *jump* is defined as jumping or moving a job from the machine to which it is currently assigned to another machine (on which it is allowed). In the jump neighborhood, jobs are iteratively jumped from a critical machine to a non-critical machine. We say that an assignment is a *jump optimal assignment* if no jump decreases the makespan or the number of critical machines without increasing the makespan.

The second neighborhood we consider is the *swap neighborhood*. Select two jobs, j and k, assigned to different machines, i. e., $j \in J_i^A$, $k \in J_h^A$ and $i \neq h$, such that $h \in \mathcal{M}_j$ and $i \in \mathcal{M}_k$. A *swap* is performed by interchanging the machine allocations of the jobs. If all jobs are assigned on the same machine, then no swap neighbor exists. Therefore, we define the swap neighborhood as one that consists of all possible jumps which jump a job from a critical machine to a non-critical machine and all possible swaps which swap a job from a critical machine with another job from a non-critical machine. We say that an assignment is a *swap optimal assignment* if no jump or swap decreases the makespan or the number of critical machines without increasing the makespan.

Next, we consider the *push neighborhood* introduced by Schuurman and Vredeveld [26]. A *push* consists of a sequence of jumps. Starting with an assignment A with makespan $C_{\max}^A$, a push is initiated by selecting a job k on a critical machine and a machine $i \in \mathcal{M}_k$ to move it. We say that k fits on i if $p_{ik} + \sum_{j \in J_i^A : p_{ij} \geq p_{ik}} p_{ij} < C_{\max}^A$. If a job k fits on some machine i, then we move j to i and iteratively remove the smallest job from i until the latency of i is less than $C_{\max}^A$. The removed jobs are gathered in a queue. We now have a queue of pending jobs and a partial assignment 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 one of its eligible 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 eligible 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 until we have determined that the push is unsuccessful. If none of the jobs on any of the critical machines can succesfully be pushed, then we are in a *push optimal assignment*. The push neighborhood is explained in more detail in [26].

The last neighborhood we consider is the *lexicographical jump* (*lexjump*) *neighborhood*. Define the *experienced latency of a job* as the latency of the ma-

chine to which the job is currently assigned. An assignment A is lexjump optimal if no jump exists which decreases the latency of a machine i without increasing the latency of another machine $h \neq i$ to a value exceeding the original latency of machine i. In other words, no job can decrease its experienced latency by jumping to another machine. That is, A is lexjump optimal if $p_{hj} + \Lambda_h^A \geq \Lambda_i^A$ for all $i \in M$, $j \in J_i^A$, $h \in \mathcal{M}_j$. Notice that the notion of a lexjump optimal assignment corresponds to the notion of pure Nash equilibrium in the context of load balancing games, see e. g., [29].

Related work Worst case analysis of local search has become increasingly popular in the last decade. A summary of the best known upperbounds on the performance guarantees for the jump, swap, push and the lexjump neighborhood for (unrestricted) identical parallel machines and (unrestricted) related parallel machines is provided in Table 1. Schuurman and Vredeveld [26] provided examples showing that the bounds of the jump and the swap neighborhood are tight for identical and related parallel machines. In addition to the bounds provided in Table 1, Brueggemann, Hurink, Vredeveld, and Woeginger [3] introduced the so-called split neighborhood and considered the corresponding performance guarantees. They showed that this exponentially sized neighborhood has a performance guarantee of at most $2-2/(m+1)$. Moreover, combining this neighborhood with the jump neighborhood improves the guarantee only to $2-4/(m+3)$. However, if the split neighborhood is combined with the lexjump neighborhood, the performance guarantee drops to $3/2$.

A lexjump optimal assignment corresponds to a pure Nash equilibrium for the appropriate defined game. Therefore, the results on the price of anarchy carry over to performance guarantees for lexjump optimal assignments. Czumaj and Vöcking [6] showed the performance guarantee of a lexjump optimal assignment on related parallel machines is in $O(\min\{\log m/\log\log m), \log(s^+/s^-)\}$, where $s^+ := \max_i s_i$ and $s^- := \min_i s_i$, and that this bound is aymptotically tight. For unrelated parallel machines, Awerbuch, Azar, Richter and Tsur [1] showed

Table 1. Local search performance guarantees for unrestricted parallel machines which are shown to be tight. "LB" and "UB" denote a lowerbound and an upperbound on the performance guarantee respectively if the performance guarantee is not shown to be tight.

Setting	Jump	Swap	Push	Lexjump
2 identical machines	$\frac{4}{3}$ [8]	$\frac{4}{3}$ [8]	$\frac{8}{7}$ [26]	$\frac{4}{3}$ [29]
Identical machines ($m > 2$)	$2 - \frac{2}{m+1}$ [8]	$2 - \frac{2}{m+1}$ [8]	UB $= \frac{4}{3} - \frac{1}{3m}$ [26] LB $= \frac{4m}{3m+1}$ [26]	$2 - \frac{2}{m+1}$ [29]
Related machines	$\frac{1+\sqrt{4m-3}}{2}$ [5]	$\frac{1+\sqrt{4m-3}}{2}$ [5]	UB $= 2 - \frac{2}{m+1}$ [26] LB $= \frac{3}{2} - \epsilon$ [26]	$O\left(\frac{\log m}{\log\log m}\right)$ [29]

that the performance guarantee is in $\Theta(p^+ + (\log m/\log(2+(\log m)/p^+)))$ where $p^+ := \max_{j,i,h:p_{ij}<\infty} p_{ij}/p_{hj}$.

Recently, results for lexjump optimal assignments on restricted parallel machines have been developed. Awerbuch et al. [1] proved that the performance guarantee for identical machines is bounded by $\Theta(\log m/(r\cdot\log(2+(\log m)/r)))$, where r denotes the ratio between the makespan of the optimal schedule and the largest task (note $r \geq 1$). Note that the general bound for identical machines of $\Theta(\log m/\log\log m)$ are obtained by setting $r = 1$, i. e., when making no assumptions on the largest job in the system. Hoefer and Souza [15] provided an alternative upper bound for the performance guarantee: $1 + m^2/\sum_{j\in J} p_j$. Gairing, Lücking, Mavronicolas and Monien [9] showed that the performance guarantee for restricted related parallel machines and identical jobs is in $\Omega(\log n/\log\log n)$. For restricted related parallel machines, they show that the performance guarantee is bounded from below by $m-1$ and bounded from above by m. Since the counterexample of Gairing et al. [9], which shows that the performance guarantee for restricted related machines can be as bad as $m-1$, is somewhat artificial, Lu and Yu [22] introduced the concept of *λ-goodness instances* to develop an alternative performance guarantee. An instance is λ-good if and only if every job can use at least one machine which has a speed of no less than s^+/λ. Lu and Yu show that for λ-good instances, the performance guarantee is in $\Theta\left(\min\left\{\frac{\log \lambda m}{\log\log \lambda m}, m\right\}\right)$.

For a more elaborate overview of worst case analysis and other theoretical aspects of local search, we refer to the book of Michiels, Aarts, and Korst [23].

In [25, 21, 19] polynomial time algorithms to solve several special cases for scheduling unit-length jobs on restricted related parallel machines to optimality are given. Glass and Kellerer [13] gave several polynomial time approximation algorithms for special cases of the problem of restricted parallel machines with performance guarantees of $2 - 1/m$ or better. A PTAS for the identical parallel machines cases with a special type of eligibility sets is given by Ou, Leung, and Li [24]. We refer to Leung and Li [18] for a survey on results on polynomial time algorithms, complexity issues and approximation schemes concerning scheduling problems with restricted machines.

Table 2. Local search performance guarantees for restricted parallel machines. Let $s^- := \min_i s_i$, $s^+ := \max_i s_i$, $\widetilde{s_i} := s_i/s^-$ and $\widetilde{s} := s^+/s^-$.

Setting	Jump/Swap/Push	Lexjump
Identical Machines	$1/2 + \sqrt{m - 3/4}$	$O\left(\frac{\log m}{\log\log m}\right)$ [1]
Identical Jobs	$\sqrt{(1 + \frac{m-1}{n})\sum_{i\in M} \widetilde{s_i}}$	$O\left(\frac{\log n}{\log\log n}\right)$ [9]
Related machines	$1/2 + \sqrt{1/4 + (m-1)\tilde{s}}$	$O\left(\frac{\log \sum_{i\in M} \widetilde{s_i}}{\log\log \sum_{i\in M} \widetilde{s_i}}\right)$

Our contribution In this paper we consider the following neighborhoods: the jump, swap, push and lexicographical jump neighborhood. We analyze the quality of each neighborhood by establishing worst-case performance guarantees for the restricted identical parallel machines, restricted related parallel machines with identical jobs and restricted related parallel machines problems. The new performance guarantees are summarized in Table 2, see the unreferenced bounds. Furthermore, we provide examples to show that these performance guarantees are tight or almost tight.

2 Performance guarantees for restricted identical parallel machines

In this section, we provide performance guarantees for the scheduling problem of minimizing makespan on restricted identical parallel machines. For the jump neighborhood we obtain the following result.

Theorem 1. *A jump-optimal assignment for restricted identical parallel machines has makespan at most* $1/2 + \sqrt{m - 3/4}$ *times the optimal makespan.*

Theorem 1 follows straightforward from Theorem 5 since for the case of identical machines $s_i = 1$ for all machines $i \in M$. The following example shows that there exist instances for which the performance guarantee is tight.

Example 1. Let k be an arbitrary positive integer and consider an instance with $n = k(k-1)+1$ jobs and $m = n$ machines. All jobs have processing time $p_j = 1$. Jobs $1, \ldots, k$ can only be processed on the first k machines. The remaining jobs are allowable on all machines. Consider the following assignment which is depicted in Figure 1. Jobs $1, \ldots, k$ are assigned to machine 1. Machines $2, \ldots, k$ process each $k-1$ of the remaining jobs. This assignment is jump optimal and has a makespan of $C^A_{\max} = k$, whereas in an optimal assignment, each machine processes only one job and $C^{OPT}_{\max} = 1$. Hence, $C^A_{\max}/C^{OPT}_{\max} = k = 1/2 + \sqrt{m - 3/4}$.

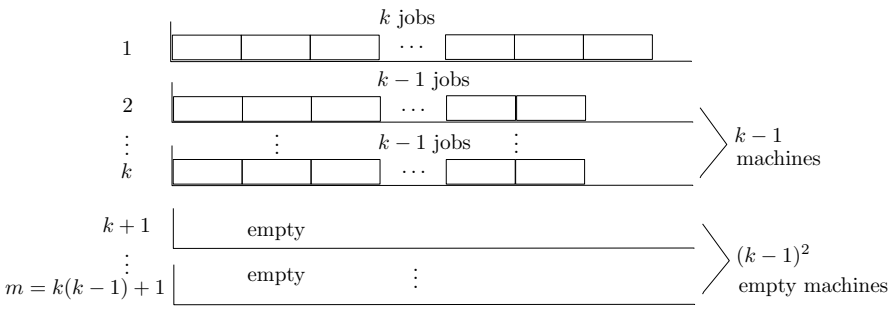

Fig. 1. Jump Optimal Assignment

The following theorem has been established independently by Awerbuch, Azar, Richter and Tsur [1] and Gairing, Lücking, Mavronicolas and Monien [9].

Theorem 2 (Awerbuch et al. [1], Gairing et al. [9]). *The performance guarantee of lexjump optimal assignments for the problem of minimizing the makespan on restricted identical parallel machines is* $O(\log m / \log\log m)$.

Gairing et al. [9] also provide an example showing that the bound of Theorem 2 is tight up to a constant factor.

3 Performance guarantees for restricted related parallel machines with identical (unit-length) jobs

In this section, we discuss performance guarantees on restricted related parallel machines for the special case of identical (unit-length) jobs. The general case for arbitrary jobs will be discussed in the next section. For now we will assume that $p_j = 1$ for all jobs $j \in J$. Denote by s^- the minimum speed among all machines, i. e., $s^- := \min_{i \in M} s_i$. For the jump neighborhood we have the following result:

Theorem 3. *A jump optimal assignment for restricted related parallel machines with identical (unit-length) jobs has a makespan of at most*

$$\sqrt{\left(1 + \frac{m-1}{n}\right) \sum_{i \in M} \widetilde{s_i}}$$

where $\widetilde{s_i}$ *denotes the relative speed, i. e.,* $\widetilde{s_i} := s_i / s^-$.

Proof: Assume without loss of generality that machine 1 is the critical machine. Let A be a jump optimal assignment. Then

$$C^A_{\max} = \Lambda^A_1 \leq \Lambda^A_i + \frac{1}{s_i} \qquad \forall\, i \in \mathcal{M}^A_1 = \bigcup_{j \in J^A_1} \mathcal{M}_j. \tag{1}$$

Multiplying the equality above by the corresponding speeds, summing over all machines $i \in \mathcal{M}^A_1 \setminus \{1\}$ and adding the load of machine 1 to both sides of the inequality yields

$$s_1 \Lambda^A_1 + \sum_{i \in \mathcal{M}^A_1 \setminus \{1\}} (s_i C^A_{\max}) = \sum_{i \in \mathcal{M}^A_1} (s_i C^A_{\max}) \tag{2}$$

$$\leq s_1 \Lambda^A_1 + \sum_{i \in \mathcal{M}^A_1 \setminus \{1\}} (s_i \Lambda^A_i + 1) = \sum_{i \in \mathcal{M}^A_1} (s_i \Lambda^A_i) + (x - 1) \tag{3}$$

where $x := \left|\mathcal{M}^A_1\right|$. Thus, in A, the machines in $\mathcal{M}^A_1$ process together a load of at least $1 - x + \sum_{i \in \mathcal{M}^A_1} (s_i C^A_{\max})$. In an optimal assignment OPT each machine

$i \in M$ processes a load of at most $s_i C^{OPT}_{\max}$. Hence, in any optimal assignment, a load of at least

$$1 - x + \sum_{i \in \mathcal{M}_1^A} \left(s_i (C^A_{\max} - C^{OPT}_{\max}) \right)$$

must be assigned to machines in $M \backslash \mathcal{M}_1^A$. Thus,

$$\sum_{i \in M \backslash \mathcal{M}_1^A} s_i C^{OPT}_{\max} \geq 1 - x + \sum_{i \in \mathcal{M}_1^A} \left(s_i (C^A_{\max} - C^{OPT}_{\max}) \right).$$

yielding

$$C^A_{\max} \cdot \sum_{i \in \mathcal{M}_1^A} s_i \leq (x-1) + C^{OPT}_{\max} \cdot \sum_{i \in M} s_i \leq (m-1) \frac{C^{OPT}_{\max}}{n} \sum_{i \in M} s_i + C^{OPT}_{\max} \cdot \sum_{i \in M} s_i$$

$$= C^{OPT}_{\max} \left(1 + \frac{m-1}{n} \right) \sum_{i \in M} s_i$$

as $x \leq m$, and $C^{OPT}_{\max} \cdot \sum_{i \in M} s_i \geq \sum_{j \in J} p_j = n$, that is $1 \leq (C^{OPT}_{\max}/n) \sum_{i \in M} s_i$. We obtain

$$\frac{C^A_{\max}}{C^{OPT}_{\max}} \leq \frac{\left(1 + \frac{m-1}{n}\right) \sum_{i \in M} s_i}{\sum_{i \in \mathcal{M}_1^A} s_i} \tag{4}$$

Furthermore, we have that $s_1 C^A_{\max} \leq \sum_{i \in \mathcal{M}_1^A} (s_i C^{OPT}_{\max})$ from which it follows that

$$\frac{C^A_{\max}}{C^{OPT}_{\max}} \leq \sum_{i \in \mathcal{M}_1^A} \frac{s_i}{s_1} \leq \sum_{i \in \mathcal{M}_1^A} \frac{s_i}{s^-}. \tag{5}$$

Combining (4) and (5) provides us with the result

$$\frac{C^A_{\max}}{C^{OPT}_{\max}} = \sqrt{\left(\frac{C^A_{\max}}{C^{OPT}_{\max}} \right)^2} \overset{(4)\&(5)}{\leq} \sqrt{\left(\frac{\left(1 + \frac{m-1}{n}\right) \sum_{i \in M} s_i}{\sum_{i \in \mathcal{M}_1^A} s_i} \right) \left(\frac{\sum_{i \in \mathcal{M}_1^A} s_i}{s^-} \right)} \tag{6}$$

$$= \sqrt{\left(1 + \frac{m-1}{n}\right) \sum_{i \in M} \widetilde{s}_i} \tag{7}$$

□

The example below shows that there exist instances of three machines and particular speeds for which the performance guarantee of Theorem 3 is asymptotically tight. We remark that the example below can be generalized to any number of machines.

Example 2. Let k be an arbitrary strictly positive integer and consider the following instance and a jump-optimal assignment A. Each job has processing requirement $p_j = 1$ as is required in this section. We have three machines for which $s_1 = s^- = 1$, $s_2 = k - 1$ and $s_3 = k(k-1) - 1$. k jobs are assigned to machine 1

but are allowed on machines 1 and 2. $k(k-1)-1$ jobs are assigned to machine 2 but are allowed on machines 2 and 3. No jobs are assigned to machine 3. We have $\Lambda_1^A = C_{\max}^A = k$, $\Lambda_2^A = k - 1/(k-1)$ and $\Lambda_3^A = 0$. An optimal assignment is obtained by assigning $k-1$ jobs, which are in A assigned to machine 1, to machine 2 and by assigning all jobs which are in A assigned to machine 2, to machine 3. Consequently, $C_{\max}^A / C_{\max}^{OPT} = k/1 = k$. Theorem 3 yields the following upper bound on the performance guarantee

$$\sqrt{\left(1 + \frac{m-1}{n}\right) \sum_{i \in M} \widetilde{s_i}} = \sqrt{\left(1 + \frac{3-1}{k^2-1}\right)(k^2-1)} = \sqrt{k^2+1} \overset{k \to \infty}{\longrightarrow} k = \frac{C_{\max}^A}{C_{\max}^{OPT}}. \quad (8)$$

The results from Gairing, Lücking, Mavronicolas and Monien [9] yield the following result.

Theorem 4 (Gairing et al. [9], Theorem 3.1 and Theorem 4.2). *The performance guarantee of a lexjump optimal assignment for the problem of minimizing the makespan on restricted related parallel machines and identical jobs is* $\Theta(\log n / \log \log n)$.

4 Performance guarantees for restricted related parallel machines

In this section, we establish performance guarantees for the scheduling problem of minimizing the makespan on restricted related parallel machines. Recall that in this machine environment $p_{ij} = p_j / s_i$ for $i \in \mathcal{M}_j$ and $p_{ij} = \infty$ otherwise. Let $s^+ := \max_{i \in M} s_i$ and let $s^- := \min_{i \in M} s_i$. For the jump neighborhood we obtain the following result.

Theorem 5. *A jump optimal assignment for restricted related parallel machines has makespan at most* $1/2 + \sqrt{1/4 + (m-1)\tilde{s}}$ *times the optimal solution value; where* $\tilde{s} := s^+/s^- = \max_{i,h \in M} s_i/s_h$.

Proof: Consider a jump optimal assignment A having makespan $C_{\max}^A$. Assume w.l.o.g. that machine 1 is a critical machine, i. e., $\Lambda_1^A = C_{\max}^A$. Let $\mathcal{M}_1^A$ be the set of machines to which a job, currently assigned to machine 1 for assignment A, can be moved, i. e., $\mathcal{M}_1^A = \bigcup_{j \in J_1^A} \mathcal{M}_j$. Let $x := |\mathcal{M}_1^A|$ and $p^+ := \max_{j \in J} p_j$. Consider a machine $i \in \mathcal{M}_1^A$ such that $i \neq 1$. Then, there exists at least one job $j \in J_1^A$ such that $i \in \mathcal{M}_j$. By jump optimality of A we have, $\Lambda_i^A + p_j/s_i \geq C_{\max}^A$. Consequently, $\Lambda_i^A \geq C_{\max}^A - p^+/s_i$ for all $i \in \mathcal{M}_1^A \setminus \{1\}$. Multiplying the last inequality by s_i and accumulating over all machines $i \in \mathcal{M}_1^A$ we obtain,

$$\sum_{i \in \mathcal{M}_1^A} L_i^A = \sum_{i \in \mathcal{M}_1^A} s_i \Lambda_i^A \geq s_1 C_{\max}^A + \sum_{i \in \mathcal{M}_1^A \setminus \{1\}} s_i \left(C_{\max}^A - \frac{p^+}{s_i} \right). \quad (9)$$

To convert assignment A to an optimal assignment with makespan $C^{OPT}_{\max}$, we need to move at least a load of

$$s_1(C^A_{\max} - C^{OPT}_{\max}) + \sum_{i \in \mathcal{M}^A_1 : i \neq 1} s_i \left(C^A_{\max} - C^{OPT}_{\max} - \frac{p^+}{s_i} \right) \tag{10}$$

from the machines in $\mathcal{M}^A_1$ to the machines in $M \backslash \mathcal{M}^A_1$. Therefore,

$$\begin{aligned}
(m-x)s^+ C^{OPT}_{\max} &\geq \sum_{i \in M \backslash \mathcal{M}^A_1} s_i \Lambda^{OPT}_i && (11)\\
&\overset{(10)}{\geq} s_1(C^A_{\max} - C^{OPT}_{\max}) && (12)\\
&\quad + \sum_{i \in \mathcal{M}^A_1 : i \neq 1} s_i \left(C^A_{\max} - C^{OPT}_{\max} \right) - \sum_{i \in \mathcal{M}^A_1 : i \neq 1} p^+ && (13)\\
&= \sum_{i \in \mathcal{M}^A_1} s_i \left(C^A_{\max} - C^{OPT}_{\max} \right) - (x-1)p^+ && (14)\\
&\geq \sum_{i \in \mathcal{M}^A_1} s_i \left(C^A_{\max} - C^{OPT}_{\max} \right) - (x-1)s^+ C^{OPT}_{\max}, && (15)
\end{aligned}$$

since $p^+/s^+ \leq C^{OPT}_{\max}$. Then,

$$\frac{C^A_{\max}}{C^{OPT}_{\max}} \leq \frac{(m-x)s^+ + \sum_{i \in \mathcal{M}^A_1} s_i + (x-1)s^+}{\sum_{i \in \mathcal{M}^A_1} s_i} = \frac{(m-1)s^+}{\sum_{i \in \mathcal{M}^A_1} s_i} + 1. \tag{16}$$

As in an optimal assignment the jobs in J^A_1 must be assigned to the machines in $\mathcal{M}^A_1$, we have $s_1 C^A_{\max} \leq \sum_{i \in \mathcal{M}^A_1} s_i C^{OPT}_{\max}$ and consequently

$$\frac{C^A_{\max}}{C^{OPT}_{\max}} \leq \sum_{i \in \mathcal{M}^A_1} \frac{s_i}{s_1} \leq \sum_{i \in \mathcal{M}^A_1} \frac{s_i}{s^-}. \tag{17}$$

Combining (16) and (17) yields

$$\frac{C^A_{\max}}{C^{OPT}_{\max}} \left(\frac{C^A_{\max}}{C^{OPT}_{\max}} - 1 \right) \overset{(17)}{\leq} \sum_{i \in \mathcal{M}^A_1} \frac{s_i}{s^-} \left(\frac{C^A_{\max}}{C^{OPT}_{\max}} - 1 \right) \overset{(16)}{\leq} (m-1)\frac{s^+}{s^-}. \tag{18}$$

From this it follows that,

$$\frac{C^A_{\max}}{C^{OPT}_{\max}} \leq \frac{1}{2} + \sqrt{\frac{1}{4} + (m-1)\frac{s^+}{s^-}}. \tag{19}$$

□

Note that the bound given in Theorem 5 corresponds to the bound given in Theorem 1 by setting $s_i = 1$ for all machines $i \in M$. Therefore, Example 1

shows that the bound of Theorem 5 is tight for $\tilde{s} = 1$. The example below shows that there exist instances with non-identical speeds for which a jump optimal assignment has a makespan of at least $\sqrt{\tilde{s}(m-1)+1/4} \cdot C_{\max}^{OPT}$, leaving a gap of less than 1/2 between the upper and the lower bound on the performance guarantee.

Example 3. Let $k > 1$ be an arbitrary strictly positive integer and consider the following instance. Let there be $m = k + 3$ machines having speeds $s_1 = 1$ and $s_2 = \ldots = s_m = k$. Let there be $k + 1$ jobs of size $p_j = 1$ for which $\mathcal{M}_j = \{1, 2\}$ and $k + 1$ jobs of size $p_j = k$ for which $\mathcal{M}_j = M$. Additionally, there is one job of size $\epsilon > 0$ which is only allowed on machine 1. In an optimal assignment, one job of size 1 and the one job of size ϵ are assigned to machine 1, k jobs of size 1 are assigned to machine 2 and 1 job of size k is assigned to each of the remaining machines. Then, $C_{\max}^{OPT} = 1+\epsilon$. Consider the following jump optimal assignment A: $k + 1$ jobs of size 1 and the one job of size ϵ are assigned to machine 1, $k + 1$ jobs of size k are assigned to machine 2 and all the other machines remain empty. Then $C_{\max}^{A} = k + 1 + \epsilon$. Hence, when ϵ tends to zero, $C_{\max}^{A}/C_{\max}^{OPT}$ tends to $k + 1$. Since $k + 1 > \sqrt{k(k+2)+1/4} = \sqrt{\tilde{s}(m-1)+1/4}$, we have established a lower bound of $\sqrt{\tilde{s}(m-1)+1/4}$ on the performance guarantee of the jump neighborhood for related parallel machines by taking ϵ small enough.

In order to prove Theorem 6, we make use of the Gamma function. The Gamma function is denoted by $\Gamma(x)$ and is defined by $\Gamma(x) := \int_0^\infty t^{x-1}e^{-x}\partial t$ for $x \in \mathbb{R}^+$. Let $\Gamma^{-1}(x)$ denote the inverse Gamma function. For any natural number N the gamma function is defined as $\Gamma(N + 1) := N!$. It is known that both the Gamma function as well as the inverse Gamma function are monotonically increasing and that $\Gamma^{-1}(x) = O(\log x/\log\log x)$. Let $C_{\max}^{OPT}$ denote the optimal makespan. We introduce the following notation. For an assignment A and for any positive integer k, we define the set $R_k^A = \left\{ i \in M : \Lambda_i^A \geq k \cdot C_{\max}^{OPT} \right\}$, i. e., R_k^A denotes the set of machines having a latency of at least $k \cdot C_{\max}^{OPT}$. Let $\overline{R_k^A} = R_k^A \backslash R_{k+1}^A$. An illustration of these definitions is provided in Figure 2. Before proving Theorem 6, we first establish the lemma below.

Lemma 1. *Let A be a lexjump optimal assignment. Then, a job $j \in J_i^A$ assigned to a machine $i \in R_{k+1}^A$ will not be assigned to a machine $h \in M \setminus R_k^A$ in any optimal assignment.*

Proof: Consider a job j such that $A(j) \in R_{k+1}^A$. If $\mathcal{M}_j \subseteq R_k^A$, then the statement of the lemma is trivially satisfied. Otherwise, let i be a machine in $M \backslash R_k^A$ and let $i \in \mathcal{M}_j$. Since A is a lexjump optimal assignment,

$$\Lambda_i^A + \frac{p_j}{s_i} \geq \Lambda_{A(j)}^A \geq (k+1) C_{\max}^{OPT}. \tag{20}$$

As $i \in M \setminus R_k^A$, $\Lambda_i^A < k\, C_{\max}^{OPT}$ and thus $p_j/s_i > C_{\max}^{OPT}$. Hence, in an optimal assignment, job j will not be assigned to any machine $i \in \mathcal{M}_j \cap \left(M \backslash R_k^A\right)$. □

For the lexjump neighborhood for restricted related parallel machines we have the following result.

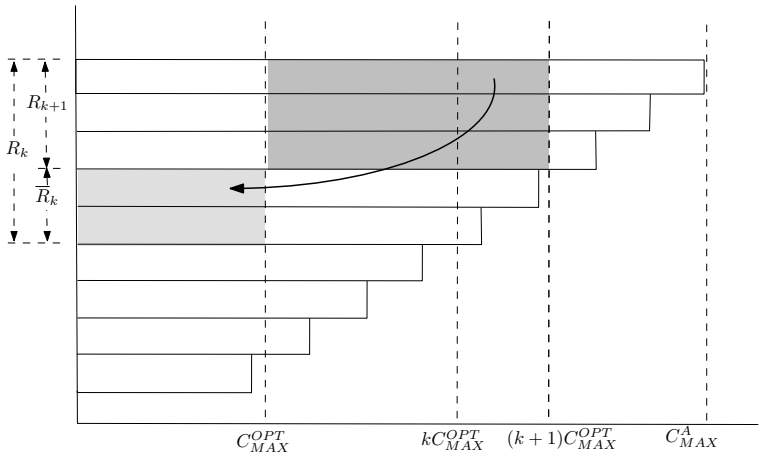

Fig. 2. Illustration of definition of R_k^A

Theorem 6. *The performance guarantee of lexjump optimal assignments for the problem of minimizing the makespan on restricted related parallel machines is*

$$O\left(\frac{\log \sum_i \widetilde{s_i}}{\log\log \sum_i \widetilde{s_i}}\right)$$

where $\widetilde{s_i}$ denotes the relative speed, i. e., $\widetilde{s_i} = s_i/s^-$.

Proof: Consider a lexjump optimal assignment A with makespan $C^A_{\max}$. By definition of R_k^A, the total load of the machines in R_{k+1}^A is $\sum_{i\in R_{k+1}^A} s_i \Lambda_i^A \geq (k+1)\sum_{i\in R_{k+1}^A} s_i C^{OPT}_{\max}$. By Lemma 1, we know that any job assigned in A on a machine in R_{k+1}^A needs to be assigned on a machine in R_k^A in an optimal assignment. Therefore, in any optimal assignment, the machines in $\overline{R_k^A}$ need to process at least a load of $k\sum_{i\in R_{k+1}^A} s_i\, C^{OPT}_{\max}$. Hence, for all k, we have

$$\sum_{i\in\overline{R_k^A}} s_i\, C^{OPT}_{\max} \;\geq\; \sum_{i\in R_{k+1}^A} k\, s_i\, C^{OPT}_{\max} \tag{21}$$

from which it follows that

$$\sum_{i\in R_k^A} s_i = \sum_{i\in\overline{R_k^A}} s_i + \sum_{i\in R_{k+1}^A} s_i \geq k \sum_{i\in R_{k+1}^A} s_i + \sum_{i\in R_{k+1}^A} s_i = (k+1)\sum_{i\in R_{k+1}^A} s_i. \tag{22}$$

Letting $c = \lfloor C^A_{\max}/C^{OPT}_{\max}\rfloor$, it follows that $\sum_{i\in R_0^A} s_i \geq c!\sum_{i\in R_c^A} s_i$ and consequently $\sum_{i\in R_0^A}\widetilde{s_i} \geq c!\sum_{i\in R_c^A}\widetilde{s_i}$. Since, $R_0 = M$ and $\sum_{i\in R_c^A}\widetilde{s_i} \geq |R_c^A| \geq 1$ as $\widetilde{s_i} \geq 1\,\forall\, i \in M$ and as the critical machine is in R_c^A, we have $\sum_{i\in M}\widetilde{s_i} \geq c! = \Gamma(c+1)$. Using the fact that the inverse of the Gamma function is monotonically

increasing, the latter inequality yields, $C^A_{\max}/C^{OPT}_{\max} \leq c+1 \leq \Gamma^{-1}\left(\sum_{i\in M}\widetilde{s}_i\right)$ and finally

$$C^A_{\max}/C^{OPT}_{\max} = O\left(\log\sum_i \widetilde{s}_i/\log\log\sum_i \widetilde{s}_i\right). \tag{23}$$

□

Note that the bound given in the above theorem confirms to the bound given in Theorem 2 by setting $s_i = 1$ for all $i \in M$. The following example shows that there exist instances for which the bound of Theorem 6 is tight up to a constant factor.

Example 4. Let $k > 1$ be an arbitrary strictly positive integer and let $s > 1$. Consider the following instance and assignment. Each job has a processing requirement $p_j = s$. The machines are partitioned into $sk+1$ groups, $S_0, S_1, \ldots, S_{sk}$. Group S_0 consists of only one machine which has a speed of one. For $l = 1, \ldots, sk$, group S_l contains $k\Pi_{i=1}^{l-1}(sk-i)$ machines each having a processing speed of s. In assignment A, each machine in group S_l, for $l \geq 1$, processes $sk-l$ jobs. k jobs are assigned to the machine in S_0. Each job $j \in J_i^A$ with $i \in S_l$ has $\mathcal{M}_j = S_l \cup S_{l+1}$. A is lexjump optimal with makespan sk, whereas $C^{OPT}_{\max} = 1$. The optimal solution is attained by assigning one job to each machine $i \in S_l$ for $l \geq 1$ and leaving the machine in S_0 empty. Moreover,

$$\sum_i \widetilde{s}_i \leq 1 + sk\sum_{i=0}^{sk-1}\frac{(sk-1)!}{i!} \leq 1 + (sk)!\sum_{i=0}^{+\infty}\frac{1}{i!} \tag{24}$$

$$\leq 1 + e\,(sk)! \leq (sk+2)! = \Gamma\,(sk+3)\,. \tag{25}$$

where $\Gamma(n)$ denotes the gamma function, i. e., $\Gamma(n) = (n-1)!$ for some integer n. Hence, using the fact that the inverse of the Gamma function is monotonically increasing, $C^A_{\max}/C^{OPT}_{\max} = sk \geq \Gamma^{-1}\left(\sum_i \widetilde{s}_i\right) - 3$. Thus, we are able to provide instances for which $C^A_{\max}/C^{OPT}_{\max}$ is in $\Omega(\log\sum_i \widetilde{s}_i/\log\log\sum_i \widetilde{s}_i)$.

Gairing, Lücking, Mavronicolas and Monien [9] established the following result.

Theorem 7 (Gairing et al. [9]). *A lexjump optimal assignment for restricted related parallel machines has a performance guarantee of at most m.*

Furthermore, they provide an example which establishes a lower bound of $m-1$ on the performance guarantee. Theorems 6 and 7 both establish an upper bound on the performance guarantee of a lexjump optimal solution. It can be shown that neither result implies the other result, i. e., we can construct examples for which the bound of Theorem 6 is tight and the bound of Theorem 7 is not and vice versa.

5 Concluding Remarks

Since swap optimal assignments as well as push optimal assignments are both jump optimal, we have that Theorems 1, 3 and 5 directly carry over to the swap and the push neighborhood. Moreover, the Examples and 1, 2 and 3 are swap and push optimal. Hence, the tightness results for the jump neighborhood apply to the swap and the push neighborhood as well.

So far we have focused on the quality of the local optima with respect to four neighborhoods. It is also interesting to know the number of iterations that an iterative improvement procedure needs to find these local optima. Brucker, Hurink, and Werner [2], Hurkens and Vredeveld [16] and Schuurman and Vredeveld [26] gave bounds to find a jump optimal solution for the identical parallel machines and the related parallel machines environments when there are no eligibility constraints. Gairing, Lücking, Mavronicolas, Monien and Spirakis [11] and Feldmann, Gairing, Lücking, Monien and Rode [7] gave bounds on the number of iterations needed to find a lexjump optimal solution for identical and related parallel machines, respectively. However, the procedure of Feldmann et al. allows for non-improving jumps too, so it is not an iterative improvement procedure. We conjecture that an iterative improvement procedure exists which reaches a jump or lexjump optimal assignment in polynomial time. However, the proofs of the results stated above for reaching a jump or lexjump optimal assignment in polynomial time on unrestricted parallel machine scheduling cannot be generalized to restricted parallel machine scheduling in a straightforward manner.

References

1. B. Awerbuch, Y. Azar, Y. Richter, and D. Tsur. Tradeoffs in worst-case equilibria. *Theoretical Computer Science*, 361:200–209, 2006.
2. P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems - II. *Discrete Applied Mathematics*, 72:47–69, 1997.
3. T. Brueggemann, J. L. Hurink, T. Vredeveld, and G. J. Woeginger. Very large-scale neighborhoods with performance guarantees for minimizing makespan on parallel machines. In C. Kaklamanis and M. Skutella, editors, *Approximation and Online Algorithms (WAOA 2007)*, volume 2909 of *Lecture Notes in Computer Science*, pages 41–55. Springer, Berlin, 2008.
4. G. Centeno and R. L. Armacost. Minimizing makespan on parallel machines with release time and machine eligibility restrictions. *International Journal of Production Research*, 42(6):1243–1256, 2004.
5. Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9:91–103, 1980.
6. A. Czumaj and B. Vocking. Tight bounds for worst-case equilibria. *ACM Transactions on Algorithms*, 3(1):4, 2007.
7. R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Nashification and the coordination ratio for a selfish routing game. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003*, volume 2719, pages 514–526, 2003.

8. G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT*, 19:312–320, 1979.
9. M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. The price of anarchy for restricted parallel links. *Parallel Processing Letters*, 16(1):117–131, 2006.
10. M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing nash equilibria for scheduling on restricted parallel links. *Theory of Computing Systems*, 2009. Published online on 12 February 2009 at http://www.springerlink.com/content/q7n8267q76716v55/fulltext.pdf.
11. M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and P. Spirakis. Structure and complexity of extreme nash equilibria. *Theoretical Computer Science*, 343:133–157, 2005.
12. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
13. C. A. Glass and H. Kellerer. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics*, 54(3):250–257, 2007.
14. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
15. M. Hoefer and A. Souza. The influence of link restriction on (random) selfish routing. *Lecture Notes in Computer Science; in Proceedings of the First Symposium on Algorithmic Game Theory (SAGT 2008)*, 4997:22–32, 2008.
16. C. A. J. Hurkens and T. Vredeveld. Local search for multiprocessor scheduling: how many moves does it take to a local optimum? *Operations Research Letters*, 31:137–141, 2003.
17. N. Immorlica, L. Li, V. Mirrokni, and A. Schulz. Coordination mechanism for selfish scheduling. *Theoretical Computer Science*, 410:1589–1598, 2009.
18. J. Y. T. Leung and C. L. Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116:251–262, 2008.
19. C. L. Li. Scheduling unit-length jobs with machine eligibility restrictions. *European Journal of Operational Research*, 174:1325–1328, 2006.
20. L. W. Liao and G. J. Sheen. Parallel machine scheduling with machine availability and eligibility constraints. *European Journal of Operational Research*, 184:458–467, 2008.
21. Y. Lin and W. Li. Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 156:261–266, 2004.
22. P. Lu and C. Yu. Worst-case nash equilibria in restricted routing. *Lecture Notes in Computer Science; Proceedings of the 4th International Workshop on Internet and Network Economics*, 5385:231–238, 2008.
23. W. Michiels, E. Aarts, and J. Korst. *Theoretical Aspects of Local Search*. Springer-Verlag, Berlin, 2007.
24. J. Ou, J. Y. T. Leung, and C. L. Li. Scheduling parallel machines with inclusive set restrictions. *Naval Research Logistics*, 55(4):328–338, 2008.
25. M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Springer, New York, 3rd edition, 2008. Original edition published by Prentice Hall, Englewood Cliffs, NJ, 1995.
26. P. Schuurman and T. Vredeveld. Performance guarantees of local search for multiprocessor scheduling. *INFORMS Journal of Computing*, 19(1):52–63, 2007.
27. E.V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(8):2266–2278, 2005.

28. G. L. Vairaktarakis and X. Cai. The value of processing flexibility in multipurpose machines. *IIE Transactions*, 35(8):763–774, 2003.
29. B. Vöcking. Selfish load balancing. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 20. Cambridge University Press, New York, NY, USA, 2007.