Experimental comparison of approximation algorithms for scheduling unrelated parallel machines

Vredeveld, T.; Hurkens, C.A.J.

Published: 01/01/2001

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 26. Oct. 2017
SPOR-Report 2001-03
Experimental comparison of approximation algorithms for scheduling unrelated parallel machines

T. Vredeveld
C.A.J. Hurkens

/ department of mathematics and computing science
Experimental comparison of approximation algorithms for scheduling unrelated parallel machines

Tjark Vredeveld Cor Hurkens
tjark@win.tue.nl wscor@win.tue.nl
Department of Mathematics and Computing Science
Technische Universiteit Eindhoven
P.O.Box 513
5600 MB Eindhoven

Abstract

This paper presents an empirical comparison of polynomial-time approximation algorithms and local search heuristics for the problem of minimizing total weighted completion time on unrelated parallel machines. Algorithms with a worst-case performance guarantee are based on rounding a fractional solution to an LP-relaxation or to a convex quadratic programming relaxation. We also investigate dominance relations among the lower bounds resulting from these relaxations.

1 Introduction

In this paper, we make an experimental comparison of approximation algorithms for which we have constant performance bounds but no empirical evidence, and local search heuristics which exhibit good empirical behavior but for which we have no performance bounds. The former algorithms are so-called polynomial-time $p$-approximation algorithms, which are algorithms that compute a feasible solution in polynomial time whose value is within a factor $p$ of the optimal solution value; $p$ is called the performance guarantee of the algorithm.

The problem under consideration is the problem of scheduling unrelated parallel machines so as to minimize the total weighted completion time. We are given a set of $n$ jobs, $J_1, \ldots, J_n$, each of which has to be scheduled without interruption on one of $m$ machines, $M_1, \ldots, M_m$, where $m$ is part of the input. A machine can process at most one job at a time and all jobs and machines are available at time 0. If a job $J_j$ is processed on a machine $M_i$, it will take a positive integral processing time $p_{ij}$. Furthermore, for each job we are given a non-negative integral weight $w_j$. The objective is to schedule the jobs so that the sum of the weighted completion times of the jobs is minimized. Graham et al. [4] denote this problem by $R||\sum w_j C_j$. Bruno, Coffman Jr., and Sethi [1] and Lenstra, Rinnooy Kan, and Brucker [10] showed that the problem of minimizing total weighted completion time on two identical parallel machines is
NP-hard; $R|| \sum w_j C_j$ is NP-hard in the strong sense. In the case that the jobs have the same weight, the problem can be formulated as a bipartite matching problem, which can be solved in polynomial time [8, 1]. In case there is only one machine the problem is also easy: sequence the jobs in order of non-increasing ratios $\frac{w_j}{p_{ij}}$ [19]. Hence, the problem reduces to assigning the jobs appropriately to the machines; once the jobs have been assigned to the machines, we sequence the jobs on each machine by the ratio rule.

The first polynomial-time approximation algorithm for minimizing the total weighted completion time on unrelated parallel machines was given by Phillips, Stein, and Wein [12], who achieved a performance ratio $O(\log^2 n)$. Hall, Schulz, Shmoys, and Wein [5] presented a polynomial-time $\frac{16}{3}$-approximation algorithm that uses an LP-relaxation in time-indexed variables and relies on the rounding technique of Shmoys and Tardos for the generalized assignment problem [16]. This result was successively improved by Schulz and Skutella to performance guarantees $2 + \epsilon$ and $\frac{3}{2} + \epsilon$, where $\epsilon$ is an arbitrary positive value [15, 14]. They also used LP-relaxations in time-indexed variables. Skutella presented a polynomial-time $\frac{3}{2}$-approximation algorithm that was based on a convex quadratic relaxation [17]. On the negative side, Hoogeveen, Schuurman, and Woeginger [7] showed that the problem is APX-hard, i.e., there exists an $\epsilon > 0$ such that there does not exist a polynomial-time $(1 + \epsilon)$-approximation for this problem, unless P=NP.

Little work has been done on experimental comparison of these kinds of approximation algorithms and local search heuristics. Previous work of this sort includes the following. Johnson and McGeoch [9] made a case study in local optimization for the traveling salesman problem, in which they also experimentally analyzed several constructive heuristics with a worst-case or probabilistic performance guarantee. In the area of scheduling Hariri and Potts examined the empirical behavior of the earliest completion time (ECT) heuristic and several two-phase heuristics for the problem of minimizing makespan on unrelated parallel machines, $R||C_{max}$ [6]. In the first phase of the two-phase algorithms, an LP-relaxation is solved to generate a partial schedule; the second phase consists of an exact or heuristic method to schedule the remaining jobs. The performance guarantees of the two-phase heuristics they examined are 2 and $(2 + \lceil \log_2 (m - 1) \rceil)$, the performance guarantee of the ECT heuristic is $m$. Their conclusion was that these constructive methods are quite unsatisfactory, as deviations of more than 10% from the optimal solution value are common. They also applied an improvement heuristic to the obtained solutions, which achieved a significant reduction in the makespan at very small computational costs. Glass, Potts, and Shade [3] extended this research by analyzing the empirical performance of other local search heuristics for this problem. Although it is difficult to compare the results of these two papers, as the quality of the solution is measured as the relative deviation from the best obtained solution, which might not be the same in both papers, and the time spent on the local search heuristics is not equivalent to the time spent on the constructive heuristics, the results indicate that genetic descent, simulated annealing, and tabu search outperform the constructive heuristics; the local search heuristics have
a deviation of about 1%. Savelsbergh, Uma, and Wein [13] studied the quality of lower bounds for the problem of minimizing total weighted completion time on a single-machine with release dates for the jobs, obtained by LP-relaxations, and they also studied the quality of upper bounds delivered by a number of approximation algorithms based on these relaxations. The best algorithms come within a few percent of the optimum. Although there are a few exceptions, they also concluded that the higher the quality of the solution to the LP-relaxation, the better the approximation algorithms performs. Uma and Wein [21] made an analytical and empirical comparison of lower bounds for the same problem and also used some rounding techniques to obtain feasible solutions. They also applied simple local search heuristics to the solutions obtained by these approximation algorithms as well as to solutions from scratch. The best results were obtained by applying the local search heuristics to the solutions obtained by good approximation algorithms. Recently Van der Linden [23] made a computational study on the empirical behavior of a convex quadratic approximation algorithm of Skutella [18] for the problem of minimizing the total weighted completion time on unrelated parallel machines with release dates for the jobs. She compared the results to the solutions of an LP-based approximation algorithm by Schulz and Skutella [14]. On all test instances, the LP-relaxation gave better lower bounds than the convex quadratic relaxation. In Section 3, we prove that for the case of trivial release dates the LP-relaxation is guaranteed to give better bounds than the convex quadratic relaxation. Van der Linden also used the convex quadratic relaxation in a Branch and Bound method, which was able to solve problem instances up to 20 jobs and 5 machines.

Our paper is organized as follows. In the following section, we discuss the heuristics for which we have a constant performance guarantee. In Section 3 we discuss the dominance relations of the lower bounds obtained by the relaxations discussed in Section 2, and in Section 4 we describe the local search heuristics. Next we make some remarks on implementation details and we discuss the results of our empirical evaluation. Finally, in Section 6 we make some concluding remarks.

2 Approximation algorithms with a constant guarantee

The heuristics for which we have a constant guarantee are all based on rounding a fractional solution to some relaxation. The rounding of the fractional solutions is done in the same way for all three relaxations. The main idea is to exploit the values of the relaxation as probabilities with which jobs are assigned to machines: each job is assigned to a machine using these probabilities. This way of randomized rounding can be derandomized using the method of conditional expectations, with no difference in performance guarantees, but at the cost of increased, but still polynomial, running times.
2.1 Convex quadratic programming relaxation

The first relaxation we consider is due to Skutella [17]. He introduced a convex quadratic programming relaxation that leads to a polynomial-time $\frac{3}{2}$-approximation algorithm. The basic observation is that the problem is reduced to an assignment problem. Therefore, the problem can be formulated as an integer quadratic program in $nm$ assignment variables, $z_{ij}$. The integer quadratic program, which forms the basis of the relaxation, is given in (IQP), where $k \prec_i j$ means that according to the ratio rule job $J_k$ precedes job $J_j$ on machine $M_i$. Constraints (2) ensure that the completion time of a job is its own processing time plus the processing times of its predecessors on the machine on which it is scheduled and constraints (1) ensure that each job is scheduled on exactly one machine.

$$(IQP)$$

$$\begin{align*}
\min & \sum_j w_j C_j \\
\text{s.t.} & \sum_i z_{ij} = 1 \quad \forall j \quad (1) \\
& C_j = \sum_i z_{ij} (p_{ij} + \sum_{k \prec_i j} p_{ik} z_{ik}) \quad \forall j \quad (2) \\
& z_{ij} \in \{0, 1\} \quad \forall i, j \quad (3)
\end{align*}$$

In (7), the quadratic objective function is convexified by carefully raising the diagonal entries of the matrix determining the quadratic term until it becomes positive semidefinite and the function becomes convex. Adding constraint (6), which ensures that the sum of weighted completion times is at least the sum of weighted processing times, results in the convex quadratic programming problem given in (CQP). Solving this problem and then applying the randomized rounding procedure is a polynomial-time $\frac{3}{2}$-approximation algorithm.

$$(CQP)$$

$$\begin{align*}
\min & \sum_i z_{ij} = 1 \quad \forall j \quad (4) \\
\text{s.t.} & \sum_i z_{ij} = 1 \quad (5) \\
& Z \geq \sum_{i,j} w_{ij} p_{ij} z_{ij} \quad (6) \\
& Z \geq \sum_{i,j} z_{ij} (\frac{1}{2} w_{ij} p_{ij} + \frac{1}{2} w_{ij} p_{ij} z_{ij} + \sum_{k \prec_i j} w_{ij} p_{ik} z_{ik}) \quad (7) \\
& z_{ij} \geq 0 \quad \forall i, j \quad (8)
\end{align*}$$

2.2 Time-indexed variables on processing intervals

Schulz and Skutella [15, 14] generalized an LP-relaxation in time-indexed variables that was introduced by Dyer and Wolsey [2] for the single-machine scheduling problem with release dates. It contains decision variables, $y_{ijt}$, indicating
whether job $J_j$ is being processed on machine $M_i$ during the time interval \([t, t + 1)\), for integral $t$, where $t$ ranges from 0 to $T - 1$, with $T$ an upper bound on the length of a schedule. The resulting LP-relaxation is a $\frac{3}{2}$-relaxation of the scheduling problem under consideration, i.e., the value of an optimal schedule is within a factor $\frac{2}{3}$ of the optimum LP value. Moreover, solving this LP-relaxation and applying the randomized rounding procedure yields a solution with expected value no more than $\frac{3}{2}$ times the optimal value.

The LP-relaxation Schulz and Skutella used is the following:

\[
\begin{align*}
\text{(LPY)} \\
\min & \sum_j w_j C_j \\
\text{s.t.} & \sum_{i, t} y_{ijt} = 1 & \forall j \\
& \sum_j y_{ijt} \leq 1 & \forall i, t \\
& C_j \geq \sum_{i, t} \left( \frac{y_{ijt}}{p_{ij}} (t + \frac{1}{2}) + \frac{1}{2} y_{ijt} \right) & \forall j \\
& C_j \geq \sum_i y_{ijt} & \forall j \\
& y_{ijt} \geq 0 & \forall i, j, t
\end{align*}
\]

To see that this is indeed a relaxation of $R||\sum w_j C_j$, consider an arbitrary feasible schedule, where job $J_j$ is being continuously processed between time $S_j$ and $S_j + p_{hj}$ on machine $M_h$. Then $y_{ijt} = 1$ if $i = h$ and $t \in \{S_j, \ldots, S_j + p_{hj} - 1\}$ and $y_{ijt} = 0$ otherwise. The right hand side in (12) corresponds for these values of $y_{ijt}$ to the completion time of $J_j$ in the schedule. This relaxation can be strengthened by adding the constraint that a job can be processed by at most one machine during each time interval:

\[
\sum_i y_{ijt} \leq 1 & \forall j, t. \\
\]

As the time horizon, $T$, can be exponential in the input size, this relaxation may suffer from an exponential number of variables and constraints. One can overcome this drawback by turning to interval-indexed variables. The time intervals Schulz and Skutella used are of the form $I_0 = [0, 1)$ and $I_l = [(1 + \epsilon)^{l-1}, (1 + \epsilon)^l)$ for $l = 1, \ldots, \frac{\log T}{\log (1 + \epsilon)}$, where $\epsilon > 0$ can be chosen arbitrarily small. The LP-relaxation, including constraints (15), is then the following:

\[
\begin{align*}
\text{(LPY')} \\
\min & \sum_j w_j C_j \\
\end{align*}
\]
\[ \sum_{i,t} \frac{y_{ijt}|I_i|}{p_{ij}} = 1 \quad \forall j \]  
\[ \sum_{j} y_{ijt} \leq 1 \quad \forall i, l \]  
\[ \sum_{i} y_{ijt} \leq 1 \quad \forall j, l \]  
\[ C_j \geq \sum_{i} \left( \left( \frac{1}{2p_{ij}} + \frac{1}{2} \right)y_{ij0} + \sum_{l \geq 1} \left( \frac{y_{ijt}|I_i|}{p_{ij}} (1 + \epsilon)^{l-1} + \frac{1}{2}y_{ijt}|I_i| \right) \right) \quad \forall j \]  
\[ C_j \geq \sum_{i,t} y_{ijt}|I_i| \quad \forall j \]  
\[ y_{ijt} \geq 0 \quad \forall i, j, t \]  

Any feasible solution, \( \tilde{y} \), of \((LPY)\) plus constraints (15) can be transformed into a feasible solution, \( y \), to \((LPY')\) with the same or lower value. This is done by setting \( y_{ijt} = \sum_{t} \frac{|I_i|}{|I_i|} \tilde{y}_{ijt} \). Hence, the optimal value of \((LPY')\) is at most equal to the optimal value of \((LPY)\), including (15).

This leads to a \( (\frac{3}{2} + \epsilon) \)-relaxation of polynomial size and a polynomial-time \( (\frac{3}{2} + \epsilon) \)-approximation algorithm; in the subsequent sections, this method will be referred to as the \( LPY \) approach. Notice that the size of the relaxation still depends substantially on the time horizon and may be huge for small values of \( \epsilon \).

### 2.3 Time-indexed variables on starting times

The LP-relaxation used by Schulz and Skutella yields a poor lower bound, as even a 0 - 1 solution to this relaxation does not necessarily correspond to a feasible schedule. Therefore, we have implemented another relaxation. This is a generalization of a second LP-relaxation introduced by Dyer and Wolsey for the single-machine scheduling problem with release dates. The problem is formulated as an integer program in time-indexed variables, \( x_{ijt} \), denoting whether job \( J_j \) starts being processed on machine \( M_i \) at time \( t \). This program is given in \((IPX)\):

\[ \text{(IPX)} \]
\[ \min \sum_{j} w_j C_j \]  
\[ \text{s.t.} \quad \sum_{i,t} x_{ijt} = 1 \quad \forall j \]  
\[ \sum_{j} \sum_{s=t-p_{ij}+1}^{t} x_{ij}s \leq 1 \quad \forall i, t \]  
\[ C_j = \sum_{i,t} (t + p_{ij})x_{ijt} \]  
\[ x_{ijt} \in \{0,1\} \quad \forall i, j, t \]

The LP-relaxation is obtained by relaxing the integrality constraints (26) to non-negativity constraints and will be denoted by \((LPX)\).
As with LP problem (LPY), the time horizon, \( T \), can be an exponential in the input size. Instead of trying to reduce the size of the LP problem, we solved this large problem by column generation, generalizing the work of Van den Akker, Hurkens, and Savelsbergh [22] on the above mentioned single-machine problem.

To reduce the number of constraints we apply Dantzig-Wolfe decomposition. The constraints (24) and the non-negativity constraints describe a polytope \( P \). This polytope can be written as the Cartesian product \( P = P_1 \times \ldots \times P_m \), where \( P_i = \{ x \in \mathbb{R}^{I_i T} : \sum_j \sum_{s=t-p_{ij}+1}^t x_{jt} \leq 1, \ t = 1, \ldots, T \} \). Hence a point \( x \in P \) can be written as \( x = (x^{(1)}, \ldots, x^{(m)}) \), where \( x^{(i)} \in P_i \) \( (i = 1, \ldots, m) \). The polytopes \( P_i \) have integral vertices [22], and these vertices can be considered as schedules on machine \( M_i \) in which jobs do not have to be processed exactly once, as they do not necessarily satisfy constraints (23). We will call such schedules semi-schedules.

Let \( \xi^{(i)}_1, \ldots, \xi^{(i)}_{K_i} \) be the extreme points of \( P_i \). Then any \( x^{(i)} \in P_i \) can be written as a convex combination \( x^{(i)} = \sum_{l=1}^{K_i} \lambda^{(i)}_l \xi^{(i)}_l \). The LP-relaxation can be reformulated using the variables \( \lambda^{(i)}_l \) as follows:

\[
(LPX') \quad \min \sum_{i} \sum_{l=1}^{K_i} \left( \sum_{j,t} c_{ijt}(x^{(i)}_{jt}) \right) \lambda^{(i)}_l \\
\text{s.t.} \quad \sum_{i} \sum_{l=1}^{K_i} \left( \sum_{t} \xi^{(i)}_{jt} \right) \lambda^{(i)}_l = 1 \quad j = 1, \ldots, n \\
\sum_{i=1}^{K_i} \lambda^{(i)}_l = 1 \quad i = 1, \ldots, m \\
\lambda^{(i)}_l \geq 0
\]

Note that the solutions of \( (LPX) \) and \( (LPX') \) are in a one-to-one correspondence.

Observe that the \( j \)th element of the column corresponding to \( \lambda^{(i)}_l \), i.e., \( \sum_t (\xi^{(i)}_t)_{jt} \), is equal to the number of times that job \( J_j \) occurs in the semi-schedule \( \xi^{(i)}_t \). This means that the column corresponding to the semi-schedule \( \xi^{(i)}_t \) only indicates how many times each job occurs in this schedule. The cost coefficient of \( \lambda^{(i)}_l \) is equal to the cost of the semi-schedule \( \xi^{(i)}_l \).

By reformulating this way, the number of constraints is decreased significantly from \( n + mT \) to \( n + m \). The number of variables, however, has increased to the total number of extreme points of the polytopes \( P_i \). Fortunately, this does not matter, since the problem can be solved through column generation. To apply column generation, we have to find an efficient way to determine a column with minimal reduced cost, i.e., to solve a pricing problem. We determine for each machine such a minimal column in the same way as Van den Akker,
The reduced cost of the variable $\lambda^i_j$ is given by

$$\sum_{j,t} c_{ijt} \xi_{i,j,t}^{(i)} - \sum_j \pi_j \left( \sum_t \xi_{i,j,t}^{(i)} \right) - \alpha_i$$

(30)

where $\pi_j$ denotes the dual variable of constraint (27) for job $J_j$, and $\alpha_i$ denotes the dual variable of constraints (28) for machine $M_i$.

Recall that each extreme point $\xi_{i}^{(i)}$ represents a semi-schedule on machine $M_i$. These semi-schedules can be represented by paths in a network in the following way. The nodes of the network correspond to time points $0, 1, \ldots, T$. For each job $J_j$ and each period $s$, with $s \leq T - p_{ij} + 1$, there is an arc from $s$ to $s + p_{ij}$ that indicates the machine processes job $J_j$ from time $s$ to time $s + p_{ij}$; we say that this arc corresponds to the variable $(x(i))_{js}$. Furthermore, for each time point $t$ there is an idle time arc from $t$ to $t + 1$ that indicates that the machine is idle in period $[t, t + 1)$. Any directed path from 0 to $T$ corresponds to a semi-schedule $\xi_{i}^{(i)}$ on machine $M_i$, and vice versa.

If we set the length of the arc corresponding to $(x(i))_{jt}$ equal to $c_{ijt} - \pi_j$, for all $j$ and $t$, and the length of all idle time arcs equal to 0, then the reduced cost of the variable $\lambda_{i}^{(i)}$ is equal to the length of path corresponding to $\xi_{i}^{(i)}$ minus the dual variable $\alpha_i$. Therefore, finding a column with minimal reduced costs boils down to finding the shortest path in a directed acyclic network with arbitrary weights. As the network is directed and acyclic, the shortest path problem can be solved in $O(nT)$ time.

As we do not know of any analytical bounds on the number of columns that have to be generated and $T$ might not be a polynomial in the input size, we have no polynomial-time guarantee for solving the LP problem.

As a corollary to the work of Schulz and Skutella, solving the LP problem and then applying the randomized rounding technique yields a performance guarantee of $\frac{3}{2}$. This method will be called the LPX approach.

3 Dominance relations among lower bounds

The LP-relaxations and the CQP-relaxation described in the previous section provide us with lower bounds. In the following theorems, we discuss the dominance relations between the lower bounds obtained by $(LPX)$, $(LPY)$, and $(CQP)$.

**Theorem 1** Let $Z_{LPX}$ be the value of an optimal solution to $(LPX)$ and $(LPX')$ and let $Z_{CQP}$ denote the value of an optimal solution to $(CQP)$. Then $Z_{LPX} \geq Z_{CQP}$.

**Proof.** Consider a feasible solution $\lambda = (\lambda^i_j)$ for $(LPX')$, and define, for $i = 1, \ldots, m$, $z_i \in \mathbb{R}^n$ as

$$z_i = \sum_t \lambda_{i}^{(i)}$$
where \( n_{i}^{(i)} \) is the vector consisting of the first \( n \) elements of the column in \((LPX')\) corresponding to variable \( \lambda_{i}^{(i)} \), i.e., the \( j \)th element of \( n_{i}^{(i)} \) is the number of copies of \( J_{j} \) occurring in semi-schedule \( \xi_{i}^{(i)} \).

Let \( c_{ij} \in \mathbb{R}^{n} \) be given by \( c_{ij} = w_{ij}p_{ij} \) and \( D_{i} = (d_{jk}^{(i)})_{jk} \in \mathbb{R}^{n \times n} \) be defined as \( d_{jk}^{(i)} = \min(w_{j} p_{ik}, w_{k} p_{ij}) \). If we define \( Z_{i}^{C} \) as

\[
Z_{i}^{C}(z) = \frac{1}{2} c_{i}^{T} z_{i} + \frac{1}{2} z_{i}^{T} D_{i} z_{i},
\]

and \( Z_{CQP}(z) \) as

\[
Z_{CQP}(z) = \max(\sum_{i} c_{i}^{T} z_{i}, \sum_{i} Z_{i}^{C}),
\]

then \( (z, Z) \in \mathbb{R}^{nm} \times \mathbb{R} \), with \( z = (z_{1}, \ldots, z_{m}) \) and \( Z \geq Z_{CQP}(z) \), is a feasible solution for \((CQP)\), as \( z_{ij} \geq 0 \) and \( \sum_{i} z_{ij} = \sum_{i} \lambda_{i} n_{i}^{(i)} = 1 \), for all \( j \).

The sum of the completion times of all copies of \( J_{j} \) in semi-schedule \( \xi_{i}^{(i)} \) is

\[
C_{i}^{(i)} = \frac{1}{2} p_{ij} n_{ij}^{(i)} + \frac{1}{2} p_{ij} (n_{ij}^{(i)})^{2} + \sum_{k < i} p_{ik} n_{ik}^{(i)} n_{ij}^{(i)}.
\]

Thus the weighted sum of completion times for a machine \( M_{i} \) in \((LPX')\) is

\[
Z_{i}^{X}(\lambda) = \sum_{j,l} w_{j} \lambda_{l}^{(i)} C_{ij}^{(i)} = \sum_{l} \lambda_{l}^{(i)} \left( \frac{1}{2} c_{i}^{T} n_{i}^{(i)} + \frac{1}{2} (n_{i}^{(i)})^{T} D_{i} n_{i}^{(i)} \right).
\]

As all elements of \( n_{i}^{(i)} \) are integer, for all \( l \) and \( i \), we know that

\[
Z_{i}^{X}(\lambda) = \sum_{j,l} w_{j} \lambda_{l}^{(i)} \left( \frac{1}{2} p_{ij} n_{ij}^{(i)} + \frac{1}{2} p_{ij} (n_{ij}^{(i)})^{2} + \sum_{k < i} p_{ik} n_{ik}^{(i)} n_{ij}^{(i)} \right)
\]

\[
\geq \sum_{j,l} w_{j} \lambda_{l}^{(i)} \left( \frac{1}{2} p_{ij} n_{ij}^{(i)} + \frac{1}{2} p_{ij} (n_{ij}^{(i)})^{2} \right)
\]

\[
= c_{i}^{T} (\sum_{l} \lambda_{l}^{(i)} n_{l}^{(i)}) = c_{i}^{T} z_{i}.
\]

Skutella [17] proved that the matrix \( D_{i} \) is positive semi-definite, thus the function \( f_{i}(x) = \frac{1}{2} c_{i}^{T} x + \frac{1}{2} x^{T} D_{i} x \) is convex. By constraint \((28)\) \( \sum_{l} \lambda_{l}^{(i)} = 1 \) and thus

\[
Z_{i}^{X}(\lambda) = \sum_{l} \lambda_{l}^{(i)} f_{i}(n_{l}^{(i)}) \geq f_{i}(\sum_{l} \lambda_{l} n_{l}^{(i)}) = Z_{i}^{C}(z).
\]

By \((31)\) we know that \( \sum_{i} Z_{i}^{X}(\lambda) \geq \sum_{i} c_{i}^{T} z_{i} \) and by \((32)\) we know that \( \sum_{i} Z_{i}^{X}(\lambda) \geq Z_{i}^{C}(z) \). Hence,

\[
Z_{LPX}(\lambda) = \max(\sum_{i} c_{i}^{T} z_{i}, \sum_{i} Z_{i}^{C}(z)) = Z_{CQP}(z).
\]

Thus, any feasible solution for \((LPX')\) can be converted to a feasible solution for \((CQP)\) with value at most equal to the value of the solution for \((LPX')\). Hence, \( Z_{LPX} \geq Z_{CQP} \). □
**Theorem 2** Let $Z_{LPX}$ be the value of an optimal solution to $(LPX)$ and $(LPX')$ and let $Z_{LPY}$ denote the optimal value to $(LPY)$ plus constraints (15). Then $Z_{LPX} \geq Z_{LPY}$.

**PROOF.** Consider a feasible solution for $(LPX)$ and construct a feasible solution, $(y, C^Y) \in \mathbb{R}^{nmT} \times \mathbb{R}^n$, for $(LPY)$ as follows:

$$y_{ijt} = \sum_{t'=t-p_{ij}+1}^t x_{ijt'}, \quad (33)$$

$$C^Y_j = \max \left( \sum_i \sum_t y_{ijt}, \sum_{i,t} \frac{y_{ijt}}{p_{ij}} \left( t + \frac{1}{2} \right) + \frac{1}{2} y_{ijt} \right). \quad (34)$$

It is easy to verify that $y$ satisfies constraints (10), (11), and (15). The right hand side of constraint (13) is

$$\sum_{i,t} y_{ijt} = \sum_{i,t} \sum_{t'=t-p_{ij}+1}^t x_{ijt'} = \sum_{i,t} p_{ij} x_{ijt} \leq \sum_{i,t} (t + p_{ij}) x_{ijt},$$

and the right hand side of constraint (12) is

$$\sum_{i,t} \left( \frac{y_{ijt}}{p_{ij}} \left( t + \frac{1}{2} \right) + \frac{1}{2} y_{ijt} \right) = \sum_{i,t} \sum_{t'=t-p_{ij}+1}^t x_{ijt'} \left( \frac{t + \frac{1}{2}}{p_{ij}} + \frac{1}{2} \right) = \sum_{i,t} \sum_{t'=t+1}^{t+p_{ij}-1} \left( \frac{t + \frac{1}{2}}{p_{ij}} + \frac{1}{2} \right) = \sum_{i,t} \sum_{t'=t+1}^{t+p_{ij}} \left( \frac{t + \frac{1}{2}}{p_{ij}} + \frac{1}{2} \right) = \sum_{i,t} \sum_{t'=t+p_{ij}}^{t} \left( \frac{t + \frac{1}{2}}{p_{ij}} + \frac{1}{2} \right) = \sum_{i,t} (t + p_{ij}) x_{ijt}.$$

Hence, $C^Y_j = \max(\sum_{i,t} y_{ijt}, \sum_{i,t} \frac{y_{ijt}}{p_{ij}} \left( t + \frac{1}{2} \right) + \frac{1}{2} y_{ijt}) \leq \sum_{i,t} (t + p_{ij}) x_{ijt}$ and the optimal value to $(LPX)$ is at least as large as the optimal value to $(LPY)$. \qed

To establish the relation between $(LPY)$ and $(CQP)$, we need the following lemma, that specifies an optimal solution to $(LPY)$ without constraints (13) for a given fractional assignment of the jobs to the machines.

**Lemma 1** Let $z = (z_{ij}) \in \mathbb{R}^{nmT}$ and let $\bar{y} \in \mathbb{R}^{nmT}$ be

$$\bar{y}_{ijt} = \left[ t, t+1 \right] \cap [S_{ij}, S_{ij} + p_{ij} z_{ij}],$$

where $S_{ij} = \sum_{k \prec i,j} p_{ik} z_{ik}$. Then $\bar{y}$ minimizes

$$f(y) = \sum_j w_j \left( \sum_{i,t} \frac{y_{ijt}}{p_{ij}} \left( t + \frac{1}{2} \right) + \frac{1}{2} y_{ijt} \right)$$

over all $y \in Y(z) = \{ y \in \mathbb{R}^{nmT} : \sum_j y_{ijt} \leq 1, \sum_t \frac{y_{ijt}}{p_{ij}} = z_{ij} \}$. 

10
PROOF. First note that, by construction, \( \tilde{y} \) satisfies \( \sum_j \tilde{y}_{ijt} \leq 1 \) for all \( i, t \), and \( \sum_t \frac{y_{ijt}}{p_{ij}} = z_{ij} \), and so \( \tilde{y} \in Y(z) \). In addition \( \tilde{y} \) has the property that for all \( i \) there exist an \( s_i \) and an \( \alpha_i \in [0,1) \) such that

\[
\sum_j \tilde{y}_{ijt} = \begin{cases} 
1 & \text{if } t < s_i, \\
\alpha_i & \text{if } t = s_i, \\
0 & \text{if } t > s_i. 
\end{cases}
\]

Consider a solution \( y \in Y(z) \), such that \( y \neq \tilde{y} \). Then there exists a triple \((i, j, t)\) such that \( y_{ijt} < \tilde{y}_{ijt} \). Let \((i, k, t')\) be such a triple, with minimal \( t' \). Note that by construction of \( \tilde{y} \) and minimality of \( t' \), \( y_{ijt} = \tilde{y}_{ijt} \) for all \( t < t' \), and thus there exists a \( t'' > t' \) such that \( y_{ikt''} > \tilde{y}_{ikt''} \).

If \( \sum_j y_{ijt'} < 1 \), then we construct \( y' \) by adding \( \beta = \min(1 - \sum_j y_{ijt'}, y_{ikt''}) \) to \( y_{ikt''} \) and subtracting the same value from \( y_{ikt''} \). Clearly, \( y' \in Y(z) \), and

\[
f(y) - f(y') = \frac{w_k}{p_{ik}} \beta (t'' - t') > 0.
\]

If \( \sum_j y_{ijt'} = 1 \), then there is an \( l \) such that \( y_{ilt'} > \tilde{y}_{ilt'} \geq 0 \). By construction of \( \tilde{y} \) we have \( k < i \). Let \( \beta = \min(y_{ilt'}, y_{ikt''}, \tilde{y}_{ikt''} - y_{ikt'}) \) and construct \( y' \) by adding \( \beta \) to \( y_{ikt'} \) and \( y_{ilt'} \) and subtracting it from \( y_{ikt''} \) and \( y_{ilt'} \). Then, clearly \( y' \in Y(z) \) and

\[
f(y) - f(y') = \frac{w_k}{p_{ik}} \beta (t'' - t') + \frac{w_i}{p_{il}} \beta (t' - t'') \geq 0.
\]

Hence, we have constructed a solution that has the same or lower value and is closer to \( \tilde{y} \). Setting \( y = y' \) and repeating this procedure results in the solution \( \tilde{y} \), and it has the same or lower value than all intermediate solutions. That is, \( f(\tilde{y}) \leq f(y) \) for all \( y \in Y(z) \). \( \square \)

**Theorem 3** Let \( Z_{LPY} \) be the value of an optimal solution to \((LPY)\) and let \( Z_{CQP} \) denote the optimal solution value to \((CQP)\). Then \( Z_{LPY} \geq Z_{CQP} \).

**Proof.** Let \((y, C^Y) \in \mathbb{R}^{nm} \times \mathbb{R}^n \) be a feasible solution for \((LPY)\) and let \( z \in \mathbb{R}^{nm} \) be defined as \( z_{ij} = \sum_j \frac{y_{ijt}}{p_{ij}} \), and \( Z_{CQP}(z) \) as

\[
Z_{CQP}(z) = \max(\sum_{i,j} w_j p_{ij} z_{ij}, \sum_{i,j} \frac{1}{2} w_j p_{ij} z_{ij}^2 + \frac{1}{2} w_j p_{ij} z_{ij}^2 + \sum_{k<i,j} p_{ik} z_{ij} z_{ik}).
\]

Then \((z, Z_{CQP}(z))\) is a feasible solution for \((CQP)\), and

\[
\sum_{i,j} w_j p_{ij} z_{ij} = \sum_j w_j \sum_{i,t} y_{ijt} \leq \sum_j w_j C^Y_j.
\]

Below, we prove that

\[
\sum_{i,j} \frac{1}{2} w_j p_{ij} z_{ij}^2 + \frac{1}{2} w_j p_{ij} z_{ij}^2 + \sum_{k<i,j} p_{ik} z_{ij} z_{ik} \leq \sum_j w_j C^Y_j(y).
\]
Let $\tilde{y}$ be the feasible solution for (LPY) as defined in Lemma 1, i.e.,

$$\tilde{y}_{ijt} = \left| [(t, t+1) \cap [S_{ij}, S_{ij} + p_{ij}z_{ij}]] \right|,$$

where $S_{ij} = \sum_{k<i} p_{ik}z_{ik}$. Let $\alpha_{ij} = S_{ij} - [S_{ij}]$ and $\beta_{ij} = [S_{ij} + p_{ij}z_{ij}] - (S_{ij} + p_{ij}z_{ij})$. Then

$$\tilde{y}_{ijt} = \begin{cases} 
1 - \alpha_{ij} & \text{if } t = [S_{ij}] \text{ and } [S_{ij}] < [S_{ij} + p_{ij}z_{ij}] - 1, \\
1 - \beta_{ij} & \text{if } t = [S_{ij} + p_{ij}z_{ij}] - 1 \text{ and } [S_{ij}] < [S_{ij} + p_{ij}z_{ij}] - 1, \\
1 - \alpha_{ij} - \beta_{ij} & \text{if } t = [S_{ij}] = [S_{ij} + p_{ij}z_{ij}] - 1, \\
1 & \text{if } [S_{ij}] < t < [S_{ij} + p_{ij}z_{ij}] - 1, \\
0 & \text{otherwise.}
\end{cases}$$

Let $C^y_{ij}(\tilde{y})$ be defined as

$$C^y_{ij}(\tilde{y}) = \sum_t \left( \frac{\tilde{y}_{ijt}}{p_{ij}} (t + \frac{1}{2}) + \frac{1}{2} \tilde{y}_{ijt} \right).$$

If $[S_{ij}] = [S_{ij} + p_{ij}z_{ij}] - 1$, then $1 - \alpha_{ij} - \beta_{ij} = p_{ij}z_{ij}$ and

$$C^y_{ij}(\tilde{y}) = \frac{1}{p_{ij}} \left( S_{ij} - \alpha_{ij} + \frac{1}{2} \right) + \frac{1}{2} p_{ij}z_{ij}$$

$$= \frac{1}{2} p_{ij}z_{ij} + z_{ij} \left( S_{ij} + \frac{1}{2} p_{ij}z_{ij} \right) + \frac{1}{p_{ij}} \left( 1 - \alpha_{ij} - \beta_{ij} \right) \left( -\frac{1}{2} (1 - \alpha_{ij} - \beta_{ij}) - \alpha_{ij} + \frac{1}{2} \right)$$

$$= \frac{1}{2} p_{ij}z_{ij} + \frac{1}{2} p_{ij}z_{ij}^2 + z_{ij} S_{ij} + \frac{1}{p_{ij}^2} \left( \alpha_{ij} - \beta_{ij} \right) \left( \alpha_{ij} + \beta_{ij} - 1 \right). \quad (35)$$

If $[S_{ij}] < [S_{ij} + p_{ij}z_{ij}] - 1$, then

$$C^y_{ij}(\tilde{y}) = \frac{1}{p_{ij}} \left( S_{ij} - \alpha_{ij} + \frac{1}{2} \right) + \frac{1}{p_{ij}} \left( S_{ij} + p_{ij}z_{ij} + \beta_{ij} - \frac{1}{2} \right) + \frac{1}{p_{ij}} (p_{ij}z_{ij} - 2 + \alpha_{ij} + \beta_{ij}) (S_{ij} + \frac{1}{2} p_{ij}z_{ij} - \frac{1}{2} (\alpha_{ij} - \beta_{ij})) + \frac{1}{2} p_{ij}z_{ij}$$

$$= \frac{1}{2} p_{ij}z_{ij} + \frac{1}{2} p_{ij}z_{ij}^2 + \frac{1}{p_{ij}^2} \left( \alpha_{ij} - \beta_{ij} \right) \left( \alpha_{ij} + \beta_{ij} - 1 \right). \quad (36)$$

Thus in both cases

$$C^y_{ij}(\tilde{y}) = \frac{1}{2} p_{ij}z_{ij} + \frac{1}{2} p_{ij}z_{ij}^2 + z_{ij} S_{ij} + \frac{1}{p_{ij}^2} \left( \alpha_{ij} - \beta_{ij} \right) \left( \alpha_{ij} + \beta_{ij} - 1 \right).$$

Consider a machine $M_t$ and assume w.l.o.g. that $\{j : p_{ij}z_{ij} > 0\} = \{1, \ldots, K\}$, and that $J_1 <_i \ldots <_i J_K$. Then $\alpha_{ii} = 0$ and, as $S_{i,j+1} = S_{ij} + p_{ij}z_{ij}$, $\alpha_{i,j+1} = 1 - \beta_{ij}$, for $j = 1, \ldots, K - 1$.  

12
Then

\[ \sum_j w_j C_{ij}^Y(y) = \sum_j \left( \frac{1}{2} w_j p_{ij} z_{ij} + \frac{1}{2} w_j p_{ij} z_{ij}^2 + \sum_{k<i,j} w_j p_{ik} z_{ij} z_{ik} \right) + \right. 
\left. + \sum_j \frac{w_j}{p_{ij}} \left( \alpha_{ij} - \beta_{ij} \right) \left( \alpha_{ij} + \beta_{ij} - 1 \right) \right.
\]
\[ \left. = \sum_j \left( \frac{1}{2} w_j p_{ij} z_{ij} + \frac{1}{2} w_j p_{ij} z_{ij}^2 + \sum_{k<i,j} w_j p_{ik} z_{ij} z_{ik} \right) + \right. 
\left. + \sum_{j=1}^{K-1} \frac{1}{2} \left( \beta_{ij} - \beta_{ij}^2 \right) \left( \frac{w_j}{p_{ij}} - \frac{w_{j+1}}{p_{i,j+1}} \right) + \frac{1}{2} \left( \beta_{iK} - \beta_{iK}^2 \right) \frac{w_K}{p_{iK}} \right. 
\]
\[ \left. \geq \sum_j \left( \frac{1}{2} w_j p_{ij} z_{ij} + \frac{1}{2} w_j p_{ij} z_{ij}^2 + \sum_{k<i,j} w_j p_{ik} z_{ij} z_{ik} \right) + \right. 
\left. + \sum_{j=1}^{K-1} \frac{1}{2} \left( \beta_{ij} - \beta_{ij}^2 \right) \left( \frac{w_j}{p_{ij}} - \frac{w_{j+1}}{p_{i,j+1}} \right) + \frac{1}{2} \left( \beta_{iK} - \beta_{iK}^2 \right) \frac{w_K}{p_{iK}} \right. 
\]
\[ \sum_j \left( \frac{1}{2} w_j p_{ij} z_{ij} + \frac{1}{2} w_j p_{ij} z_{ij}^2 + \sum_{k<i,j} w_j p_{ik} z_{ij} z_{ik} \right) \leq \sum_{i,j} w_j C_{ij}^Y(y) \leq \sum_{i,j} w_j C_{ij}^Y(y). \] (37)

The last inequality is due to Lemma 1. And thus $Z_{CQP}(z) \leq Z_{LPY}(y)$. \(\square\)

(LPY) plus constraints (15) yields a higher lower bound than (LPY). However, as (LPY') is a underestimate of (LPY) plus constraints (15), it does not need to be the case that $Z_{LPY'} \geq Z_{CQP}$, where $Z_{LPY'}$ is the optimal solution to (LPY').

The inequalities in the above theorems are strict, as is shown in the following example.

**Example 1** Consider an instance with three jobs and two machines. Job $J_1$ can only be processed on machine $M_1$ and has processing time $p_{11} = 3$ and weight $w_1 = 11$. Job $J_2$ can only be processed on machine $M_2$ and has weight $w_2 = 1$ and processing time $p_{22} = 1$. Job $J_3$ can be processed on both machines. Its processing times are $p_{13} = 2$ and $p_{23} = 6$ and its weight is $w_3 = 7$.

The optimal solution to (LPX) is the optimal schedule: $J_1$ and $J_3$ are processed by $M_1$ and $J_2$ is scheduled on machine $M_2$. The optimal value to (LPX) is $Z_{LPX} = 69$. The optimal solution to (LPY) with or without constraints (15) has value $Z_{LPY} = 67\frac{1}{2}$ and in this solution $J_1$ is assigned to the time slots $[0, 1], [1, 2]$, and $[2, 3]$ on $M_1$, $J_2$ is assigned to the time slot $[2, 3]$ on $M_2$, and $J_3$ is fully assigned to the time slots $[3, 4]$ on $M_1$ and $[0, 1]$ and $[1, 2]$ on $M_2$ and for one third to time slot $[4, 5]$ on $M_1$. Finally, the optimal solution to (CQP) assigns $J_3$ for $\frac{11}{50}$ to $M_1$ and for $\frac{18}{50}$ to $M_2$. The optimal value to (CQP) is $Z_{CQP} = \frac{7563}{112} = 66.991$. \(\square\)
4 Local search

We compare the algorithms described in the previous section to local search heuristics. Local search is a family of methods that iteratively search through the set of feasible solutions. Starting from an initial solution, a local search procedure moves from one 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 methods. Recall that the problem can be reduced to the problem of assigning the jobs appropriately to the machines. Hence, we represent a schedule by the assignment of the jobs to the machines.

We consider two types of neighborhood functions. First, for the jump neighborhood, we select a job $J_j$ and a machine $M_i$ such that $J_j$ is not scheduled on machine $M_i$. A neighbor is formed by moving job $J_j$ to machine $M_i$ (see Figure 1).

The second neighborhood function is called swap. For this neighborhood, we select two jobs $J_j$ and $J_k$, assigned to different machines, and the neighbor is obtained by interchanging their machine allocation (see Figure 2).

Besides applying the local search methods to the solutions obtained by the heuristics described in the previous section, they are also applied to randomly generated initial solutions. We have two strategies to generate these random solutions. The first strategy is the completely random strategy, in which we assign each job independently and uniformly to one of the $m$ machines. In the second strategy, called random greedy, the jobs are, in random order, greedily assigned to the machines, that is, given a partial schedule and the first job that still has to be scheduled, the job is assigned to the machine for which the total weighted completion time of the new partial schedule is minimal.

We have implemented two local search heuristics: multi-start iterative improvement and tabu search.
4.1 Multi-start iterative improvement

Iterative improvement is a simple form of local search. A neighbor is only accepted if this solution has lower cost. If no neighbor has lower cost, a local minimum has been found and the procedure terminates.

There are several ways to select the neighbor to move to. First there is first improvement: we move to the first neighbor encountered that has lower cost. Another strategy for selecting the neighbor is best improvement: move to the neighbor that has lowest cost among all neighbors. Initial tests showed that the neighbor selecting methods perform equally well.

Multi-start iterative improvement refers to a repeated application of the iterative improvement procedure on multiple initial solutions. Note that we only apply multi-start iterative improvement on randomly generated starting solutions. To make a fair comparison with the other methods, the number of repetitions is chosen such that the total time spent on this procedure is about the same as that for the others.

4.2 Tabu search

In tabu search, moves to non-improving neighboring solutions are also allowed, so that we can get out of local minima. To avoid cycling, that is, returning to the same solutions, some information about solutions visited in past iterations are stored in a so-called tabu list. If a neighbor satisfies the properties of an entry in the tabu list, this solution is tabu and it may not be visited, unless it has value that is lower than the best found solution so far or satisfies some other aspiration criteria. The tabu list contains a limited number of entries, and thus a limited number of forbidden properties.

The information stored in the tabu list due to a jump move is the job that has changed its machine allocation. If the move is due to a swap, we look at the contribution of the two swapped jobs to the objective function before and after the swap. The swapped job that has lowest decrease or highest increase in its contribution is stored in the tabu list. Hence, after each move, jump or swap, we store one job in the tabu list. This job has to remain on its new machine for a number of iterations, unless moving it yields a better solution than found so far. The number of iterations during which a job has a fixed machine allocation is equal to the length of the tabu list, which, after some initial experiments, we have chosen to be $n/2$ whenever $n < 40$ and 20 otherwise.

In our tabu search procedure, we use best improvement for finding the neighbor to move to. However, if a move is made to a non-improving neighbor, we only allow jumps. The reason for this is that a jump creates more space on the machine from which the job is leaving than a swap and thus allows for better improvements in the subsequent iterations. Initial tests show indeed that tabu search with this feature gives better results than tabu search where non-improving swaps are also accepted.

Another feature of our tabu search procedure is the backjump, cf. the tabu search procedure of Nowicki and Smutnicki for the job-shop problem [11]: if we have made 500 non-improving moves without improving the best found solution,
we return to this solution and move to a neighbor that has not been visited yet directly from this solution.

In the case that tabu search is applied to the solutions obtained by the heuristics with constant performance guarantee, the procedure terminates when there have been too many, that is, 20, backjumps to the same solution. In the case of randomly generated start solutions, we repeat the procedure of applying tabu search to a start solution until the total time spent is about the same as for the other procedures.

5 Computational experience

5.1 Test instances

Our heuristics have been tested on instances with size varying from 10 jobs and 5 machines to 100 jobs and 50 machines. The solution quality may depend on the structure of the test instances. To allow for possible variations in performance, three classes of test instances were considered, each of which is based on a different method of generating the processing time \( p_{ij} \) of job \( J_j \) on machine \( M_i \).

- **No correlation:** all processing times \( p_{ij} \) are independently drawn from the uniform distribution over \([10, 100]\); \( w_j \) is an integer from the uniform distribution over \([1, 100]\).

- **Machine correlation:** \( p_{ij} \) is an integer from the uniform distribution over \([\alpha_i, \alpha_i + 10]\) where \( \alpha_i \) is an integer from the uniform distribution over \([1, 100]\); \( w_j \) is an integer from the uniform distribution over \([1, 100]\).

- **Favorite machines:** each job \( J_j \) has two favorite machines, \( M_{i_1(j)} \) and \( M_{i_2(j)} \), which are randomly selected; \( p_{i_1(j),j} \) and \( p_{i_2(j),j} \) are drawn from the uniform distribution over \([\beta_j, \beta_j + 4]\), where \( \beta_j \) is an integer from the uniform distribution over \([15, 25]\), and \( p_{ij} \) (\( i \neq i_1(j), i_2(j) \)) is drawn from the uniform distribution over \([70, 90]\); \( w_j \) is an integer from the uniform distribution over \([1, 100]\).

For each value of \( m \) and \( n \), 50 instances have been generated for each of the three instance classes.

5.2 Implementational details

Multi-start iterative improvement, tabu search and both LP approaches have been implemented in C, using CPLEX for solving the LP problems. The CQP approach has been coded in MATLAB, and we use SEDuM1 [20] for solving the CQP problem. The reason for implementing the CQP approach in MATLAB instead of C is that SEDuM1 is only available in MATLAB. The tests have been run on a Sun Ultra-1, 140 MHz, with 256 MB memory.
Table 1: Lower bounds: average (maximum) relative deviation in % from best lower bound

<table>
<thead>
<tr>
<th>n x m</th>
<th>10 x 10</th>
<th>20 x 20</th>
<th>50 x 50</th>
<th>10 x 5</th>
<th>20 x 10</th>
<th>50 x 20</th>
<th>100 x 50</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQP</td>
<td>-5.13</td>
<td>-5.17</td>
<td>-4.35</td>
<td>-3.69</td>
<td>-3.67</td>
<td>-2.76</td>
<td>-3.35</td>
<td>-4.02</td>
</tr>
<tr>
<td>(</td>
<td>-11.82</td>
<td>-10.23</td>
<td>-9.02</td>
<td>-8.02</td>
<td>-8.46</td>
<td>-4.98</td>
<td>-5.69</td>
<td></td>
</tr>
<tr>
<td>LPY'</td>
<td>-4.48</td>
<td>-4.41</td>
<td>-3.85</td>
<td>-5.01</td>
<td>-5.10</td>
<td>-4.95</td>
<td>-5.00</td>
<td>-4.68</td>
</tr>
</tbody>
</table>

Table 2: Lower bounds: average relative deviation in % from best lower bound - machine correlated instances

<table>
<thead>
<tr>
<th>n x m</th>
<th>10 x 10</th>
<th>20 x 20</th>
<th>50 x 50</th>
<th>10 x 5</th>
<th>20 x 10</th>
<th>50 x 20</th>
<th>100 x 50</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPY'</td>
<td>-7.17</td>
<td>-7.31</td>
<td>-7.12</td>
<td>-6.26</td>
<td>-6.61</td>
<td>-6.25</td>
<td>-6.52</td>
<td>-6.75</td>
</tr>
</tbody>
</table>

5.3 Computational results

Before looking how good the schedules, obtained by the several algorithms, are, we first look at the quality of the lower bounds. In Table 1, we show the average relative deviation of the lower bounds obtained by the CQP and LPY'-relaxation from the best obtained lower bounds, which are all obtained by the LPX-relaxation as shown in Section 3. The average and maximum are taken over all instances.

We see that for the square instances, where the number of jobs is equal to the number of machines, the LPY'-relaxation yields somewhat better lower bounds, whereas the CQP-relaxation yields the better lower bounds for the rectangular instances. The reason can be found in the fact that the schedule length for square instances is on average shorter than for the rectangular instances with the same number of machines and therefore the LPY'-relaxation is closer to the LPY-relaxation. On average, the CQP and LPY'-relaxation are about 4% to 5% from the lower bounds obtained by the LPX-relaxation. For the uncorrelated and favorite machine instances, there is the same behavior, whereas for the machine correlated instances the CQP-relaxation is better for the square as well as the rectangular instances; these results are found in Table 2. In the machine correlated instances all jobs have the same favored machine, which results in a somewhat higher schedule length. Because of this higher schedule length, the LPY'-relaxation has a larger deviation from the LPY-relaxation, resulting in a worse lower bound.

The upper bounds, i.e., values of schedules, obtained by the CQP, LPY, and LPX approach are given in Table 3.

The LPX approach yields the best upper bounds of these three methods, on average around 0.13% from the best lower bound. The reason that the LPX approach yields better solutions is that the solution to the LPX-relaxation is much closer to a real schedule than the solutions to the CQP and LPY'-
Table 3: Heuristics with constant guarantee: average (maximum) relative deviation in % from best lower bound

<table>
<thead>
<tr>
<th>$n \times m$</th>
<th>10 x 10</th>
<th>20 x 20</th>
<th>50 x 50</th>
<th>10 x 5</th>
<th>20 x 10</th>
<th>50 x 20</th>
<th>100 x 50</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CQP</strong></td>
<td>1.20</td>
<td>1.23</td>
<td>1.48</td>
<td>1.02</td>
<td>1.19</td>
<td>1.37</td>
<td>1.24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(9.38)</td>
<td>(6.57)</td>
<td>(3.56)</td>
<td>(5.61)</td>
<td>(3.46)</td>
<td>(3.26)</td>
<td></td>
<td>0.89</td>
</tr>
<tr>
<td><strong>LPY</strong></td>
<td>0.69</td>
<td>0.77</td>
<td>0.81</td>
<td>0.84</td>
<td>0.96</td>
<td>1.05</td>
<td>1.12</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>(7.05)</td>
<td>(2.99)</td>
<td>(2.68)</td>
<td>(4.77)</td>
<td>(4.10)</td>
<td>(3.07)</td>
<td>(3.01)</td>
<td></td>
</tr>
<tr>
<td><strong>LPX</strong></td>
<td>0.07</td>
<td>0.08</td>
<td>0.06</td>
<td>0.15</td>
<td>0.16</td>
<td>0.20</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1.51)</td>
<td>(1.25)</td>
<td>(0.71)</td>
<td>(3.48)</td>
<td>(2.65)</td>
<td>(0.91)</td>
<td>(1.24)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Heuristics with constant guarantee: average relative deviation in % from best lower bound - machine correlated instances

<table>
<thead>
<tr>
<th>$n \times m$</th>
<th>10 x 10</th>
<th>20 x 20</th>
<th>50 x 50</th>
<th>10 x 5</th>
<th>20 x 10</th>
<th>50 x 20</th>
<th>100 x 50</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CQP</strong></td>
<td>1.54</td>
<td>1.56</td>
<td>1.69</td>
<td>1.16</td>
<td>1.63</td>
<td>1.75</td>
<td>1.68</td>
<td>1.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LPY</strong></td>
<td>0.94</td>
<td>1.30</td>
<td>1.43</td>
<td>1.15</td>
<td>1.36</td>
<td>1.63</td>
<td>1.61</td>
<td>1.34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LPX</strong></td>
<td>0.05</td>
<td>0.09</td>
<td>0.11</td>
<td>0.07</td>
<td>0.04</td>
<td>0.06</td>
<td>0.07</td>
<td>0.07</td>
</tr>
</tbody>
</table>

relaxations, that is, the number of jobs $J_j$ for which there is a machine $M_i$ such that $\sum_t x_{ijt} = 1$ is much larger than the number of jobs for which there is a machine $M_i$ such that $\sum_t y_{ijt} = 1$ or $z_{ij} = 1$, respectively.

In Table 4 we report on the average relative deviation from the best lower bound obtained by the CQP, LPY and LPX approach for the machine correlated instances. Although the lower bounds for the LPY'-relaxation are worse than those for the CQP-relaxation, we see that the values of the schedules obtained by the LPY approach are not worse than those obtained by the CQP approach. The reason for this is that the average number of fractional assignments obtained by the CQP-relaxation is not less than the number of fractional assignment obtained by the LPY'-relaxation.

Table 5 shows the results of applying iterative improvement, denoted by II, and tabu search, denoted by TS, to the solutions of the heuristics with a constant guarantee. We see that applying a simple iterative improvement procedure to the CQP and LPY solutions, although improving the solutions significantly, still yields results that are not better than the solution of the LPX approach. Applying tabu search to the solutions of the CQP and LPY approach yields results that are on average as good as the LPX approach; note that the tabu search solutions are all very close to optimal as their values are only about 0.1% away from the best lower bound. Tabu search applied to the LPX solution yields the best results: on average less than 0.1% from the best lower bound.

In Table 6, the results of the local search heuristics are given. We have used two different time bounds as stopping criteria for the local search heuristics. The first one is that the time is equal to the time used by the LPX approach; the second time bound is that the time equal to the time used by the LPX.
Table 5: Heuristics with constant guarantee plus local search: average relative deviation in % from best lower bound

<table>
<thead>
<tr>
<th>$n \times m$</th>
<th>10 $\times$ 10</th>
<th>20 $\times$ 20</th>
<th>50 $\times$ 50</th>
<th>10 $\times$ 5</th>
<th>20 $\times$ 10</th>
<th>50 $\times$ 20</th>
<th>100 $\times$ 50</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQP</td>
<td>1.20</td>
<td>1.23</td>
<td>1.48</td>
<td>1.02</td>
<td>1.19</td>
<td>1.19</td>
<td>1.37</td>
<td>1.24</td>
</tr>
<tr>
<td>CQP + II</td>
<td>0.50</td>
<td>0.45</td>
<td>0.68</td>
<td>0.34</td>
<td>0.40</td>
<td>0.44</td>
<td>0.59</td>
<td>0.48</td>
</tr>
<tr>
<td>CQP + TS</td>
<td>0.06</td>
<td>0.06</td>
<td>0.18</td>
<td>0.12</td>
<td>0.11</td>
<td>0.18</td>
<td>0.27</td>
<td>0.14</td>
</tr>
<tr>
<td>LPY</td>
<td>0.69</td>
<td>0.77</td>
<td>0.81</td>
<td>0.84</td>
<td>0.96</td>
<td>1.05</td>
<td>1.12</td>
<td>0.89</td>
</tr>
<tr>
<td>LPY + II</td>
<td>0.29</td>
<td>0.23</td>
<td>0.36</td>
<td>0.25</td>
<td>0.35</td>
<td>0.40</td>
<td>0.47</td>
<td>0.34</td>
</tr>
<tr>
<td>LPY + TS</td>
<td>0.06</td>
<td>0.07</td>
<td>0.15</td>
<td>0.12</td>
<td>0.11</td>
<td>0.16</td>
<td>0.23</td>
<td>0.13</td>
</tr>
<tr>
<td>LPX</td>
<td>0.07</td>
<td>0.08</td>
<td>0.06</td>
<td>0.15</td>
<td>0.16</td>
<td>0.20</td>
<td>0.17</td>
<td>0.13</td>
</tr>
<tr>
<td>LPX + II</td>
<td>0.06</td>
<td>0.07</td>
<td>0.04</td>
<td>0.13</td>
<td>0.13</td>
<td>0.14</td>
<td>0.11</td>
<td>0.10</td>
</tr>
<tr>
<td>LPX + TS</td>
<td>0.04</td>
<td>0.05</td>
<td>0.03</td>
<td>0.12</td>
<td>0.09</td>
<td>0.11</td>
<td>0.08</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Table 6: LPX versus local search: average relative deviation in % from best lower bound

<table>
<thead>
<tr>
<th>$n \times m$</th>
<th>10 $\times$ 10</th>
<th>20 $\times$ 20</th>
<th>50 $\times$ 50</th>
<th>10 $\times$ 5</th>
<th>20 $\times$ 10</th>
<th>50 $\times$ 20</th>
<th>100 $\times$ 50</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPX</td>
<td>0.07</td>
<td>0.08</td>
<td>0.06</td>
<td>0.15</td>
<td>0.16</td>
<td>0.20</td>
<td>0.17</td>
<td>0.13</td>
</tr>
<tr>
<td>LPX + TS</td>
<td>0.04</td>
<td>0.05</td>
<td>0.03</td>
<td>0.12</td>
<td>0.09</td>
<td>0.11</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>II</td>
<td>0.06</td>
<td>0.07</td>
<td>0.27</td>
<td>0.12</td>
<td>0.09</td>
<td>0.10</td>
<td>0.38</td>
<td>0.16</td>
</tr>
<tr>
<td>II long</td>
<td>0.05</td>
<td>0.05</td>
<td>0.22</td>
<td>0.12</td>
<td>0.09</td>
<td>0.09</td>
<td>0.37</td>
<td>0.14</td>
</tr>
<tr>
<td>TS</td>
<td>0.09</td>
<td>0.30</td>
<td>0.56</td>
<td>0.13</td>
<td>0.19</td>
<td>0.30</td>
<td>0.39</td>
<td>0.28</td>
</tr>
<tr>
<td>TS long</td>
<td>0.08</td>
<td>0.19</td>
<td>0.45</td>
<td>0.12</td>
<td>0.12</td>
<td>0.20</td>
<td>0.35</td>
<td>0.22</td>
</tr>
</tbody>
</table>

approach and applying tabu search to these solutions. The latter will be denoted by the extension "long". We see that tabu search applied to a good starting solution yields better results than applying it to random start solutions and the multi-start iterative improvement procedures. The multi-start iterative improvement procedures perform better than the tabu search applied to the randomly generated start solutions. This implies that it is hard to get out of the local optima and it is better to have many start solutions and do a simple improvement procedure than to do a more sophisticated improvement procedure on few start solutions.

In Table 7, we report on the performance of the several heuristics on the hardest instances for the LPX approach. For each size, we have chosen from all the instances the one for which the LPX approach has the worst performance ratio. We see that for these instances the tabu search procedure applied to the schedule obtained by the LPX approach yields a schedule that is the best found or close to the best found schedule. Moreover, we see that the CQP and LPY approach on these hard instances for the LPX approach do not perform much better, except for the CQP approach on the instance of size 10$\times$10.

Finally, Table 8 reports on the time it takes to find the solutions for the heuristics with guarantees. As the local search procedures that are applied to the randomly generated start solutions take about the same time as the LPX approach, we do not report on the time usage of these heuristics. Applying
Table 7: Hardest instances for LPX approach

<table>
<thead>
<tr>
<th>$n \times m$</th>
<th>10 x 10</th>
<th>20 x 20</th>
<th>50 x 50</th>
<th>10 x 5</th>
<th>20 x 10</th>
<th>50 x 20</th>
<th>100 x 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQP</td>
<td>0.03</td>
<td>0.62</td>
<td>1.51</td>
<td>4.11</td>
<td>2.25</td>
<td>1.52</td>
<td>1.43</td>
</tr>
<tr>
<td>LPY</td>
<td>1.51</td>
<td>0.62</td>
<td>2.21</td>
<td>4.11</td>
<td>1.38</td>
<td>1.49</td>
<td>1.40</td>
</tr>
<tr>
<td>LPX</td>
<td>1.51</td>
<td>1.25</td>
<td>0.71</td>
<td>3.48</td>
<td>2.65</td>
<td>0.91</td>
<td>1.24</td>
</tr>
<tr>
<td>CQP + TS</td>
<td>0.03</td>
<td>0.33</td>
<td>0.34</td>
<td>3.48</td>
<td>1.14</td>
<td>0.46</td>
<td>0.61</td>
</tr>
<tr>
<td>LPY + TS</td>
<td>0.03</td>
<td>0.33</td>
<td>0.64</td>
<td>3.48</td>
<td>1.14</td>
<td>0.46</td>
<td>0.55</td>
</tr>
<tr>
<td>LPX + TS</td>
<td>0.03</td>
<td>0.33</td>
<td>0.26</td>
<td>3.48</td>
<td>1.14</td>
<td>0.52</td>
<td>0.60</td>
</tr>
<tr>
<td>II</td>
<td>0.03</td>
<td>0.33</td>
<td>0.04</td>
<td>3.48</td>
<td>1.19</td>
<td>0.48</td>
<td>0.48</td>
</tr>
<tr>
<td>II long</td>
<td>0.03</td>
<td>0.33</td>
<td>0.03</td>
<td>3.48</td>
<td>1.14</td>
<td>0.48</td>
<td>0.48</td>
</tr>
<tr>
<td>TS</td>
<td>0.77</td>
<td>0.79</td>
<td>0.24</td>
<td>3.48</td>
<td>1.57</td>
<td>0.52</td>
<td>0.57</td>
</tr>
<tr>
<td>TS long</td>
<td>0.77</td>
<td>0.33</td>
<td>0.24</td>
<td>3.48</td>
<td>1.35</td>
<td>0.52</td>
<td>0.57</td>
</tr>
</tbody>
</table>

Table 8: Time usage in seconds: average over all instances

<table>
<thead>
<tr>
<th>$n \times m$</th>
<th>10 x 10</th>
<th>20 x 20</th>
<th>50 x 50</th>
<th>10 x 5</th>
<th>20 x 10</th>
<th>50 x 20</th>
<th>100 x 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQP</td>
<td>1.66</td>
<td>6.72</td>
<td>153.92</td>
<td>1.12</td>
<td>3.46</td>
<td>41.22</td>
<td>985.98</td>
</tr>
<tr>
<td>CQP + TS</td>
<td>1.75</td>
<td>8.92</td>
<td>225.79</td>
<td>1.21</td>
<td>5.25</td>
<td>95.66</td>
<td>1278.72</td>
</tr>
<tr>
<td>LPY</td>
<td>2.00</td>
<td>13.33</td>
<td>442.15</td>
<td>0.81</td>
<td>5.93</td>
<td>68.58</td>
<td>2407.76</td>
</tr>
<tr>
<td>LPY + TS</td>
<td>2.08</td>
<td>15.40</td>
<td>507.83</td>
<td>0.90</td>
<td>7.70</td>
<td>122.55</td>
<td>2684.35</td>
</tr>
<tr>
<td>LPX</td>
<td>0.05</td>
<td>0.41</td>
<td>14.36</td>
<td>0.08</td>
<td>0.59</td>
<td>28.56</td>
<td>684.89</td>
</tr>
<tr>
<td>LPX + TS</td>
<td>0.12</td>
<td>2.04</td>
<td>42.47</td>
<td>0.16</td>
<td>1.98</td>
<td>66.57</td>
<td>883.09</td>
</tr>
</tbody>
</table>

iterative improvement to the solutions obtained by the good start solutions takes about 0.02 seconds for the largest instances. Therefore, the time for obtaining the good start solution hardly differs from the time of obtaining it and then applying iterative improvement to it. Although solving the LPX-relaxation has no polynomial-time guarantee, we see that this method is fastest. The smallest instances are solved in a few seconds. The CQP approach needs on average more than 15 minutes and the LPY approach even needs around 40 minutes for solving instances of size 100 x 50. The LPX approach is much faster: it takes about 10 minutes. The application of tabu search takes about another 3 to 5 minutes for the largest instances.

6 Concluding remarks

Our main goal in this paper was to make an empirical comparison of approximation algorithms with performance guarantees and local search heuristics for $R|| \sum w_jC_j$. The algorithms with worst-case performance guarantees are based on rounding solutions to relaxations of the scheduling problem; these relaxations also provide us with lower bounds. In Section 3, we proved that the best lower bounds are obtained by the LPX-relaxation. In Section 5, we saw that rounding the solution to this relaxation and then applying tabu search on this feasible schedule also yields the best upper bounds.
Comparing our results with the work of Savelsbergh, Uma, and Wein and Uma and Wein on $1|\tau_j|\sum w_jC_j$, we come to the same conclusions: rounding a better relaxation yields a better schedule and the best schedules are obtained by combining a heuristic based on a good relaxation and local search.

With the work of Hariri and Potts and Glass, Potts, and Shade on $R\|C_{\text{max}}$ there is some difference. In an empirical evaluation of heuristics with a constant performance guarantee, Hariri and Potts concluded that the performance of these heuristics is unsatisfactory as a deviation from the optimal solution of more than 10% was normal. Applying a simple improvement procedure resulted in a significant improvement of the schedules. Glass, Potts, and Shade showed that some good local search heuristics yield schedules within 1% of the best found solution. The procedures with a constant performance guarantee that we evaluated resulted in solutions that were within 1 or 2% of optimal and for the $LPX$ approach even within 0.2%. The reason that the heuristics with constant performance guarantee we considered are much better can be found in the objective function. We considered the sum of weighted completion times and each job contributes to the objective function. Some jobs will have lower completion time compared to an optimal schedule and other jobs will have higher completion time. In $R\|C_{\text{max}}$ the objective is makespan minimization, so we only look at the last job to be completed: of course this will never be lower than in an optimal schedule. Also the objective value for the sum of weighted completion times is much larger than for makespan minimization and the same absolute difference yields a lower relative difference for the problem considered in this paper than for makespan minimization problems.

Acknowledgments

The authors thank Jan Karel Lenstra for his useful comments. This research was supported by the project “High performance methods for mathematical optimization” of the Netherlands Organization for Scientific Research (NWO).

References


