On the relationship between combinatorial and LP-based lower bounds for NP-hard scheduling problems

R.N. Uma\textsuperscript{a},*,\textsuperscript{1}, Joel Wein\textsuperscript{b},\textsuperscript{2}, David P. Williamson\textsuperscript{c},\textsuperscript{3}

\textsuperscript{a}Department of Mathematics and Computer Science, North Carolina Central University, 1801 Fayetteville St., Durham, NC 27707, USA
\textsuperscript{b}Department of Computer Science, Polytechnic University, 5 MetroTech Center, Brooklyn, NY 11201, USA
\textsuperscript{c}School of Operations Research & Industrial Engineering and Computing & Information Science, Cornell University, 236 Rhodes Hall, Ithaca, NY 14853, USA

Abstract

Enumerative approaches to solving optimization problems, such as branch and bound, require a subroutine that produces a lower bound on the value of the optimal solution. In the domain of scheduling problems the requisite lower bound has typically been derived from either the solution to a linear-programming (LP) relaxation of the problem or the solution to a combinatorial relaxation. In this paper we investigate, from a theoretical perspective, the relationship between several LP-based lower bounds and combinatorial lower bounds for three scheduling problems in which the goal is to minimize the average weighted completion time of the jobs scheduled.

We establish a number of facts about the relationship between these different sorts of lower bounds, including the equivalence of certain LP-based lower bounds for these problems to combinatorial lower bounds used in successful branch-and-bound algorithms. As a result, we obtain the first worst-case analysis of the quality of the lower bounds delivered by these combinatorial relaxations. © 2006 Elsevier B.V. All rights reserved.

Keywords: Combinatorial lower bounds; LP-based lower bounds; Scheduling

1. Introduction

A well-studied approach to the exact solution of NP-hard scheduling problems may be called \textit{enumerative methods}, in which (implicitly) every possible solution to an instance is considered in an ordered fashion. An example of these methods is branch and bound, which uses upper and lower bounds on the value of the optimal solution to cut down the search space to a (potentially) computationally tractable size. Such methods are typically most effective when

---

* Corresponding author.
E-mail addresses: ruma@nccu.edu (R.N. Uma), wein@mem.poly.edu (J. Wein), dpw@cs.cornell.edu (D.P. Williamson).

1 This work was done while the author was a graduate student at Polytechnic University, Brooklyn, NY. Research partially supported by NSF Grant CCR-9626831, NSF Grant DMI-9970063 and a grant from the New York State Science and Technology Foundation, through its Center for Advanced Technology in Telecommunications. Part of this paper was prepared while this author was affiliated with the University of Texas at Dallas.

2 Research partially supported by NSF Grant CCR-9626831, NSF Grant DMI-9970063, IBM, and a grant from the New York State Science and Technology Foundation, through its Center for Advanced Technology in Telecommunications. Part of this paper was prepared while this author was visiting the IBM T.J. Watson Research Center.

3 This work was done while the author was affiliated with IBM T. J. Watson Research Center and IBM Almaden Research Center.
the subroutines used to calculate both the upper and lower bounds are fast and yield strong bounds, hence quickly eliminating much of the search space from consideration.

Although there are a wealth of approaches to designing the lower-bounding subroutines, we can identify two that have been particularly prominent. The first relies on a linear-programming (LP) relaxation of the problem, which itself is often derived from an integer LP formulation by relaxing the integrality constraints; Queyranne and Schulz give an extensive survey of this approach [21]. The second relies on what we will call a combinatorial relaxation of the problem and yields what we will call a combinatorial lower bound. By this we simply mean that the lower bound is produced by exploiting some understanding of the structure of the problem as opposed to by solving a mathematical program. For example, in this paper we focus on combinatorial lower bounds that are obtained by relaxing the constraint (in a nonpreemptive scheduling problem) that the entire job must be processed in an uninterrupted fashion.

Another approach to an NP-hard scheduling problem is to develop an approximation algorithm. Here the goal is to design an algorithm that runs in polynomial time and produces a near-optimal solution of some guaranteed quality. Specifically, we define a $\rho$-approximation algorithm to be an algorithm that runs in polynomial time and delivers a solution of value at most $\rho$ times optimal; see Hall [15] for a survey. $\rho$ is also referred to as the performance guarantee. In contrast, an enumerative approach attempts to solve a (usually small) problem to optimality, with no guarantee that the solution will be obtained in time polynomial in the size of the input.

Over the past several years, researchers have been successful in creating new connections between LP relaxations used to give lower bounds for certain scheduling problems and the design of approximation algorithms. Specifically, they have used these LP relaxations to develop approximation algorithms with small-constant-factor worst-case performance guarantees; as a by-product one obtains worst-case bounds on the quality of the lower bound delivered by these relaxations [18, 16, 4, 12, 6, 25, 24, 13]. We define a $\rho$-relaxation of a problem to be a relaxation that yields a lower bound that is always within a factor of $\rho$ of the optimal solution.

In this paper, we establish additional connections between different approaches to these problems. We consider three NP-hard scheduling problems in which the goal is to minimize the average weighted completion time of the jobs scheduled: (i) $1|\sum j w_j C_j$, the problem of scheduling $n$ jobs with release dates on a single machine; (ii) $P||\sum j w_j C_j$, the problem of scheduling $n$ jobs on identical parallel machines and (iii) $P|\sum j w_j C_j$, the problem of scheduling $n$ jobs with release dates on identical parallel machines. For each problem we show that a combinatorial lower bound that was used successfully in a branch-and-bound code for the problem is equivalent to the solution of a LP relaxation that has been used in the design of approximation algorithms. As a consequence, we give the first worst-case analysis of these sorts of combinatorial lower bounds. We also consider several related lower bounds and establish a number of facts about their relative strengths.

In all three problems that we consider, we are given $n$ jobs, $j = 1, \ldots, n$. Each job has a nonnegative integer processing time $p_j$ and a weight $w_j$. In the first problem, $1|\sum j w_j C_j$, each job has a release date $r_j$ before which it is unavailable for processing. The goal is to schedule the $n$ jobs nonpreemptively, that is process each job without any interruptions, on one machine so as to minimize their average weighted completion time; namely, if we let $C_j$ denote the completion time of job $j$ in a schedule, then we can express the average weighted completion time as

$$\frac{1}{n} \sum_{j=1}^{n} w_j C_j.$$

In the second and third problems, $P||\sum j w_j C_j$ and $P|\sum j w_j C_j$, we are required to schedule the jobs nonpreemptively on $m$ identical machines, with the goal again being the minimization of average weighted completion time. In the second problem there are no release-date constraints; namely all jobs are available at time 0. In the third problem, there are release date constraints.

1.1. Discussion of previous related work

We begin with $1|\sum j w_j C_j$. Dyer and Wolsey considered LP relaxations of this problem as a tool for producing strong lower bounds [9]. Among those considered were two time-indexed linear programming relaxations, in which the linear program contains a variable for every job at every point in time. In the first relaxation $\{0,1\}$-variables $y_{jt}$ determine whether job $j$ is processed during time $t$, whereas in a second stronger relaxation $\{0,1\}$-variables $x_{jt}$ determine whether job $j$ completes at time $t$.

Although both linear programs are of exponential size, Dyer and Wolsey showed that the $y_{jt}$-LP is a transportation problem with a very special structure and thus can be solved in $O(n \log n)$ time [12]. The $x_{jt}$-LP, which has been observed empirically to give strong lower bounds [27, 28], is very difficult to solve due to its size.
Van den Akker et al. [30] developed a column-generation approach to solving these linear programs that made feasible the solution of instances with up to 50 jobs with processing times in the range of 0–30.

Inspired by the empirical strength of this relaxation, Hall et al. [16] gave a 3-approximation algorithm for \| r_j \| \sum w_j C_j based on time-indexed linear programs. Their approximation algorithm in fact relies only on the weaker \( y_{jt} \)-relaxation and simultaneously proves that the \( y_{jt} \)-LP (and hence the stronger \( x_{jt} \)-LP) are 3-relaxations of the problem. Subsequent papers gave improved algorithms based on this LP with better constant performance guarantees [16,4,12,6,25,24,13]. Among these, the result due to [13] gives the best performance guarantee of 1.6853. Queyranne and Wang showed that one cannot get better than a \( e/(e-1) \approx 1.58 \)-approximation algorithm based on a relaxation equivalent to the \( y_{jt} \)-relaxation. 4

In parallel with work on LP lower bounds for \( 1| r_j | \sum w_j C_j \), there has been much work on branch-and-bound algorithms for \( 1| r_j | \sum w_j C_j \) based on combinatorial lower bounds [22,5,8,3,19,17,2]. The most successful of these is due to Belouadah et al. [2] who made use of two combinatorial lower bounds based on job splitting, and an upper bound based on a simple greedy heuristic.

Although it is difficult to compare the efficacy of the branch-and-bound code of Belouadah, Posner and Potts with the branch-and-cut code due to Van den Akker et al. based on \( x_{jt} \)-relaxations [29] (since they were developed several years apart in different programming languages on different architectures, etc.) the evidence seems to be that neither much dominates the other; however, that of Belouadah et al. seems to have been somewhat stronger, as they were able to solve to optimality problems with processing times up to about 50 whereas Van den Akker et al. solved to optimality problems with processing times up to 30. In essence, the enhanced strength of the lower bounds due to the \( x_{jt} \)-relaxations does not appear to make up for the amount of time it takes to solve them.

1.2. Discussion of results

This paper was born out of an interest to make more precise the comparison between the LP-based techniques and the techniques associated with the best combinatorial branch-and-bound algorithm. In this process, several interesting relationships between these two approaches arose. Specifically, we show that the solution delivered by the \( y_{jt} \)-based relaxation for \( 1| r_j | \sum w_j C_j \) is identical to that used to deliver the weaker of the two lower bounds used by Belouadah et al. We present our proof using algebra and two-dimensional Gantt charts [10,14]. Two-dimensional Gantt charts have been used by Eastman et al. [10] and by Goemans and Williamson [14] as a graphical method of proving theorems for related scheduling problems. We also show that the stronger of the two lower bounds due to Belouadah et al., while empirically usually weaker than the \( x_{jt} \)-based relaxation, neither dominates that lower bound nor is it dominated by it. A corollary of this observation is that the optimal preemptive schedule for an instance of \( 1| r_j | \sum w_j C_j \) neither dominates nor is dominated by the solution to the \( x_{jt} \)-relaxation.

We then establish a similar relationship for a different problem. Webster [31] gave a series of lower bounds for \( P|| \sum w_j C_j \) that are based on a notion similar to the job-splitting approach of Belouadah et al. The weakest of his lower bounds was in fact originally proposed by Eastman et al. in 1964 [10]. We show that the Eastman et al. bound is identical to the bound obtained from a generalization of the \( y_{jt} \)-relaxation to parallel machines.

In the next section we review the relevant lower bounds. We present our results on the strength of different lower bounds in Section 3. Section 3.1 contains the results for the single machine case and Section 3.2 for the case of parallel machines.

2. Background

In this section we describe how the lower bounds are computed for the single machine problem \( 1| r_j | \sum w_j C_j \). To compute lower bounds, we first obtain a relaxation to the problem and then solve the relaxed version of the problem. The solution to the relaxed version gives the lower bound on the cost of the original problem. A relaxation is obtained by slackening some of the input constraints in the problem; that is, by making some of the input constraints less rigid or less stringent. For example, lower bounds obtained through LP-based relaxations allow the scheduling of fractions 5 of

---

4 Note that a polynomial time approximation scheme (PTAS) exists for \( 1| r_j | \sum w_j C_j \) [1].

5 The fractions could be either preemptive fractions as in the \( y_{jt} \)-relaxation or nonpreemptive fractions as in the \( x_{jt} \)-relaxation. Furthermore, there is no reassignment of the weights of the jobs to their component fractions.
jobs, and lower bounds obtained through combinatorial relaxations allow jobs to be broken down into smaller pieces\(^6\) followed by the scheduling of the smaller pieces. In this section, we will only present the LP-based relaxations and defer the discussion on combinatorial relaxations to Section 3.1.

### 2.1. LP-relaxations

We begin with the two relevant LP relaxations of \(1|\sum w_j C_j\). As mentioned earlier, Dyer and Wolsey [9] introduced several integer LP formulations of the problem. We focus on two LP formulations—\(y_{jt}\) and \(x_{jt}\)—that have been useful in the design of approximation algorithms.

**The \(y_{jt}\)-LP:**

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{n} w_j \left( \frac{p_j}{2} + \frac{1}{p_j} \sum_{t=2}^{T} \left( t - \frac{1}{2} \right) y_{jt} \right) \\
\text{subject to} & \quad \sum_{j=1}^{n} y_{jt} \leq 1, \quad t = 1, \ldots, T, \\
& \quad \sum_{t=r+j+1}^{T} y_{jt} = p_j, \quad j = 1, \ldots, n, \\
& \quad y_{jt} \in \{0, 1\}, \quad j = 1, \ldots, n, \quad t = r+j+1, \ldots, T.
\end{align*}
\]

The binary variable \(y_{jt}\) for each job \(j\) \((j = 1, \ldots, n)\) and time period \([t-1, t]\) \((t = 1, \ldots, T)\) indicates whether job \(j\) is processed in period \([t-1, t]\) \((y_{jt} = 1)\) or not \((y_{jt} = 0)\). \(T = \max_j r_j + \sum_j p_j\) is an upper bound on the makespan of the schedule. The term \(p_j/2 + 1/p_j \sum_{t=r+j+1}^{T} (t - \frac{1}{2}) y_{jt}\) in the objective function corresponds to the actual completion time of job \(j\) if the job were continuously processed from \(C_j - p_j\) to \(C_j\) (where \(C_j\) is the completion time of job \(j\)). The second term in this expression, namely \((1/p_j) \sum_{t=r+j+1}^{T} (t - \frac{1}{2}) y_{jt}\), computes the midpoint of the computation for each job \(j\) (using the middle of each time unit when the job runs). Adding to this the remaining half of the processing time \(p_j/2\) gives the effective completion time of job \(j\) in the schedule. The term \(p_j/2 + 1/p_j \sum_{t=r+j+1}^{T} (t - \frac{1}{2}) y_{jt}\) in the objective function corresponds to the actual completion time of job \(j\) if the job were continuously processed from \(C_j - p_j\) to \(C_j\) (where \(C_j\) is the completion time of job \(j\)). The second term in this expression, namely \((1/p_j) \sum_{t=r+j+1}^{T} (t - \frac{1}{2}) y_{jt}\), computes the midpoint of the computation for each job \(j\) (using the middle of each time unit when the job runs). Adding to this the remaining half of the processing time \(p_j/2\) gives the effective completion time of job \(j\) in the schedule. The constraints (3) require that each job should be processed in its entirety between its release date \(r_j\) and \(T\). The capacity constraints (2) state that the machine can handle at most one job during any time period.

In a relaxation to the integer program (1) –(4), the integrality constraint (4) is relaxed to

\[
0 \leq y_{jt} \leq 1, \quad j = 1, \ldots, n, \quad t = r+j+1, \ldots, T.
\]

This linear program is a valid relaxation of the optimal preemptive schedule as well [16], and can be solved in \(O(n \log n)\) time [9]. The structure of the solution is in fact quite simple: at any point in time, schedule the available unfinished job with maximum \(w_j/p_j\) (this may involve preemption) [12]. As a result, the \(y_{jt}\) variables take values either 0 or 1 in the solution to the relaxation.

In the second linear program, which is much harder to solve, there is a binary variable \(x_{jt}\) for each job \(j\) \((j = 1, \ldots, n)\) and time period \([t-1, t]\) \((t = p_j, \ldots, T)\), where \(T = \max_j r_j + \sum_j p_j\) is an upper bound on the schedule’s makespan. The variable \(x_{jt}\) indicates whether job \(j\) completes in period \([t-1, t]\) \((x_{jt} = 1)\) or not \((x_{jt} = 0)\). This relaxation is stronger than the \(y_{jt}\)-relaxation; in particular it is not a valid relaxation of the optimal preemptive schedule, and its integer solutions yield only nonpreemptive schedules.

**The \(x_{jt}\)-LP:**

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{n} \sum_{t=p_j}^{T} w_j \cdot t \cdot x_{jt}, \\
\end{align*}
\]

\(^{6}\)The pieces can be viewed as sub-jobs. The weights of the jobs are reassigned to their component pieces.
subject to
\[ \sum_{t=r_j+p_j}^{T} x_{jt} = 1, \quad j = 1, \ldots, n, \quad (7) \]
\[ \sum_{j=1}^{n} \sum_{s=t}^{t+p_j-1} x_{js} \leq 1, \quad t = 1, \ldots, T, \quad (8) \]
\[ x_{jt} \geq 0, \quad j = 1, \ldots, n, \quad t = r_j + p_j, \ldots, T. \quad (9) \]

The assignment constraints (7) state that each job has to be completed exactly once, and the capacity constraints (8) state that the machine can process at most one job during any time period.

3. Analytical evaluation of strength of different lower bounds

In this section we compare the lower bounds obtained through combinatorial relaxations with lower bounds obtained through LP-relaxations for single machine scheduling problems and parallel machine scheduling problems. The problems discussed in this section are complex variants of a fundamental scheduling problem, namely, 1||\( \sum w_j C_j \); this is a single machine scheduling problem with no side constraints. Smith [26] showed that this problem can be solved optimally in polynomial time by scheduling the jobs in nonincreasing order of their \( w_j / p_j \) ratios. This ordering is commonly referred to as Smith’s rule.

3.1. One machine

For the single machine scheduling problem 1\( |r_j| \sum w_j C_j \), we compare the two combinatorial lower bounds (\( \text{BPP}1 \) and \( \text{BPP}2 \)) with the two LP-based lower bounds (\( y_{jt} \) and \( x_{jt} \)). First, we describe in more detail how the combinatorial lower bounds are computed.

Belouadah et al. [2] obtain a relaxation by allowing jobs to be broken into smaller pieces and then constructing a schedule of these smaller pieces. Let \( \mathcal{I} \) be the given instance of 1\( |r_j| \sum w_j C_j \). Each relaxation corresponding to \( \text{BPP}1 \) and \( \text{BPP}2 \) is also an instance of 1\( |r_j| \sum w_j C_j \) but with smaller pieces (of jobs) than in \( \mathcal{I} \); what is relaxed in the instances corresponding to \( \text{BPP}1 \) and \( \text{BPP}2 \) is that although they contain the same set of jobs as in instance \( \mathcal{I} \), some (or all) of the jobs are split into smaller pieces. Let \( \mathcal{I}_{\text{BPP}1} \) correspond to the instance of \( \text{BPP}1 \) and \( \mathcal{I}_{\text{BPP}2} \) correspond to the instance of \( \text{BPP}2 \). We proceed to describe how \( \mathcal{I}_{\text{BPP}1} \) and \( \mathcal{I}_{\text{BPP}2} \) are generated from \( \mathcal{I} \). By virtue of the method of generating \( \mathcal{I}_{\text{BPP}1} \) and \( \mathcal{I}_{\text{BPP}2} \) we can at the same time describe the optimal schedule for \( \mathcal{I}_{\text{BPP}1} \) and \( \mathcal{I}_{\text{BPP}2} \). To obtain instances \( \mathcal{I}_{\text{BPP}1} \) and \( \mathcal{I}_{\text{BPP}2} \) from \( \mathcal{I} \), Belouadah et al. [2] make use of the shortest weighted processing time (SWPT) heuristic which, at each decision point in time, schedules the job with the lowest \( p_j / w_j \) value or equivalently the highest \( w_j / p_j \) value from among the available jobs. They run the shortest weighted processing time heuristic on the original instance \( \mathcal{I} \), breaking the currently executing job/piece whenever a job with a higher priority (that is, higher \( w_j / p_j \)) arrives. Therefore, the length of the currently executing piece is determined by the arrival of a new job. The weights of the pieces are determined by one of two schemes corresponding to either \( \text{BPP}1 \) or \( \text{BPP}2 \). The execution of the heuristic is resumed on the remaining pieces/jobs. Once we have a collection of the pieces with their respective weight assignments, we schedule the pieces optimally. The weights are assigned to the pieces in such a fashion that applying the SWPT heuristic on the resulting pieces yields an optimal schedule of the pieces. Therefore, the cost of this optimal solution gives the corresponding lower bound. Note that instances \( \mathcal{I}_{\text{BPP}1} \) and \( \mathcal{I}_{\text{BPP}2} \) contain the same set of jobs/pieces with identical processing times; they only differ in the weights assigned to the pieces.

We now describe in detail how instances \( \mathcal{I}_{\text{BPP}1} \) and \( \mathcal{I}_{\text{BPP}2} \) are constructed from \( \mathcal{I} \) and why they give a valid relaxation. We say job \( l \) is “better” than job \( j \) if \( p_l / w_l < p_j / w_j \), or, equivalently, if \( w_l / p_l > w_j / p_j \). We begin at time 0 by processing the best available job. When a better job arrives, we split the currently executing job into two pieces such that one piece completes at the arrival time of the new job and the second piece is considered for scheduling later. When a job is split into pieces, its weight is also split. So if job \( j \) is split into \( k \) pieces, then each piece \( i \) has a processing time \( p_j^i \), a weight \( w_j^i \) and release date \( r_j \), such that \( \sum_{i=1}^{k} p_j^i = p_j \) and \( \sum_{i=1}^{k} w_j^i = w_j \).

As before, let \( \mathcal{I} \) be an instance of 1\( |r_j| \sum w_j C_j \). To see why optimal solutions to the BPP instances \( \mathcal{I}_{\text{BPP}1} \) and \( \mathcal{I}_{\text{BPP}2} \) give valid lower bounds to an optimal solution to \( \mathcal{I} \), we first consider an instance \( \mathcal{I}_1 \) that is intermediate between
I and the instances \( I_{BPP1} \) and \( I_{BPP2} \). In \( I_1 \) we constrain the pieces of each job to be scheduled contiguously; that is, we impose a contiguity constraint on \( I_1 \). For ease of presentation, we assume only one job \( j \) (with processing time \( p_j \), release date \( r_j \), and weight \( w_j \) and weight \( w_j \) and release date \( r_j \)) was split into \( k \) pieces in \( I_1 \). Note that the following argument holds even if more than one job is split. Say each piece \( i \) of job \( j \) is of length \( p^i_j \) and is assigned a weight \( w^i_j \) for \( i = 1, \ldots, k \) such that \( \sum_{i=1}^{k} p^i_j = p_j \) and \( \sum_{i=1}^{k} w^i_j = w_j \). The set of jobs in \( I_1 \) are the \( k \) pieces of the split job \( j \) plus the remaining jobs from \( I \). Obviously there is a one-to-one correspondence between feasible schedules for \( I \) and for \( I_1 \). Note that in \( I_1 \), the pieces of job \( j \) will be scheduled exactly during the interval \( I \) schedules job \( j \). Therefore, it is sufficient, for our argument, to consider the contribution of this one job to the weighted completion time in both schedules. Let the \( k \) pieces of job \( j \) start at times \( t^1_j, \ldots, t^k_j \), respectively. So in \( I \), this job is scheduled during \([t^1_j, t^1_j + p_j]\) and it contributes \( w_j \cdot (t^1_j + p_j) \).

The following equality was shown by Belouadah et al. in [2].

\[
 w_j \cdot (t^1_j + p_j) = \sum_{i=1}^{k} w_j \cdot C^i_j + CBRK_j,
\]

where \( C^i_j \) denotes the completion time of piece \( i \) of job \( j \) and

\[
 CBRK_j = \sum_{i=1}^{k-1} w_j \sum_{h=i+1}^{k} p^h_j
\]

can be thought of as the cost of breaking job \( j \) into \( k \) pieces.

The cost of a schedule of \( I \) is equal to the cost of the corresponding schedule of \( I_1 \) for any weight assignment to the pieces \( I \) (due to (10) and the one-to-one correspondence between schedules of \( I \) and \( I_1 \)). That is, \( \text{cost}(I) = \text{cost}(I_1) + \text{CBRK}_j \). Let us now relax \( I_1 \) further by removing the contiguity constraint. Let the resulting instance be \( I_2 \). Instance \( I_2 \) is a relaxation to \( I_1 \) and to \( I \) also. Therefore, the cost of the optimal solution to instance \( I_2 \) is a lower bound on the cost of the optimal solution to instance \( I \). That is, \( \text{cost}(I_2) + \text{CBRK}_j \leq \text{cost}(I_1) + \text{CBRK}_j = \text{cost}(I) \). This can be generalized even if more than one job is split: \( \text{cost}(I_2) + \sum_j \text{CBRK}_j \leq \text{cost}(I_1) + \sum_j \text{CBRK}_j = \text{cost}(I) \). Therefore, the idea is to split the jobs so that the optimal schedule for the resulting instance \( I_2 \) can be computed easily.

Instances \( I_{BPP1} \) and \( I_{BPP2} \) contain the same collection of split jobs as in instance \( I_2 \) with identical processing times but differ in the assignment of weights to the pieces of the split job. For the \( BPP1 \) bound which is the cost of the optimal solution of instance \( I_{BPP1} \), the weights are assigned to the pieces of a job such that \( w^i_j / p^i_j = w_j / p_j \) for all \( i = 1, \ldots, k \). For the \( BPP2 \) lower bound which is the cost of the optimal schedule of instance \( I_{BPP2} \), the weights are assigned in a greedy fashion so as to give as much weight as possible to later scheduled pieces of the job while maintaining the invariant that at each moment in the schedule the job being scheduled is better than any available job. \(^8\)

The \( BPP1 \) and \( BPP2 \) weight assignments are such that the SWPT heuristic yields optimal schedules for instances \( I_{BPP1} \) and \( I_{BPP2} \). Belouadah, Posner and Potts showed that the \( BPP2 \) lower bound is always greater than or equal to the \( BPP1 \) lower bound.

Let us denote the lower bound given by \( BPP1 \) as \( LB^{BPP1} \) and the lower bound given by the \( y_{jt} \) LP relaxation as \( LB^{y} \); likewise, \( LB^{BPP2} \) and \( LB^{x} \) will denote the lower bounds given by \( BPP2 \) and the \( x_{jt} \) LP relaxation, respectively.

**Theorem 3.1.** \( LB^{y} = LB^{BPP1} \).

Our proof makes use of two-dimensional Gantt charts [10,14]. Therefore, we first describe two-dimensional Gantt charts before presenting the proof.

Traditional Gantt charts can be considered one-dimensional with only one axis representing processing time (Fig. 1).

\( ^7 \) Of course, the sum of the weights of the pieces of a job should equal the weight of the job.

\( ^8 \) We omit the details of the weight assignment for \( BPP2 \).
Fig. 1. Representation of a schedule using 1D Gantt Chart. The length of each job denotes its processing time. The x-axis denotes time and the y-axis denotes the capacity of the machine (which is 1 unit).

Fig. 2. Representation of a schedule using 2D Gantt Chart. The length of each job denotes its processing time (represented along the x-axis) and the height of each job denotes its weight (represented along the y-axis). The jobs are represented by the patterned (not shaded) rectangles. The weighted completion time of the schedule is the patterned and shaded area. The weighted completion time of each job is the area of the rectangle that represents the job plus the area to its direct left bounded by the y-axis.

Using 1D Gantt charts, the makespan (length) of the schedule can be represented pictorially. But the drawback of 1D Gantt charts is that the weighted completion time objective cannot be represented pictorially. This drawback is overcome in 2D Gantt charts by using a second axis to represent the weights of the jobs. In this representation, each job is a rectangle of length equal to its processing time and height equal to its weight; see Fig. 2 (for now disregard the two solid-color-shaded rectangles). We define the quantity, slope of job $j = w_j/p_j$, where $p_j$ is the processing time of job $j$ and $w_j$ is the weight of job $j$. So the slope of job $j$ is essentially the slope of the diagonal that runs from the lower left corner to the upper right corner of the rectangle that represents job $j$. Under this representation, the weighted completion time is the patterned and shaded area as shown in Fig. 2.

We now turn to proving our theorem.

**Proof of Theorem 3.1.** First we remark that the structure of the schedule given by $BPP1$ is the same as that given by the $y_{jt}$ LP solution. That is, the time intervals in which a job $j$ is processed are exactly the same in both the cases. Recall that the SWPT heuristic computes optimal $BPP1$ schedule.

Consider the (weighted) contribution of just one job, say $j$, to the respective lower bounds (denote these contributions as $LB_{j}^{BPP1}$ and $LB_{j}^{BPP1}$). Let job $j$ be released at $r_j$ with a processing time requirement of $p_j$ and weight $w_j$. Let this job be split into $k$ pieces of lengths $p_{j1}, \ldots, p_{jk}$ starting at times $t_{j1}, \ldots, t_{jk}$, respectively (in $BPP1$). So we have $\sum_{i=1}^{k} p_{ji} = p_j$. $BPP1$ would assign weights $w_{ji} = (p_{ji}/p_j)w_j$ for $i = 1, \ldots, k$. The cost of breaking job $j$ is
The cost of breaking a job can be represented in a 2D Gantt chart as shown in Fig. 3 (where the cost of breaking job 2 (middle job) is indicated). The middle job is broken into three pieces. This leaves out a reverse “L” shaped region that was part of the unsplit job which is indicated by “CBRK” in Fig. 3.

A sample schedule that contains the pieces of the split job as well as the remaining unsplit jobs is shown in Fig. 4 (disregard the solid-color-shaded rectangles for now). Let job $j$ be the split job. The contribution of job $j$ to the $BPP_1$ lower bound is represented by the patterned and shaded area in Fig. 4, excluding the rectangles with dashed-line boundaries. That is,

$$L_{BPP}^j = \sum_{i=1}^{k} w_j^i (t_j^i + p_j^i) + CBRK_j$$

$$= \sum_{i=1}^{k} w_j^i (t_j^i + p_j^i) + \frac{1}{2} w_j p_j - \frac{1}{2} \sum_{i=1}^{k} w_j^i p_j^i.$$
Fig. 4. A sample schedule of the pieces of the split job and the remaining jobs. The piece indicated by “CBRK” is not part of the schedule but contributes to the lower bound. The patterned and shaded area, disregarding the rectangles with dashed-line boundaries, indicates the contribution of the split job to the BPP lower bound.

Fig. 5. A rearrangement of the areas in Fig. 4 (corresponding to the pieces of the split job) to picture the representation of the yjt lower bound. Trapezoid ABCD corresponding to time period [t − 1, t] is highlighted.

as composed of trapezoids and triangles. We can rearrange some of these pieces to get Fig. 5. We will show that the patterned and shaded area in Fig. 5 is exactly the contribution of job j to the yjt lower bound. Recall that in the solution (given by Dyer and Wolsey [9] and by Goemans [12]) to the yjt LP, yjt is set to 1 if job j is processed in the time period [t − 1, t] and it is set to 0 otherwise. So each trapezoid in Fig. 5 corresponds to a time period [t − 1, t] when the job is being processed. Let us focus on the trapezoid ABCD in Fig. 5. The area of trapezoid ABCD is \( \frac{1}{2} \cdot (AB + DC) \cdot AD \). The slope of line BC is the slope of the split job which is \( m_j = \frac{w_j}{p_j} \). Let c be the point where line BC intersects the weight axis. So the coordinates of vertex B are \( (t - 1, m_j \cdot (t - 1) + c) \) and those of vertex C are \( (t, m_j \cdot t + c) \).
Table 1
An instance where $LBP^{PP2}$ dominates $LB^{S_j}$ (i.e., $LBP^{PP2} > LB^{S_j}$); $LBP^{PP2} = 558.00$ and $LB^{S_j} < 555.33$

<table>
<thead>
<tr>
<th>$j$</th>
<th>$r_j$</th>
<th>$p_j$</th>
<th>$w_j$</th>
<th>$w_j/p_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>8.00</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>2.33</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>1.50</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>9</td>
<td>1.80</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>2.00</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>2.67</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>2.00</td>
</tr>
</tbody>
</table>

So we have,

Patterned shaded area in Fig. 5

$$\frac{w_j p_j}{2} + (\text{sum of areas of the trapezoids})$$

$$= \frac{w_j p_j}{2} + \sum_j y_{jt} \cdot (\frac{1}{2} \cdot ((t-1) + t) \cdot ((m_j \cdot t + c) - (m_j \cdot (t-1) + c)))$$

$$= \frac{w_j p_j}{2} + \sum_j y_{jt} \cdot \left(\left(\frac{t}{2}\right) \cdot m_j\right)$$

$$= \frac{w_j p_j}{2} + \sum_j y_{jt} \cdot \left(\frac{w_j}{p_j} \cdot \left(\frac{t}{2}\right)\right)$$

Therefore, it follows that $LBP^{PP1} = LB^{y_{jt}}$. Summing over all jobs $j$ we have the required result. □

As an immediate corollary, we obtain an upper bound on the quality of the lower bounds provided by both $BPP1$ and $BPP2$. Goemans et al. [13] proved that the $y_{jt}$-relaxation is a 1.685-relaxation of $1|r_j|\sum w_j C_j$; thus, we see that $BPP1$ and $BPP2$ are as well. We now turn to the relationship with the $x_{jt}$-relaxation; it is known that this is stronger than the $y_{jt}$-relaxation [9].

**Theorem 3.2.** The lower bound given by $BPP2$ neither always dominates nor is dominated by the $x_{jt}$-lower bound.

The proof is by exhibiting two instances; one on which $BPP2$ is better and one on which $x_{jt}$ is better. For the instance in Table 1, the lower bound given by $BPP2$ (see Fig. 6(b)) dominates the $x_{jt}$-lower bound (see Fig. 6(a)). For the instance in Table 2, the $x_{jt}$-lower bound (see Fig. 7(a)) dominates the lower bound given by $BPP2$ (see Fig. 7(b)).

As discussed earlier, since the $BPP2$ bound is always dominated by the optimal preemptive schedule, we have the following corollary.

**Corollary 3.3.** The solution to the optimal preemptive schedule for an instance of $1|r_j|\sum w_j C_j$ neither dominates nor is dominated by the $x_{jt}$-lower bound.

To the best of our knowledge this has not been observed before, and is interesting since the $x_{jt}$-based relaxation is a relaxation only of nonpreemptive schedules and not of preemptive schedules.
Fig. 6. Solutions to the $x_{jt}$ lower bound and $BPP_2$ lower bound for the instance in Table 1. The circled numbers indicate the job id's. The $x$-axis denotes time and the $y$-axis denotes the capacity of the machine (which is 1 unit). (a) $LB^{x_{jt}} \leq 555.33$. (b) The numbers in parentheses above each job denotes the weight assigned to the pieces by $BPP_2$. $LB^{BPP_2} = 558.00$.

Table 2

An instance where $LB^{x_{jt}}$ dominates $LB^{BPP_2}$ (i.e., $LB^{x_{jt}} > LB^{BPP_2}$); $LB^{BPP_2} = 115.00$ and $LB^{x_{jt}} = 120.00$

<table>
<thead>
<tr>
<th>$j$</th>
<th>$r_j$</th>
<th>$p_j$</th>
<th>$w_j$</th>
<th>$w_j/p_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0.10</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Fig. 7. Solutions to the $x_{jt}$ lower bound and $BPP_2$ lower bound for the instance in Table 2. The circled numbers indicate the job id's. The $x$-axis denotes time and the $y$-axis denotes the capacity of the machine (which is 1 unit). (a) The shaded rectangle to the left of job 2 denotes machine idle time. $LB^{x_{jt}} \leq 120.00$. (b) The numbers in parentheses above each job denotes the weight assigned to the pieces by $BPP_2$. $LB^{BPP_2} = 115.00$.

Finally, we note that although $BPP_2$ dominates $BPP_1$ and the equivalent $y_{jt}$-based lower bound, the examples given by Queyranne and Wang, that show that the $y_{jt}$-based lower bound can be a factor of $e/(e - 1)$ from optimal, yield the same bound for the $BPP_2$ bound.
Consider the instance given by Queyranne and Wang. There are \((n + 1)\) jobs with one large job denoted \(j = 1, \ldots, n\) and \(n\) small jobs denoted \(j = 1, \ldots, n\). The large job has processing time \(p_0 = 1\), weight \(w_0 = 1\) and release date \(r_0 = 0\). Each of the \(n\) small jobs \(j\), for \(j = 1, \ldots, n\), has processing time \(p_j = p < 1/n\), weight \(w_j = 1/(n - 1) \cdot (n/(n - 1))^{(n-j)}\) and release date \(r_j = (j - 1)/n\). First, we let \(p\) go to zero for any fixed \(n\) and then we let \(n\) go to infinity. Since \(p\) goes to zero, the small jobs have a higher \(w_j/p_j\) than the large job, or the split pieces of the large job (irrespective of the weight assignment to the pieces). The \(BPP_2\) bound therefore splits the large job whenever a small job is released starting from the second small job. The large job is split into \(n\) pieces where each piece has processing time \(1/n\). The first \((n - 1)\) pieces are assigned a weight 0 and the last \(n\)th piece is assigned all of the weight of the large job (= 1). The structure of the \(BPP_2\) schedule is therefore a small job followed by a piece of the large job, repeated for each small job; the small jobs are scheduled in order of their release dates. The cost of the \(BPP_2\) schedule as \(p\) goes to zero is

\[
\sum_{j=0}^{n} w_j C_j = 1 + \sum_{j=1}^{n} \frac{1}{n-j} \cdot \left(\frac{n}{n-1}\right)^{n-j} \left(\frac{j-1}{n}\right)
\]

\[
= \left(\frac{n}{n-1}\right)^{n-1} \left(\frac{1}{n-1}\right)
\]

\[
= a(n)^{n-1} \left(\frac{1}{n}\right)
\]

\[
= a(n)^{n-1} - \frac{1}{a(n)}
\]

where \(a(n) = n/(n - 1)\).

Queyranne and Wang show that the optimal schedule for their instance is of the following type, for some \(k \in \{1, \ldots, n\}:\)

\[
C^k_j = r_j + p_j \quad \text{for} \quad j = 1, \ldots, k,
\]

\[
C^k_0 = r_k + p_k + 1,
\]

\[
C^k_j = r_k + p_k + 1 + \sum_{i=k+1}^{j} p_i \quad \text{for} \quad j = k + 1, \ldots, n.
\]

That is, the optimal schedule processes jobs \(j = 1, \ldots, k\) at their release dates, processes job 0 (nonpreemptively) from \(r_k + p_k\) to \(r_k + p_k + 1\) and processes jobs \(j = k + 1, \ldots, n\) consecutively in that order starting at the completion time of job 0. The cost of the optimal solution is

\[
\sum_{j=0}^{n} w_j C^k_j = \sum_{j=1}^{k} w_j C^k_j + w_0 C^k_0 + \sum_{j=k+1}^{n} w_j C^k_j
\]

\[
= \sum_{j=1}^{k} w_j (r_j + p_j) + (r_k + p_k + 1) + \sum_{j=k+1}^{n} w_j \left( r_k + p_k + 1 + \sum_{i=k+1}^{j} p_i \right)
\]

\[
= \sum_{j=1}^{k} w_j (r_j + p) + (r_k + p + 1) + \sum_{j=k+1}^{n} w_j \left( r_k + p + 1 + \sum_{i=k+1}^{j} p \right)
\]

\[
= p \sum_{j=1}^{n} w_j + \sum_{j=1}^{k} w_j r_j + (r_k + 1) \left( 1 + \sum_{j=k+1}^{n} w_j \right) + p + \sum_{j=k+1}^{n} w_j \sum_{i=k+1}^{j} p
\]

\[= \cdot \cdot \cdot \text{(omitting some details)}\]
\[ p(a(n)^{n-1} - a(n)^{n-k}) + \left(\frac{k-1}{n-1}\right) a(n)^{n-k-1} + \left(\frac{k-1}{n} + 1\right) a(n)^{n-k} + p + pn(a(n)^{n-1-k-1} - (n-k)p \]

As \( p \) goes to 0, the optimum cost \( \sum w_j C_j^k \) goes to \( ((n-1)/n) a(n)^{n-1} \). All \( C_k \) are asymptotically optimal for \( k > 0 \).

The ratio of cost of optimal schedule to cost of \( \text{BPP}^2 \) schedule as \( n \) goes to infinity is

\[ \lim_{n \to \infty} \frac{a(n)^{n-1}}{(a(n)^n - 1)/a(n)} = \lim_{n \to \infty} \frac{a(n)^n}{a(n)^n - 1} = \frac{e}{e-1} \]

since \( \lim_{n \to \infty} a(n)^n = e \).

**Theorem 3.4.** \( \text{BPP}^2 \) is no better than an \( e/(e-1) \)-relaxation of \( 1|r_j| \sum w_j C_j \).

### 3.2. Parallel identical machines

In this subsection we consider two parallel machine scheduling problems—one without release dates \( P||\sum w_j C_j \) and the other with release dates \( P|r_j| \sum w_j C_j \). We consider LP-relaxations of the parallel machine scheduling problems that are direct analogues to the LP-relaxations of the one machine case and show that the lower bounds yielded by these LP-relaxations are equivalent to certain combinatorial lower bounds given in the literature for these parallel machine scheduling problems.

\( P||\sum w_j C_j \) is the problem of scheduling \( n \) jobs on \( m \) parallel machines with the goal of minimizing the total weighted completion time. Each job has a processing requirement \( p_j \) and a weight \( w_j \). For this problem, Webster [31] gives a series of progressively stronger lower bounds all of which are based on ideas similar to the job splitting ideas of [2]. The weakest of his lower bounds is originally due to Eastman et al. [10].

Eastman et al. [10] do not use the idea of job splitting in their combinatorial lower bound. Instead they relate the cost of scheduling \( n \) jobs on \( m \) identical machines to the cost of scheduling the \( n \) jobs on one machine and thereby obtain a lower bound in terms of the cost of scheduling on the single machine. Let \( (C_j^m) \) denote the completion time of job \( j \) in an optimal schedule in the 1-machine environment and let \( (C_j^1) \) denote the completion time of job \( j \) in an optimal schedule in the \( m \)-machine environment. Note that the 1-machine case can be solved in polynomial time by scheduling the jobs using Smith’s [26] rule; that is, schedule the jobs in nonincreasing order of their \( w_j/p_j \) ratios. The following result is the lower bound\(^9\) given by Eastman et al. [10].

\[ \sum j w_j (C_j^m) \geq \frac{1}{m} \sum j w_j (C_j^1) + \frac{m-1}{2m} \sum j w_j p_j. \]

Now let us consider an LP-relaxation for \( P||\sum w_j C_j \). Let \( M_j \) denote the time of processing the midpoint of job \( j \) and let \( p(S) = \sum_{j \in S} p_j \). \( N \) denotes the set of all \( n \) jobs. One of the standard LPs for \( 1||\sum w_j C_j \) with the change of variable \( M_j = C_j - p_j/2 \) [32,9,20,11] is

\[ \text{minimize} \sum_{j=1}^{n} w_j \left( M_j + \frac{p_j}{2} \right) \]

\[ \sum_{j \in S} p_j M_j \geq \frac{1}{2} p(S)^2, \quad S \subseteq N. \]

\(^9\) For details on how this lower bound is obtained, see [10]. Intuitively, the \( m \)-machine schedule cost should be at least an \( m \)th-fraction of the 1-machine schedule cost.
As has been shown in [23], we can use the LP in which we simply divide the right-hand sides by \( m \) to get a valid relaxation for \( P || \sum w_j C_j \):

\[
\text{minimize } \sum_{j=1}^{n} w_j \left( M_j + \frac{P_j}{2} \right) \tag{13}
\]

\[
(M) \sum_{j \in S} p_j M_j \geq \frac{1}{2m} p(S)^2, \quad S \subseteq N. \tag{14}
\]

The linear program \((M)\) is the parallel machine analogue of the linear program for one machine given by (1). Let \( LP_1 \) denote the optimal solution to the LP (1) and \( LP_m \) denote the optimal solution to the relaxation given by the LP \((M)\). We now give a simple expression for \( LP_m \). First observe that we know the optimal solution to LP \((1)\) via Smith’s rule. There is a one-to-one correspondence between solutions to \((1)\) and \((M)\). For any solution to \((1)\) we can obtain a solution to \((M)\) by dividing each variable by \( m \). In particular, let the jobs be indexed so that \( \frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \cdots \geq \frac{w_n}{p_n} \); this is Smith’s ordering. It can be shown that the optimal value to (1) is

\[
LP_1 = \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i
\]

\[
= \frac{1}{2} \sum_{j=1}^{n} w_j \left( \left( \sum_{i=1}^{j} p_i \right)^2 - \left( \sum_{i=1}^{j-1} p_i \right)^2 \right) + \sum_{j=1}^{n} \frac{w_j p_j}{2}.
\]

Thus a solution to the relaxation given by the LP \((M)\) is

\[
LP_m = \frac{1}{2m} \sum_{j=1}^{n} w_j \left( \left( \sum_{i=1}^{j} p_i \right)^2 - \left( \sum_{i=1}^{j-1} p_i \right)^2 \right) + \sum_{j=1}^{n} \frac{w_j p_j}{2}
\]

\[
= \frac{1}{m} \left( LP_1 - \sum_{j=1}^{n} \frac{w_j p_j}{2} \right) + \sum_{j=1}^{n} \frac{w_j p_j}{2}
\]

\[
= \frac{1}{m} LP_1 + \frac{m - 1}{2m} \sum_{j=1}^{n} w_j p_j
\]

\[
= LB^{EEI},
\]

where \( LB^{EEI} \) denotes the combinatorial lower bound due to Eastman et al. [10]. Therefore, we have proved:

**Theorem 3.5.** \( LP_m = LB^{EEI} \).

Schulz and Skutella [24] (and Chudak [7]) considered a preemptive time-indexed formulation for \( P || \sum w_j C_j \) (similar to the \( y_{jt} \)-formulation (1)–(5)) and showed that the resulting linear program is a \( \frac{3}{2} \)-relaxation. Recall that Webster [31] gave a series of progressively stronger lower bounds based on ideas similar to job splitting of [2], the weakest of which was equal to the EEI lower bound. In a manner analogous to the proof of Theorem 3.1 we can prove that the lower bound obtained through Schulz and Skutella’s LP-relaxation is equal to the weakest of Webster’s lower bounds. Therefore, we obtain a worst-case upper bound on the performance of all of Webster’s lower bounds.

Finally, for \( P|r_j| \sum w_j C_j \), Schulz and Skutella [24] considered a preemptive time-indexed formulation (equivalent to the one they considered for \( P || \sum w_j C_j \)). They showed that the resulting linear program is a 2-relaxation. It can be proved that the solution to this LP-relaxation is equal to a combinatorial lower bound for \( P|r_j| \sum w_j C_j \). Although no one has explicitly considered a combinatorial lower bound for \( P|r_j| \sum w_j C_j \), one can be constructed in a straightforward manner by combining the job splitting ideas of Webster [31] (which is for parallel machines without release dates) and the job splitting ideas of Belouadah et al. [2] (which is for single machine with release dates). We omit the details as they involve no new ideas.
Acknowledgments

We are grateful to Jan Karel Lenstra, Maurice Queyranne, David Shmoys and Chris Potts for helpful discussions. The second author would also like to thank Andreas Schulz for inviting him to ISMP’97, where the idea for this paper arose in discussions with Chris Potts. Finally, the authors also thank the anonymous referees for their thorough review which improved the presentation.

References