Convex Quadratic Programming Relaxations for Network Scheduling Problems[*]

Martin Skutella[**]

Technische Universität Berlin
`skutella@math.tu-berlin.de`
`http://www.math.tu-berlin.de/~skutella/`

Abstract. In network scheduling a set of jobs must be scheduled on unrelated parallel processors or machines which are connected by a network. Initially, each job is located on some machine in the network and cannot be started on another machine until sufficient time elapses to allow the job to be transmitted there. This setting has applications, e. g., in distributed multi-processor computing environments and also in operations research; it can be modeled by a standard parallel machine environment with machine-dependent release dates. We consider the objective of minimizing the total weighted completion time.
The main contribution of this paper is a provably good convex quadratic programming relaxation of strongly polynomial size for this problem. Until now, only linear programming relaxations in time- or interval-indexed variables have been studied. Those LP relaxations, however, suffer from a huge number of variables. In particular, the best previously known relaxation is of exponential size and can therefore not be solved exactly in polynomial time. As a result of the convex quadratic programming approach we can give a very simple and easy to analyze randomized 2–approximation algorithm which slightly improves upon the best previously known approximation result. Furthermore, we consider preemptive variants of network scheduling and derive approximation results and results on the power of preemption which improve upon the best previously known results for these settings.

1 Introduction

We study the following parallel machine scheduling problem. A set J of n jobs has to be scheduled on m unrelated parallel machines which are connected by a network. The jobs continually arrive over time and each job originates at some node of the network. Therefore, before a job can be processed on another machine, it must take the time to travel there through the network. This is

[*] © Springer-Verlag. To appear in the proceedings of the 7th Annual European Symposium on Algorithms (ESA'99).

[**] This research was partially supported by DONET within the frame of the TMR Programme (contract number ERB FMRX-CT98-0202) while the author was staying at C.O.R.E., Louvain-la-Neuve, Belgium, for the academic year 1998/99.

modeled by machine-dependent release dates $r_{ij} \geq 0$ which denote the earliest point in time when job j may be processed on machine i. Together with each job j we are given its positive processing requirement which also depends on the machine i job j will be processed on and is therefore denoted by p_{ij}. Each job j must be processed for the respective amount of time without interruption on one of the m machines, and may be assigned to any of them. However, for a given job j it may happen that $p_{ij} = \infty$ for some (but not all) machines i such that job j cannot be scheduled on those machines. Every machine can process at most one job at a time. This network scheduling model has been introduced in [4, 1].

We denote the completion time of job j by C_j. The goal is to minimize the total weighted completion time: a weight $w_j \geq 0$ is associated with each job j and we seek to minimize $\sum_{j \in J} w_j C_j$. In scheduling, it is quite convenient to refer to the respective problems using the standard classification scheme of Graham, Lawler, Lenstra, and Rinnooy Kan [7]. The problem $\mathrm{R}\,|\,r_{ij}\,|\sum w_j C_j$, just described, is strongly NP-hard, even for the special case of two identical parallel machines without nontrivial release dates, see [2, 12].

Since we cannot hope to be able to compute optimal schedules in polynomial time, we are interested in how close one can approach the optimum in polynomial time. A (randomized) α–approximation algorithm computes in polynomial time a feasible solution to the problem under consideration whose (expected) value is bounded by α times the value of an optimal solution; α is called the performance guarantee or performance ratio of the algorithm. All randomized approximation algorithms that we discuss or present can be derandomized by standard methods; therefore we will not go into the details of derandomization.

The first approximation result for the scheduling problem $\mathrm{R}\,|\,r_{ij}\,|\sum w_j C_j$ was obtained by Phillips, Stein, and Wein [15] who gave an algorithm with performance guarantee $O(\log^2 n)$. The first constant factor approximation was developed by Hall, Shmoys, and Wein [9] (see also [8]) whose algorithm achieves performance ratio $\frac{16}{3}$. Generalizing a single machine approximation algorithm of Goemans [6], this result was then improved by Schulz and Skutella [18] to a $(2 + \varepsilon)$–approximation algorithm. All those approximation results rely somehow on (integer) linear programming formulations or relaxations in time-indexed variables. In the following discussion we assume that all processing times and release dates are integral; furthermore, we define $p_{\max} := \max_{i,j} p_{ij}$.

Phillips, Stein, and Wein modeled the network scheduling problem as a hypergraph matching problem by matching each job j to p_{ij} consecutive time intervals of length 1 on a machine i. The underlying graph contains a node for each job and each pair formed by a machine and a time interval $[t, t+1)$ where t is integral and can achieve values in a range of size np_{max}. Therefore, since $p_{\max}$ may be exponential in the input size, the corresponding integer linear program contains exponentially many variables as well as exponentially many constraints. Phillips et al. eluded this problem by partitioning the set of jobs into groups such that the jobs in each group can be scaled down to polynomial size. However, this complicates both the design and the analysis of their approximation algorithm.

The result of Hall, Shmoys, and Wein is based on a polynomial variant of time-indexed formulations which they called *interval-indexed*. The basic idea is to replace the intervals of length 1 by time intervals $[2^k, 2^{k+1})$ of geometrically increasing size. The decision variables in the resulting linear programming relaxation then indicate on which machine and in which time interval a given job completes. Notice, however, that one looses already at least a factor of 2 in this formulation since the interval-indexed variables do not allow a higher precision for the completion times of jobs. The approximation algorithm of Hall et al. relies on Shmoys and Tardos' rounding technique for the generalized assignment problem [20].

Schulz and Skutella generalized an LP relaxation in time-indexed variables that was introduced by Dyer and Wolsey [5] for the corresponding single machine scheduling problem. It contains a decision variable for each triple formed by a job, a machine, and a time interval $[t, t+1)$ which indicates whether the job is being processed in this time interval on the respective machine. The resulting LP relaxation is a 2–relaxation of the scheduling problem under consideration, i. e., the optimum LP value is within a factor 2 of the value of an optimal schedule. However, as the formulation of Phillips et al., this relaxation suffers from an exponential number of variables and constraints. One can overcome this drawback by turning again to interval-indexed variables. However, in order to ensure a higher precision, Schulz and Skutella used time intervals of the form $[(1+\varepsilon)^k, (1+\varepsilon)^{k+1})$ where $\varepsilon > 0$ can be chosen arbitrarily small; this leads to a $(2+\varepsilon)$–relaxation of polynomial size. Notice, however, that the size of the relaxation still depends substantially on $p_{\max}$ and may be huge for small values of ε. The approximation algorithm based on this LP relaxation uses a randomized rounding technique.

For the problem of scheduling unrelated parallel machines in the absence of nontrivial release dates $\mathrm{R}\,|\;|\sum w_j C_j$, the author has introduced a convex quadratic programming relaxation that leads to a simple $\frac{3}{2}$–approximation algorithm [22]. One of the basic observations for this result is that in the absence of nontrivial release dates the parallel machine problem can be reduced to an assignment problem of jobs to machines; for a given assignment of jobs to machines the sequencing of the assigned jobs can be done optimally on each machine i by applying Smith's Ratio Rule [24]: schedule the jobs in order of nonincreasing ratios w_j/p_{ij}. Therefore, the problem can be formulated as an integer quadratic program in assignment variables. An appropriate relaxation of this program together with randomized rounding leads to the approximation result mentioned above. Independently, the same result has later also been derived by Jay Sethuraman and Mark S. Squillante [19].

Unfortunately, for the general network scheduling problem including release dates the situation is more complicated; for a given assignment of jobs to machines, the sequencing problem on each machine is still strongly NP-hard, see [12]. However, we know that in an optimal schedule a 'violation' of Smith's Ratio Rule can only occur after a new job has been released; in other words, whenever two successive jobs on machine i can be exchanged without violating release dates, the job with the higher ratio w_j/p_{ij} will be processed first in an optimal schedule. Therefore, the sequencing of jobs that are being processed between two successive release dates can be done optimally by Smith's Ratio Rule. We make use of this insight by partitioning the processing on each machine i into n *time slots* which are essentially defined by the n release dates r_{ij}, $j \in J$; since the sequencing of jobs in each time slot is easy, we have to solve an assignment problem of jobs to time slots and can apply similar ideas as in [22]. In particular, we derive a convex quadratic programming relaxation in n^2m assignment variables and $O(nm)$ constraints. Randomized rounding based on an optimal solution to this relaxation finally leads to a very simple and easy to analyze 2–approximation algorithm for network scheduling.

Our technique can be extended to network scheduling problems with preemptions. In preemptive scheduling, a job may repeatedly be interrupted and continued later on another (or the same) machine. In the context of network scheduling it is reasonable to assume that after a job has been interrupted on one machine, it cannot immediately be continued on another machine; it must again take the time to travel there through the network. We call the delay caused by such a transfer *communication delay.* In a similar context, communication delays between precedence constrained jobs have been studied, see, e. g., [14].

We give a 3–approximation algorithm for the problem $\mathrm{R} \,|\, r_{ij},\, pmtn \,|\, \sum w_j C_j$ that, in fact, does not make use of preemptions but computes nonpreemptive schedules. Therefore, this approximation result also holds for preemptive network scheduling with arbitrary communication delays. Moreover, it also implies a bound on the power of preemption, i. e., one cannot gain more than a factor 3 by allowing preemptions. For the problem without nontrivial release dates $\mathrm{R} \,|\, pmtn \,|\, \sum w_j C_j$, the same technique yields a 2–approximation algorithm. For the preemptive scheduling problems without communication delays, Phillips, Stein, and Wein [16] gave an $(8 + \varepsilon)$–approximation. In [21] the author has achieved slightly worse results than those presented here, based on a time-indexed LP relaxation in the spirit of [18].

The paper is organized as follows. In the next section we introduce the concept of scheduling in time slots. We give an integer quadratic programming formulation of the network scheduling problem in Section 3 and show how it can be relaxed to a convex quadratic program. In Section 4 we present a simple 2–approximation algorithm and prove a bound on the quality of the convex quadratic programming relaxation. Finally, in Section 5, we briefly sketch the results and techniques for preemptive network scheduling.

Due to space limitations, we do not provide proofs in this extended abstract; we refer to the full paper [23] which combines [22] and the paper at hand and can be found on the authors homepage.

2 Scheduling in time slots

The main idea of our approach for the scheduling problem $\mathrm{R} \,|\, r_{ij} \,|\, \sum w_j C_j$ is to somehow get rid of the release dates of jobs. We do this by partitioning time on each machine i into several time slots. Each job is being processed on one machine in one of its time slots and we make sure that job j can only be processed in a slot that starts after its release date.

Let $\rho_{i_1} \leq \rho_{i_2} \leq \cdots \leq \rho_{i_n}$ be an ordering of the release dates r_{ij}, $j \in J$; moreover, we set $\rho_{i_{n+1}} := \infty$. For a given feasible schedule we say that i_k, the k^{th} time slot on machine i, contains all jobs j that are started within the interval $[\rho_{i_k}, \rho_{i_{k+1}})$ on machine i; we denote this by $j \in i_k$. We may assume that there is no idle time between the processing of jobs in one time slot, i. e., all jobs in a slot are processed one after another without interruption.

Moreover, as a consequence of Smith's Ratio Rule we can restrict to schedules where the jobs in time slot i_k are sequenced in order of nonincreasing ratios w_j/p_{ij}. Throughout the paper we will use the following convention: whenever we apply Smith's Ratio Rule in a time slot on machine i and $w_k/p_{ik} = w_j/p_{ij}$ for a pair of jobs j, k, the job with smaller index is scheduled first. For each machine $i = 1, \ldots, m$ we define a corresponding total order $(J, \prec_i)$ on the set of jobs by setting $j \prec_i k$ if either $w_j/p_{ij} > w_k/p_{ik}$ or $w_j/p_{ij} = w_k/p_{ik}$ and $j < k$.

Lemma 1. *In an optimal solution to the scheduling problem under consideration, the jobs in each time slot i_k are scheduled without interruption in order of nondecreasing ratios w_j/p_{ij}. Furthermore, there exists an optimal solution where the jobs are sequenced according to $\prec_i$ in each time slot i_k.*

Notice that there may be several empty time slots i_k. This happens in particular if $\rho_{i_k} = \rho_{i_{k+1}}$. Therefore it would be sufficient to introduce only q_i time slots for machine i where q_i is the number of different values r_{ij}, $j \in J$. For example, if there are no nontrivial release dates (i. e., $r_{ij} = 0$ for all i and j), we only need to introduce one time slot $[0, \infty)$ on each machine. The problem $\mathrm{R} \mid \mid \sum w_j C_j$ has been considered in [22]; for this special case our approach coincides with the one given there.

Up to now we have described how a feasible schedule can be interpreted as a feasible assignment of jobs to time slots. We call an assignment *feasible* if each job j is being assigned to a time slot i_k with $\rho_{i_k} \geq r_{ij}$. On the other hand, for a given feasible assignment of the jobs in J to time slots we can easily construct a corresponding feasible schedule: Sequence the jobs in time slot i_k according to $\prec_i$ and start it as early as possible after the jobs in the previous slot on machine i are finished but not before ρ_{i_k}; in other words, the starting time s_{i_k} of time slot i_k is given by $s_{i_1} := \rho_{i_1}$ and $s_{i_{k+1}} := \max\{\rho_{i_{k+1}}, s_{i_k} + \sum_{j \in i_k} p_{ij}\}$, for $k = 1, \ldots, n-1$.

Lemma 2. *Given its assignment of jobs to time slots, we can reconstruct an optimal schedule meeting the properties described in Lemma 1.*

We close this section with one final remark. Notice that several feasible assignments of jobs to time slots may lead to the same feasible schedule. Consider, e. g., an instance consisting of three jobs of unit length and unit weight that have to be scheduled on a single machine. Jobs 1 and 2 are released at time 0, while job 3 becomes available at time 1. We get an optimal schedule by processing the jobs without interruption in order of increasing numbers. This schedule corresponds to five different feasible assignments of jobs to time slots. We can assign job 1 to one of the first two slots, job 2 to the same or a later slot, and finally job 3 to slot 3.

3 A convex quadratic programming relaxation

As a consequence of Lemma 2 we have reduced the scheduling problem under consideration to finding an optimal assignment of jobs to time slots. Therefore we can give a formulation of $R \mid r_{ij} \mid \sum w_j C_j$ in assignment variables $a_{i_k j} \in \{0,1\}$ where $a_{i_k j} = 1$ if job j is being assigned to time slot i_k, and $a_{i_k j} = 0$ otherwise. This leads to the following integer quadratic program:

$$
\begin{array}{llll}
\text{minimize} & \sum_j w_j C_j & & \\
\text{subject to} & \sum_{i,k} a_{i_k j} = 1 & \text{for all } j & (1) \\
& s_{i_1} = \rho_{i_1} & \text{for all } i & (2) \\
& s_{i_{k+1}} = \max\{\rho_{i_{k+1}}, s_{i_k} + \sum_j a_{i_k j} p_{ij}\} & \text{for all } i,\ k & (3) \\
& C_j = \sum_{i,k} a_{i_k j} \Big(s_{i_k} + p_{ij} + \sum_{j' \prec_i j} a_{i_k j'} p_{ij'} \Big) & \text{for all } j & (4) \\
& a_{i_k j} = 0 & \text{if } \rho_{i_k} < r_{ij} & (5) \\
& a_{i_k j} \in \{0,1\} & \text{for all } i,\ k,\ j &
\end{array}
$$

Constraints (1) ensure that each job is being assigned to exactly one time slot. In constraints (2) and (3) we set the starting times of the time slots as described in Section 2. If job j is being assigned to time slot i_k, its completion time is the sum of the starting time s_{i_k} of this slot, its own processing time p_{ij}, and the processing times of other jobs $j' \prec_i j$ that are also scheduled in this time slot. The right hand side of (4) is the sum of these expressions over all time slots i_k weighted by $a_{i_k j}$; it is thus equal to the completion time of j. Finally, constraints (5) ensure that no job is being processed before its release date.

It follows from our considerations in Section 2 that we could replace (5) by the stronger constraint

$$
a_{i_k j} = 0 \qquad\qquad \text{if } \rho_{i_k} < r_{ij} \text{ or } \rho_{i_k} = \rho_{i_{k+1}}
$$

which reduces the number of available time slots on each machine. For the special case $R \mid \mid \sum w_j C_j$ this leads to the integer quadratic program that has been introduced in [22]. It is also shown there that it is still NP-hard to solve the continuous relaxation of this integer quadratic program; however, it can be solved in polynomial time if the term p_{ij} on the right hand side of (4) is replaced by $p_{ij}(1 + a_{i_k j})/2$.

Observe that this replacement does not affect the value of the integer quadratic program since the new term is equal to p_{ij} whenever $a_{i_k j} = 1$. This motivates the study of the following quadratic programming relaxation (QP) for the general problem including release dates:

$$
\begin{array}{llll}
 & \text{minimize} & \sum_j w_j C_j & \\
 & \text{subject to} & \sum_{i,k} a_{i_k j} = 1 & \text{for all } j \\
 & & s_{i_1} = \rho_{i_1} & \text{for all } i \qquad (6) \\
(QP) & & s_{i_{k+1}} = \max\{\rho_{i_{k+1}}, s_{i_k} + \sum_j a_{i_k j} p_{ij}\} & \text{for all } i,\ k \quad (7) \\
 & & C_j = \sum_{i,k} a_{i_k j}\Big(s_{i_k} + \frac{1 + a_{i_k j}}{2} p_{ij} + \sum_{j' \prec_i j} a_{i_k j'} p_{ij'}\Big) & \text{for all } j \qquad (8) \\
 & & a_{i_k j} = 0 & \text{if } \rho_{i_k} < r_{ij} \\
 & & a_{i_k j} \geq 0 & \text{for all } i,\ k,\ j
\end{array}
$$

Notice that a solution to this program is uniquely determined by giving the values of the assignment variables $a_{i_k j}$. In contrast to the case without nontrivial release dates, we cannot directly prove that this quadratic program is convex. Nevertheless, in the remaining part of this section we will show that it can be solved in polynomial time. The main idea is to show that one can restrict to solutions satisfying $s_{i_k} = \rho_{i_k}$ for all i and k. Adding these constraints to (QP) then leads to a convex quadratic program.

Lemma 3. *For all instances of* $\mathrm{R} \,|\, r_{ij} \,|\, \sum w_j C_j$ *there exists an optimal solution to* (QP) *satisfying* $s_{i_k} = \rho_{i_k}$ *for all* i *and* k.

As a consequence of Lemma 3 we can replace the variables s_{i_k} in (QP) by the constants ρ_{i_k} by changing constraints (7) to

$$\sum_j a_{i_k j} p_{ij} \leq \rho_{i_{k+1}} - \rho_{i_k} \qquad \text{for all } i,\ k.$$

Furthermore, if we remove constraints (8) and replace C_j in the objective function by the right hand side of (8), we can reformulate the quadratic programming relaxation as follows:

$$
\begin{array}{llll}
 & \text{minimize} & b^T a + \tfrac{1}{2} a^T D a & \qquad (9) \\
 & \text{subject to} & \sum_{i,k} a_{i_k j} = 1 & \text{for all } j \qquad (10) \\
(CQP) & & \sum_j a_{i_k j} p_{ij} \leq \rho_{i_{k+1}} - \rho_{i_k} & \text{for all } i,\ k \quad (11) \\
 & & a_{i_k j} = 0 & \text{if } \rho_{i_k} < r_{ij} \\
 & & a \geq 0 &
\end{array}
$$

Here, $a \in \mathbb{R}^{mn^2}$ denotes the vector consisting of all variables $a_{i_k j}$ lexicographically ordered with respect to the natural order $1_1, 1_2, \ldots, m_n$ of the time slots and then, for each slot i_k, the jobs ordered according to $\prec_i$. The vector $b \in \mathbb{R}^{mn^2}$ is given by $b_{i_k j} = \frac{1}{2} w_j p_{ij} + w_j \rho_{i_k}$, and $D = \big(d_{(i_k j)(i'_{k'} j')}\big)$ is a symmetric $mn^2 \times mn^2$–matrix given through

$$d_{(i_k j)(i'_{k'}, j')} = \begin{cases} 0 & \text{if } i_k \neq i'_{k'} \,, \\ w_{j'} p_{ij} & \text{if } i_k = i'_{k'} \text{ and } j \prec_i j' \,, \\ w_j p_{ij'} & \text{if } i_k = i'_{k'} \text{ and } j' \prec_i j \,, \\ w_j p_{ij} & \text{if } i_k = i'_{k'} \text{ and } j = j' \,. \end{cases}$$

Because of the lexicographic order of the indices the matrix D is decomposed into mn diagonal blocks corresponding to the mn time slots. If we assume that the jobs are indexed according to $\prec_i$ and if we denote p_{ij} simply by p_j, each block corresponding to a time slot on machine i has the following form:

$$\begin{pmatrix} w_1p_1 & w_2p_1 & w_3p_1 & \cdots & w_np_1 \\ w_2p_1 & w_2p_2 & w_3p_2 & \cdots & w_np_2 \\ w_3p_1 & w_3p_2 & w_3p_3 & \cdots & w_np_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_np_1 & w_np_2 & w_np_3 & \cdots & w_np_n \end{pmatrix}$$

It has been observed in [22] that those matrices are positive semidefinite and therefore the whole matrix D is positive semidefinite. In particular, the objective function (9) is convex and the quadratic programming relaxation can be solved in polynomial time, see, e. g., [11, 3].

The convex quadratic programming relaxation (CQP) is in some sense similar to the linear programming relaxation in time-indexed variables that has been introduced in [18]. Without going into the details, we give a rough idea of the common underlying intuition of both relaxations: a job may be split into several parts (corresponding to fractional values $a_{i_k j}$ in (CQP)) who can be scattered over the machines and over time. The completion time of a job in such a 'fractional schedule' is somehow related to its mean busy time; the mean busy time of a job is the average point in time at which its fractions are being processed (see (8) where C_j is set to the average over the terms in brackets on the right hand side weighted by $a_{i_k j}$). However, in contrast to the time-indexed LP relaxation, the construction of the convex quadratic program (CQP) contains more insights into the structure of an optimal schedule. As a result, (CQP) is of strongly polynomial size while the LP relaxation contains an exponential number of time-indexed variables and constraints.

4 A simple 2–approximation algorithm

The value of an optimal solution to the convex quadratic programming relaxation (CQP) of the last section is a lower bound on the value of an optimal schedule.

Moreover, from the structure of an optimal solution to the relaxation we can gain important insights that turn out to be useful in the construction of a provably good solution to the scheduling problem under consideration. In this context, randomized rounding has proved to be a powerful algorithmic tool. On the one hand, it yields very simple and easy to analyze algorithms; on the other hand, it is able to minutely capture the structure of the solution to the relaxation and to carry it over to a feasible schedule. The idea of using randomized rounding in the study of approximation algorithms was introduced by Raghavan and Thompson [17], an overview can be found in [13].

For a given optimal solution a to (CQP), we compute an integral solution $\bar{a}$ by setting for each job j exactly one of the variables $\bar{a}_{i_k j}$ to 1 with probabilities given through $a_{i_k j}$. Notice that $0 \leq a_{i_k j} \leq 1$ and the sum of the $a_{i_k j}$ for job j is equal to one by constraints (10). Although the integral solution $\bar{a}$ does not necessarily fulfill constraints (11), it represents a feasible assignment of jobs to time slots, i.e., a feasible solution to (QP), and thus a feasible schedule. For our analysis we require that the random choices are performed pairwise independently for the jobs.

Theorem 1. *Computing an optimal solution to (CQP) and using randomized rounding to turn it into a feasible schedule is a 2–approximation algorithm for the problem $\mathrm{R}\,|\,r_{ij}\,|\sum w_j C_j$.*

Theorem 1 follows from the next lemma which gives a slightly stronger result including job-by-job bounds.

Lemma 4. *Using randomized rounding in order to turn an arbitrary feasible solution to (QP) into a feasible assignment of jobs to time slots yields a schedule such that the expected completion time of each job is bounded by twice the corresponding value (8) in the given solution to (QP).*

Since the value of an optimal solution to (CQP) is a lower bound on the value of an optimal schedule, Theorem 1 follows from Lemma 4 and linearity of expectations.

Our result on the quality of the computed schedule described in Theorem 1 also implies a bound on the quality of the quadratic programming relaxation that served as a lower bound in our estimations.

Corollary 1. *For instances of $\mathrm{R}\,|\,r_{ij}\,|\sum w_j C_j$, the value of an optimal solution to the relaxation (CQP) is within a factor 2 of the value of an optimal schedule. This bound is tight even for the case of identical parallel machines without release dates $\mathrm{P}\,|\,|\sum w_j C_j$.*

5 Extensions to scheduling with preemptions

In this section we discuss the preemptive problem $\mathrm{R}\,|\,r_{ij}, pmtn\,|\sum w_j C_j$ and generalizations to network scheduling. In contrast to the nonpreemptive setting, a job may now repeatedly be interrupted and continued later on another (or the same) machine. In the context of network scheduling, it is reasonable to assume that after a job has been interrupted on one machine it cannot be continued on another machine until a certain communication delay is elapsed that allows the job to travel through the network to its new machine.

The ideas and techniques presented in the last section can be generalized to this setting. However, since we have to use a somewhat weaker relaxation in order to capture the possibility of preemptions, we only get a 3–approximation algorithm. This result can be improved to performance guarantee 2 in the absence of nontrivial release dates $\mathrm{R}\,|\,pmtn\,|\sum w_j C_j$ but with arbitrary communication delays. For reasons of brevity we only give a brief sketch of the main differences to the nonpreemptive setting.

Although the quadratic program (QP) allows to break a job into fractions and thus to preempt it by choosing fractional values $a_{i_k j}$, it is not a relaxation of $\mathrm{R}\,|\,r_{ij},\,pmtn\,|\sum w_j C_j$. However, we can turn it into a relaxation by replacing (8) with the weaker constraint

$$C_j = \sum_{i,k} a_{i_k j}\Big(s_{i_k} + \frac{a_{i_k j}}{2} p_{ij} + \sum_{j' \prec_i j} a_{i_k j'} p_{ij'}\Big) \qquad \text{for all } j.$$

Moreover, we restrengthen the relaxation by adding the following constraint

$$\sum_j w_j C_j \geq \sum_j w_j \sum_{i,k} a_{i_k j} p_{ij}$$

which bounds the objective value from below by the weighted sum of processing times (a similar constraint has already been used in [22]). Since Lemma 3 can be carried over to the new setting, we again get a convex quadratic programming relaxation.

In order to turn an optimal solution to this relaxation into a feasible schedule, we apply exactly the same randomized rounding heuristic as in the nonpreemptive case. In particular, we do not make use of the possibility to preempt jobs but compute a nonpreemptive schedule. Therefore, our results hold for the case of arbitrary communication delays.

Theorem 2. *Randomized rounding based on an optimal solution to the convex quadratic programming relaxation yields a 3–approximation algorithm for* $\mathrm{R}\,|\,r_{ij},\,pmtn\,|\sum w_j C_j$ *and a 2–approximation algorithm for* $\mathrm{R}\,|\,pmtn\,|\sum w_j C_j$*, even for the case of arbitrary communication delays. The same bounds hold for the quality of the relaxation.*

Theorem 2 also implies bounds on the power of preemption. Since we can compute a nonpreemptive schedule whose value is bounded by 3 respectively 2 times the value of an optimal preemptive schedule, we have derived upper bounds on the ratios of optimal nonpreemptive to optimal preemptive schedules.

Corollary 2. *For instances of* $\mathrm{R}\,|\,r_{ij}\,|\sum w_j C_j$*, the value of an optimal nonpreemptive schedule is at most a factor* 3 *above the value of an optimal preemptive schedule. In the absence of nontrivial release dates, this bound can be improved to* 2*.*

6 Conclusion

We have presented convex quadratic programming relaxations of strongly polynomial size which lead to simple and easy to analyze approximation algorithms for preemptive and nonpreemptive network scheduling. Although our approach and the presented results might be at first sight of mainly theoretical interest, we hope that nonlinear relaxations like the one we discuss in this paper will also prove useful in solving real world scheduling problems in the near future. With the development of better algorithms that solve convex quadratic programs more efficiently in practice, the results obtained by using such relaxations might become comparable or even better than those based on linear programming relaxations with a huge number of time-indexed variables and constraints.

Precedence constraints between jobs play a particularly important role in most real world scheduling problems. Therefore it would be both of theoretical and of practical interest to incorporate those constraints into our convex quadratic programming relaxation.

Hoogeveen, Schuurman, and Woeginger [10] have shown that the problems $\mathrm{R}\,|\,r_j\,|\sum C_j$ and $\mathrm{R}\,|\,|\sum w_j C_j$ cannot be approximated in polynomial time within arbitrarily good precision, unless P=NP. It is an interesting open problem to close the gap between this negative result and the 2–approximation algorithm presented in this paper.

References

1. B. Awerbuch, S. Kutten, and D. Peleg. Competitive distributed job scheduling. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 571 – 581, 1992.
2. J. L. Bruno, E. G. Coffman Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the Association for Computing Machinery*, 17:382 – 387, 1974.
3. S. J. Chung and K. G. Murty. Polynomially bounded ellipsoid algorithms for convex quadratic programming. In O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, editors, *Nonlinear Programming 4*, pages 439 – 485. Academic Press, 1981.
4. X. Deng, H. Liu, J. Long, and B. Xiao. Deterministic load balancing in computer networks. In *Proceedings of the 2nd Annual IEEE Symposium on Parallel and Distributed Processing*, pages 50 – 57, 1990.
5. M. E. Dyer and L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26:255 – 270, 1990.
6. M. X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 591 – 598, 1997.
7. 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.

8. L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off–line and on–line approximation algorithms. *Mathematics of Operations Research*, 22:513 – 544, 1997.
9. L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off–line and on–line algorithms. In *Proceedings of the 7th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 142 – 151, 1996.
10. H. Hoogeveen, P. Schuurman, and G. J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 353 – 366. Springer, Berlin, 1998.
11. M. K. Kozlov, S. P. Tarasov, and L. G. Hačijan. Polynomial solvability of convex quadratic programming. *Soviet Mathematics Doklady*, 20:1108 – 1111, 1979.
12. J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343 – 362, 1977.
13. R. Motwani, J. Naor, and P. Raghavan. Randomized approximation algorithms in combinatorial optimization. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, chapter 11, pages 447 – 481. Thomson, 1996.
14. C. H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, 19:322 – 328, 1990.
15. C. Phillips, C. Stein, and J. Wein. Task scheduling in networks. *SIAM Journal on Discrete Mathematics*, 10:573 – 598, 1997.
16. C. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199 – 223, 1998.
17. P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365 – 374, 1987.
18. A. S. Schulz and M. Skutella. Scheduling–LPs bear probabilities: Randomized approximations for min–sum criteria. In R. Burkard and G. J. Woeginger, editors, *Algorithms – ESA '97*, volume 1284 of *Lecture Notes in Computer Science*, pages 416 – 429. Springer, Berlin, 1997.
19. J. Sethuraman and M. S. Squillante. Optimal scheduling of multiclass prallel machines. In *Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 963 – 964, 1999.
20. D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461 – 474, 1993.
21. M. Skutella. *Approximation and Randomization in Scheduling.* PhD thesis, Technical University of Berlin, Germany, 1998.
22. M. Skutella. Semidefinite relaxations for parallel machine scheduling. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 472 – 481, 1998.
23. M. Skutella. Convex Quadratic and Semidefinite Programming Relaxations in Scheduling. Manuscript, 1999.
24. W. E. Smith. Various optimizers for single–stage production. *Naval Research and Logistics Quarterly*, 3:59 – 66, 1956.