INFORMS Journal on Computing
Vol. 17, No. 1, Winter 2005, pp. 123–136
ISSN 0899-1499 | EISSN 1526-5528 | 05 | 1701 | 0123

DOI 10.1287/ijoc.1030.0055

An Experimental Study of LP-Based Approximation Algorithms for Scheduling Problems

Martin W. P. Savelsbergh
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0205, USA, mwps@isye.gatech.edu

R. N. Uma
Department of Computer Science, University of Texas at Dallas, M/S EC 31, Richardson, Texas 75083-0688, USA, rnuma@utdallas.edu

Joel Wein
Department of Computer Science, Polytechnic University, Brooklyn, New York 11201, USA, wein@mem.poly.edu

Recently there has been much progress on the design of approximation algorithms for a variety of scheduling problems in which the goal is to minimize the average weighted completion time of the jobs scheduled. Many of these approximation algorithms have been inspired by polyhedral formulations of the scheduling problems and their use in computing optimal solutions to small instances.

In this paper we demonstrate that the progress in the design and analysis of approximation algorithms for these problems also yields techniques with improved computational efficacy. Specifically, we give a comprehensive experimental study of a number of these approximation algorithms for $1|r_j|\sum w_jC_j$, the problem of scheduling jobs with release dates on one machine so as to minimize the average weighted completion time of the jobs scheduled. We study both the quality of lower bounds given for this problem by different linear-programming relaxations and combinatorial relaxations, and the quality of upper bounds delivered by a number of approximation algorithms based on them. The best algorithms, on almost all instances, come within a few percent of the optimal average weighted completion time. Furthermore, we show that this can usually be achieved with $O(n \log n)$ computation.

In addition we observe that on most kinds of synthetic data used in experimental studies a simple greedy heuristic, used in successful combinatorial branch-and-bound algorithms for the problem, outperforms (on average) all of the LP-based heuristics. We identify, however, other classes of problems on which the LP-based heuristics are superior and report on experiments that give a qualitative sense of the range of dominance of each. We consider the impact of local improvement on the solutions as well.

We also consider the performance of the algorithms for the average weighted flow-time criterion, which, although equivalent to average weighted completion time at optimality, is provably much harder to approximate. Nonetheless, we demonstrate that for most instances we consider that the algorithms give very good results for this criterion as well.

Finally, we extend the techniques to a rather different and more complex problem that arises from an actual manufacturing application: resource-constrained project scheduling. In this setting as well, the techniques yield algorithms with improved performance; we give the best-known solutions for a set of instances provided by BASF AG, Germany.

Key words: single-machine scheduling; linear programming; approximation algorithms
History: Accepted by Jan Karel Lenstra, Area Editor; received August 2002; revised August 2003; accepted August 2003.

1. Introduction

In this paper we experimentally evaluate the strengths of both combinatorial and linear-programming relaxations for the single-machine scheduling problem, $1|r_j|\sum w_jC_j$. We also evaluate the empirical performance of the approximation algorithms based on the linear-programming relaxations.

Over the last fifteen years the single-machine scheduling problem, $1|r_j|\sum w_jC_j$, has received significant attention from the enumerative and polyhedral community: Bianco and Ricciardelli (1982), Hariri and Potts (1983), Belouadah et al. (1992), Dyer and Wolsey (1990), De Sousa and Wolsey (1992), Wolsey (1985), Queyranne (1993), Goemans (1996), Van den Akker (1994), Van den Akker et al. (1999, 2000). Recently work on polyhedral formulations of scheduling problems has inspired a number of results on approximation algorithms. One interesting consequence of this work was that a heuristic that yielded excellent performance in empirical experiments was

established to be a small-constant-factor approximation algorithm for $1|r_j|\sum w_jC_j$ (Hall et al. 1997). Thus results from the enumerative "community" have had a direct impact on the approximation community. Subsequently, several papers appeared that gave more ingenious variants of the algorithms in Hall et al. (1997) with improved performance guarantees (Goemans 1997, Chekuri et al. 2000, Schulz and Skutella 1997a, Wang 1996, Goemans et al. 2002). In this paper, we demonstrate experimentally that these improved approximation algorithms yield improved empirical performance as well. We also generalize the techniques to yield improved algorithms for a more complex scheduling problem that arises in an actual manufacturing application from BASF AG, Germany. We feel that the experimental results in this paper, taken in conjunction with a number of previous papers on both enumerative and approximation approaches, represent an example of good synergy between theory, experimentation, and practice.

In this paper, we demonstrate the impact that progress in the approximation approach can have on enumerative/computational approaches. We show that the ideas that lead to improved approximation algorithms also lead to heuristics that are quite effective in empirical experiments. Furthermore we show that they can be extended to give improved heuristics for more complex problems that arise in practice.

Specifically, we study the performance of a suite of different algorithms based both on the C_j-relaxation and on the x_{jt}-relaxation. We demonstrate that, although the x_{jt} formulation is known to provide stronger lower bounds, for most of the instances we considered, the C_j formulation, at a significantly less computational cost, provides both a lower bound (through the solution of its relaxation) and an upper bound (through associated heuristics) that are on average just a few percent from those given by the time-indexed formulation. Furthermore, the best algorithms give very high-quality upper bounds for average weighted completion time; the ratio of the computed upper bound to the lower bound almost always comes within a few percent of optimal.

In parallel with work on linear-programming lower bounds for $1|r_j|\sum w_jC_j$ there has been significant work on branch-and-bound algorithms for $1|r_j|\sum w_jC_j$ based on combinatorial lower bounds (Belouadah et al. 1992, Bianco and Ricciardelli 1982, Dessouky and Deogun 1981, Hariri and Potts 1983). The most successful of these is due to Belouadah et al. (1992) who made use of two combinatorial lower bounds based on *job splitting* (Posner 1985, Belouadah 1985, Belouadah et al. 1992), and an upper bound based on a simple greedy heuristic. We evaluate their lower bounds and the simple greedy heuristic. We also apply the LP-based heuristics to their combinatorial relaxations and study the quality of their performance.

Another optimality criterion that is closely related to average weighted completion time is the average weighted flow time, $\sum w_j(C_j - r_j)$ (often denoted $\sum w_jF_j$), which in many settings is a much better criterion of good average service. These two criteria are equivalent at optimality, but from the perspective of approximation they are very different there is no ρ-approximation algorithm for the nonpreemptive minimization of average flow time of jobs with release dates on one machine with $\rho = o(\sqrt{n})$ unless $P = NP$ (Kellerer et al. 1999). Nonetheless, we demonstrate that for many of the instances that we considered, the approximation algorithms perform very well for the average weighted flow-time criterion as well. We also identify, however, several categories of problem instances for which the performance of these techniques degrades dramatically for the average weighted flow-time criterion.

We also study a number of more detailed issues about the performance of the algorithms, including the impact of the quality of the linear-programming relaxation to which they are applied and the power of randomization. In addition, we note that simple local-improvement techniques are often very successful in giving good solutions to scheduling problems (Anderson et al. 1997). We therefore consider the impact of some simple local-improvement techniques when applied both "from scratch" and to the solutions yielded by the various heuristics that we consider.

Finally, we show that the ideas behind the more sophisticated algorithms lead to improved heuristics for a rather different sort of scheduling problem that arises in practice. Specifically, we consider a problem and data arising from a manufacturing application from BASF AG, Germany, and give the best known solutions for several specific instances of this problem.

This last result highlights an important aspect of our study. Although $1|r_j|\sum w_jC_j$ is a natural and much-studied problem, instances of scheduling problems that are exactly of this form are rare in practice. The rationale for its importance is captured by the following quote due to Dyer and Wolsey (1990, p. 255).

> This is not an end in itself, but it is we believe one of the inherently difficult single-machine problems for which it is a challenge to obtain strong lower bounds. We hope that ultimately this approach will allow us to tackle and solve problems including many machines and other types of constraints including deadlines, precedence constraints and order dependent processing times.

In the development of approximation algorithms their hope has been borne out, as techniques developed for the one-machine problem led to the design of

approximation algorithms for a wide variety of problems (Hall et al. 1997; Chakrabarti et al. 1998; Chekuri et al. 2000; Schulz and Skutella 1997a, b; Goemans et al. 2002). We view the experimental results in this paper as further evidence towards the validation of their thesis.

Finally, we note that our goal is not an in-depth understanding of the running times of the different algorithms, but rather an understanding of the approximation performance of two different approaches with radically different running times; therefore we do not report on running times in depth in this paper. We also note that we do not claim that our best algorithms are the best known, as we have not conducted in-depth comparisons with all other approaches. Our goal is simply to understand the behavior of the above-mentioned approximation algorithms and to demonstrate the possible impact of these approaches on real problems.

2. Background: Formulations and Algorithms

2.1. Time-Indexed Formulations

A time-indexed formulation is based on time discretization. That is, time is divided into periods, where period t starts at time $t-1$ and ends at time t. The planning horizon is denoted by T and therefore we consider the time periods $1, 2, \ldots, T$. Dyer and Wolsey (1990) had proposed a number of time-indexed formulations for scheduling problems. Here we focus on two of their formulations, x_{jt}-formulation and y_{jt}-formulation. These two formulations have been studied extensively by de Sousa and Wolsey (1992), van den Akker et al. (1999, 2000), van den Akker (1994), and Queyranne and Schulz (1994) and have been useful in the design of approximation algorithms (Hall et al. 1997, Goemans 1997, Chekuri et al. 2000, Schulz and Skutella 1997a, Wang 1996, Goemans et al. 2002). We describe below these two formulations for single-machine scheduling problems.

x_{jt}-Formulation. This formulation is general in that it models several single-machine scheduling problems including $1|r_j|\sum w_j C_j$.

$$\text{minimize} \quad \sum_{j=1}^{n}\sum_{t=1}^{T} c_{jt}x_{jt} \tag{1}$$

$$\text{subject to} \quad \sum_{t=1}^{T} x_{jt} = 1, \quad j = 1, \ldots, n; \tag{2}$$

$$\sum_{j=1}^{n}\sum_{s=t}^{t+p_j-1} x_{js} \le 1, \quad t = 1, \ldots, T; \tag{3}$$

$$x_{jt} \in \{0, 1\}, \quad j = 1, \ldots, n, \quad t = 1, \ldots, T \tag{4}$$

where the binary variable x_{jt} for each job j ($j = 1, \ldots, n$) and time period $[t-1, t]$ ($t = 1, \ldots, T$) indicates whether job j completes in period $[t-1, t]$ ($x_{jt} = 1$) or not ($x_{jt} = 0$). The assignment constraints (2) state that each job has to be completed exactly once, and the capacity constraints (3) state that the machine can handle at most one job during any time period. This formulation can be used to model several single-machine scheduling problems by an appropriate choice of the objective coefficients and possibly a restriction on the set of variables. For instance, if the objective is to minimize the weighted sum of the completion times, we take coefficients $c_{jt} = w_j t$, where w_j denotes the weight of job j; if there are release dates r_j, i.e., job j becomes available at time r_j, then we discard the variables x_{jt} for $t = 1, \ldots, r_j + p_j - 1$.

The x_{jt}-LP is obtained from (1)–(4) by relaxing the integrality constraint on x_{jt} (4) to

$$0 \le x_{jt} \le 1 \qquad j = 1, \ldots, n, \quad t = 1, \ldots, T. \tag{5}$$

The x_{jt}-LP has been observed to give strong lower bounds (de Sousa and Wolsey 1992, van den Akker 1994) but is very difficult to solve due to its size. Because the number of constraints is $n + T$ and the number of variables is roughly nT with $T > \sum_j p_j$, even for instances with relatively few jobs, the size can be enormous. As a result, the memory required to store an instance and the time required to solve just the LP relaxation may be prohibitively large. Therefore, for time-indexed formulations to be useful, ways are needed to reduce the memory requirements and the solution times of the LP relaxation. van den Akker et al. (1999, 2000) show that Dantzig-Wolfe decomposition techniques and column-generation techniques can be used for partial alleviation of the difficulties associated with the size of time-indexed formulations. Even though Dantzig-Wolfe decomposition techniques and column-generation techniques allow the solution of the LP relaxation of much larger instances, there are some inherent difficulties with that approach too. Column-generation techniques tend to converge very slowly, especially on larger more difficult instances.

The x_{jt}-LP is a nonpreemptive relaxation. That is, the relaxation is obtained by slicing jobs into pieces in a horizontal fashion such that any one piece requires only a fractional capacity of the machine but the full processing requirement of the job that it belongs to. A feasible solution to the x_{jt}-LP for the instance in Table 1 is given in Figure 1. The solution in Figure 1 can be viewed as a schedule of such pieces (slices) of jobs. For example, the slice of job j corresponding to $x_{jt} = 0.25$ utilizes one fourth of the machine's capacity and is scheduled nonpreemptively from $t - p_j$ to t, for some t.

Table 1 A Problem Instance

j	r_j	p_j
1	0	10
2	4	5
3	6	2
4	7	4

y_{jt}-Formulation. Another formulation for the single-machine scheduling problem $1|r_j|\sum w_jC_j$ is:

$$\text{minimize} \quad \sum_{j=1}^{n} w_j\left(\frac{p_j}{2}+\frac{1}{p_j}\sum_{t=r_j+1}^{T}\left(t-\frac{1}{2}\right)y_{jt}\right) \tag{6}$$

$$\text{subject to} \quad \sum_{j=1}^{n} y_{jt} \le 1, \quad t=1,\dots,T; \tag{7}$$

$$\sum_{t=r_j+1}^{T} y_{jt} = p_j, \quad j=1,\dots,n; \tag{8}$$

$$y_{jt}\in\{0,1\}, \quad j=1,\dots,n, \quad t=r_j+1,\dots,T \tag{9}$$

where the binary variable y_{jt} for each job j ($j=1,\dots,n$) and time period $[t-1,t]$ ($t=1,\dots,T$) indicates whether job j is processed in period $[t-1,t]$ ($y_{jt}=1$) or not ($y_{jt}=0$). The term in the objective function, namely $p_j/2+(1/p_j)\sum_{t=r_j+1}^{T}(t-\frac{1}{2})y_{jt}$, corresponds to the actual completion time of job j if the job were continuously processed from C_j-p_j to C_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 the remaining half of the processing time $p_j/2$ to this midpoint of computation of job j gives the effective completion time of job j. This effective completion time of job j is earlier than the completion time of the last piece of job j even if the y_{jt} are integral (unless all pieces of the job are scheduled consecutively). Thus the solution to the y_{jt}-LP relaxation provides only a lower bound to the preemptive schedule. The constraints (8) require that each job should be processed in its entirety between its release date r_j and T. The capacity constraints (7) state that the machine can handle at most one job during any time period.

In a relaxation to the integer program (6)–(9), the integrality constraint (9) is relaxed to

$$0\le y_{jt}\le 1 \qquad j=1,\dots,n, \quad t=r_j+1,\dots,T \tag{10}$$

We will refer to this linear program as y_{jt}-LP. This linear program is a valid relaxation to the optimal preemptive schedule as well (Hall et al. 1997).

Although the y_{jt} formulation is of exponential size, with $n+T$ constraints and roughly nT variables where $T\ge\sum_j p_j$, Dyer and Wolsey (1990) showed that it is a transportation problem with a very special structure and thus can be solved in $O(n\log n)$ time (see also Goemans 1997). The structure of the solution is simple: at any point in time, schedule the available unfinished job with maximum w_j/p_j (this may involve preemption). Note that p_j denotes the total processing requirement of job j and not the remaining processing requirement. The y_{jt}-LP is a preemptive relaxation. A feasible solution to the y_{jt}-LP for the instance in Table 1 is given in Figure 2.

Although both formulations are exponential in size, the relaxation to the y_{jt}-formulation can be solved in polynomial time whereas the relaxation to the x_{jt}-formulation requires a pseudo-polynomial-time solution. As a result, any approximation algorithm that is based on the y_{jt}-relaxation is a polynomial-time algorithm whereas any approximation algorithm based on the x_{jt}-relaxation is only a pseudo-polynomial-time algorithm. Note however, that the x_{jt}-relaxation gives stronger lower bounds than does the y_{jt}-relaxation.

2.2. Completion-Time Formulations

A different approach to model the problem is by using variables C_j that represent the completion time of job j in a schedule. Let N be the set of all n jobs and define for any set $S\subseteq N$, $r_{\min}(S)=\min_{j\in S} r_j$ and $p(S)=\sum_{j\in S}p_j$. Further, define $p(S)^2=(\sum_{j\in S}p_j)^2$ and $p^2(S)=\sum_{j\in S}p_j^2$. We will refer to the following relaxation to $1|r_j|\sum w_jC_j$ as the C_j-relaxation:

$$\text{minimize} \quad \sum_{j=1}^{n} w_jC_j$$

$$\text{subject to} \quad \sum_{j\in S} p_jC_j \ge \ell(S), \quad \text{for each } S\subseteq N, \tag{11}$$

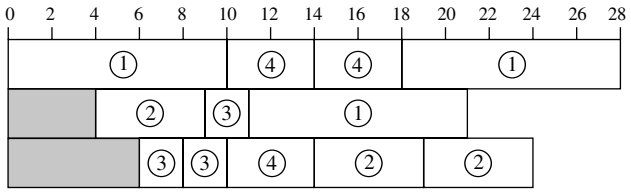

Figure 1 A Feasible Solution to the x_{jt}-LP (1–3,5) for the Instance in Table 1

Note. 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).

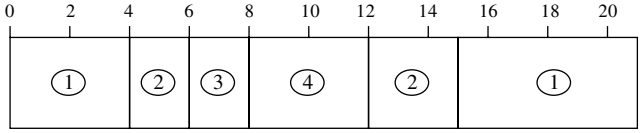

Figure 2 A Feasible Solution to the y_{jt}-LP (6–8,10) for the Instance in Table 1

Note. 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).

where

$$\ell(S) = r_{\min}(S)p(S) + \tfrac{1}{2}(p^2(S) + p(S)^2).$$

The constraints of the linear program are valid constraints for the problem but do not completely characterize the set of feasible schedules. Nevertheless, solutions to this relaxation can be used to compute near-optimal solutions. Although this relaxation has an exponential number of constraints, Goemans (1996) showed that its solution can be computed in $O(n \log n)$ time and that it is equivalent to the y_{jt}-relaxation of Dyer and Wolsey (1990).

2.3. Combinatorial Relaxations

Another method for obtaining a relaxation, as opposed to solving relaxations to integer-programming formulations, is to relax certain constraints about the structure of the schedule. This makes use of the underlying combinatorial structure of the problem in some fashion. For example, the nonpreemptive constraints in a problem could be relaxed by allowing preemptions of jobs.

For the single-machine scheduling problem $1|r_j|\sum w_jC_j$ and its variants a variety of combinatorial relaxations have been proposed including Rinaldi and Sassano (1977), Chandra (1979), Dessouky and Deogun (1981), Bianco and Ricciardelli (1982), Potts and van Wassenhove (1983), Hariri and Potts (1983), Posner (1985), Belouadah (1985), and Belouadah et al. (1992). For instance, in Chandra (1979) and Bianco and Ricciardelli (1982), they relax the problem by considering partial sequences of jobs. In Dessouky and Deogun (1981), they partially relax the release-date constraints. Posner (1985), Belouadah (1985), and Belouadah et al. (1992) propose a variant of preemptive relaxation.

We consider the combinatorial relaxations of Belouadah et al. (1992). Their relaxations are based on *job splitting* (Posner 1985, Belouadah 1985, Belouadah et al. 1992) where jobs are split into smaller pieces. This technique is based on the idea that a relaxation to a nonpreemptive scheduling problem may be obtained by splitting each job into smaller pieces that can be scheduled individually. In the case of $1|r_j|\sum w_jC_j$, when we split job j into pieces we must split its weight w_j among the pieces as well. In essence, we create a number of smaller jobs. If we split the jobs in such a way that we can solve the resulting relaxed problem in polynomial time, we obtain a polynomial-time computable lower bound on the optimal solution to the original problem. This *job-splitting* type of relaxation has been observed to give good lower bounds and has been successfully used in branch-and-bound algorithms.

Belouadah et al. (1992) give two lower bounds (to which we will refer as BPP1 and BPP2) based on job splitting. In BPP1, the pieces of job j are *exactly* those that arise in the optimal preemptive solution to the y_{jt} relaxation, and the optimal solution to the resulting split problem has the same structure as that of the optimal preemptive solution to the y_{jt}-relaxation. In this lower bound each piece of job j receives a fraction of weight w_j in exact proportion to the fraction of the size of p_j that its size is. In the BPP2 lower bound, weights are assigned greedily. For each job, as much weight as possible is shifted to later scheduled pieces of that job. The weight assignment should be such that the optimal solution of the split instance can be computed in polynomial time.

The lower bounds achieved by this process, BPP1 and BPP2, are both bounded above by the solution to the optimal preemptive schedule. However, the preemptive version of $1|r_j|\sum w_jC_j$ is also NP-hard and hence we must settle for solving something weaker that can be computed in polynomial time.

A result of Uma et al. (2003) it was proven that the BPP1 lower bound is equal to the solution to the y_{jt}-relaxation and also that neither the solution to the x_{jt}-relaxation nor the BPP2 relaxation dominates the other.

Therefore, of the five lower bounds considered—x_{jt}, y_{jt}, C_j-based, BPP1, and BPP2—three (y_{jt}, C_j-based, and BPP1) are identical. Hence, in this paper, we empirically evaluate the strength of the three distinct lower bounds—x_{jt}, y_{jt}, and BPP2.

2.4. Approximation Algorithms and Heuristics

In this section, we present the various approximation algorithms for $1|r_j|\sum w_jC_j$. Progress on approximation algorithms for $1|r_j|\sum w_jC_j$ began with the special case of $w_j = 1$ for all j, for which Phillips et al. (1998) introduced the idea of taking a relaxation of the problem and inferring an ordering from that relaxation. Progress on the case of general w_j arose from the use of one of the two linear programs discussed earlier as a relaxation from which to infer an ordering. Since the solutions to either relaxation correspond to fractional solutions to a time-indexed formulation (either x_{jt} or y_{jt}-based), we wish to find a way to capture from the fractional assignment of jobs to points in time a measure of where the important part of a job is scheduled.

Two basic ideas have been introduced. The first is to use the values suggested by the linear program. For example, let the solution to the C_j-based relaxation (which is equal to the solution to the y_{jt}-relaxation) be $\overline{C}_j$, $j = 1, \ldots, n$. We sort the jobs by nondecreasing $\overline{C}_j$ and schedule in that order, respecting release-date constraints (Schulz 1996, Hall et al. 1997). In the case of the x_{jt}-based formulation, we again order the jobs by their suggested completion times $\sum_{t=p_j}^{T} t x_{jt}$. This algorithm, applied to either relaxation, is known to be a 3-approximation algorithm (Hall et al. 1997) and

we will refer to it as SCHEDULE-BY-$\overline{C}_j$; when applied to the solution of x_{jt}-based formulations, it was demonstrated empirically to be very effective in practice (van den Akker et al. 2000).

A different approach to constructing an ordering is to make use of the notion of an α-point (Phillips et al. 1998, Hall et al. 1996). The α-point of job j, $0 \le \alpha \le 1$, is defined to be the first point in time, in the solution to a time-indexed relaxation, at which an α fraction of job j has been completed. We define the algorithm SCHEDULE-BY-FIXED-α, that can be applied to the solution of either relaxation, as ordering the jobs by their α-points and scheduling in that order. Goemans (1997) has shown that for $\alpha = 1/\sqrt{(2)}$ this is a $(\sqrt{2}+1)$-approximation algorithm. Goemans also showed that by choosing α randomly according to a uniform distribution and then scheduling in this order, one obtains a randomized 2-approximation algorithm (Goemans 1997) and if one chooses using a different distribution, a 1.7451-approximation algorithm (Goemans 1997). Either randomized algorithm can be derandomized by considering n different values of α, scheduling according to each of them, and then choosing the best. We call these two algorithms SCHEDULE-BY-RANDOM-α and SCHEDULE-BY-BEST-α. In our experiments, for SCHEDULE-BY-RANDOM-α, α is chosen randomly from a uniform distribution. Finally, Schulz and Skutella (1997a) introduced the idea of randomly choosing n values of α, one for each job, and ordering the jobs according to these α-points (Schulz and Skutella 1997a); this has been proven to be a 1.6853-approximation algorithm (Goemans et al. 2002). We call this algorithm SCHEDULE-BY-RANDOM-α_j.

Finally we also consider the simple greedy heuristic used in Belouadah et al. (1992) in their branch-and-bound code to compute the upper bound. This heuristic, SWPT (shortest weighted processing time), selects for processing when the machine is idle the unprocessed available job with the smallest p_j/w_j ratio and schedules it nonpreemptively. It is trivial to see that the worst-case performance of this heuristic is unbounded. For example, consider the following scenario with two jobs: Job 1 is released at time 0, has weight 1, and processing time p. Job 2 is released at time 1, has weight w, and processing time 1. For sufficiently large values of w and p, SWPT will schedule job 1 followed by job 2, whereas optimal will schedule job 2 followed by job 1. As w and p tend to infinity, the cost of the SWPT schedule can be made arbitrarily large compared to the optimal.

3. Experiments

In this experimental study our main goal is to understand the quality of the different relaxations and the quality of the upper bounds they deliver and to determine if these techniques can be extended to real domains. As a result, we investigate the following issues:

- Do the ideas that led to a series of successively improved approximation algorithms have a similar impact in practice as well? We would particularly like to understand the quality of lower bounds delivered by the relaxations and to understand the quality of performance of the approximation algorithms and heuristics.
- To what extent can one achieve speed-accuracy trade-offs using randomization and sampling?
- How much influence does the relaxation have on the quality of the schedule that is built from it?
- How does a very simple heuristic compare to the sophisticated approximation algorithms?
- How effective are local-improvement techniques?
- Finally, do these sophisticated techniques have any practical significance? Can we extend these techniques to solve hard-to-solve problems that arise in the real world?

We therefore utilized a number of sets of instances, designed with the hope of providing a rich test set on which to observe different behaviors.

3.1. Experimental Design

We consider three categories of data that we call *Optimal*, *Synthetic*, and *Hard*. The *Optimal* set is a set of 60 instances with $n = 30$ jobs; for all of these instances we know the exact optimal solutions, computed by the branch-and-cut code of van den Akker et al. (1999); these instances represent the limits of what they were able to solve optimally with their code. For all these instances, the release dates were generated uniformly in $[0, \frac{1}{2}\sum_{j=1}^{n} p_j]$ and w_j was generated uniformly in $[1, 10]$. Twenty instances had p_j generated uniformly in $[1, 5]$ and forty in $[1, 10]$.

The *Synthetic* set is a large collection of instances generated according to four parameters: Number of jobs (n), arrival rate (a), distribution for the random generation of the weights w_j, and distribution for the random generation of the processing times p_j. The number of jobs (in each instance) was chosen from $\{50, 100, 200, 500\}$. The release dates r_j were generated by a Poisson process to model that on average a jobs arrived every $p_{\max}$ units of time where $a \in \{2, 5, 10, 20\}$ and $p_{\max} = 10$. Three types of distributions were used for the random generation of w_j and p_j—(i) uniform in the range $[1, 10]$, (ii) a normal distribution $\mathcal{N}(5.0, 2.5)$ (where $\mathcal{N}(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ), and (iii) a bimodal distribution where with probability 0.5 we chose from a normal distribution $\mathcal{N}(2.5, 0.5)$ and with probability 0.5 we chose from a normal distribution $\mathcal{N}(7.5, 0.5)$. For each combination of parameters 10 instances were generated randomly for a total of 1440 instances.

For all of these synthetic instances we solved the associated C_j-based relaxation (and also the BPP2 relaxation). We solved the x_{jt}-based relaxation, using the column-generation code, for as many instances as feasible. After a significant amount of computation time, we solved this relaxation exactly for all of the $n = 50$ instances and the set of $n = 100$ instances with arrival rates 2 and 5; for $n = 100$ we cut off the column generation somewhat early and therefore only produced approximate solutions. This demonstrates an obvious and major advantage of the C_j-based LP-relaxations over the x_{jt}: They are solvable efficiently.

The *Hard* set was designed to provoke poor performance from the LP-based heuristics. All of these instances attempt to exploit the fact that the y_{jt} relaxation is also a valid relaxation of the optimal preemptive schedule (Hall et al. 1997), and that therefore on instances for which the optimal preemptive schedule is much better than the optimal nonpreemptive schedule, the y_{jt}-based relaxation should perform poorly. Therefore these instances have one or several very large jobs, and a large number of tiny jobs that are released regularly at small intervals. For the *Hard* set we created instances with processing times 1 and 20 or 1 and 30. The size 1 jobs were generated on average 9 times more frequently than the size 20 or size 30 jobs and $n \in \{50, 100\}$.

In this paper we focus on subsets of the data sets that turn out to be among the most difficult in their categories for the heuristics and that are at the same time representative of the overall behavior of that set. Specifically, we focus on the *Optimal*, *Synthetic* 1 and *Synthetic* 2, and *Hard* data sets. The *Synthetic* 1 set corresponds to the subset of the *Synthetic* set where $n \in \{50, 100\}$ and $a \in \{2, 5\}$ and the *Synthetic* 2 data set corresponds to a variation of the *Synthetic* set with release dates being generated uniformly and $n \in \{50, 100, 200, 500\}$.

On each solution to either linear program we ran a suite of algorithms, including SCHEDULE-BY-$\overline{C}_j$, SCHEDULE-BY-FIXED-α (for $\alpha = 1/\sqrt{(2)}$), SCHEDULE-BY-BEST-α, SCHEDULE-BY-RANDOM-α (based on the uniform distribution), and SCHEDULE-BY-RANDOM-α_j.

3.2. Experimental Results

3.2.1. Relative Strengths of Lower and Upper Bounds. We now look more carefully at the performance of the algorithms, first those based on the C_j relaxations. With respect to the $\sum w_j C_j$ optimality criterion the two basic algorithms (SCHEDULE-BY-$\overline{C}_j$ and SCHEDULE-BY-FIXED-α) give solutions that are almost always within 9.5% of optimal, and SCHEDULE-BY-BEST-α is almost always within 2% of optimal, with a maximum observed performance of a factor of 1.037 times optimal. As this suggests, in general it is easy to approximate the $\sum w_j C_j$ criterion within a few percent of optimal, unless the instances are constructed in a very careful fashion. Therefore we focus for the rest of the paper on the $\sum w_j F_j$ criterion. Note that the relative error with respect to average weighted flow time is an upper bound on the relative error with respect to average weighted completion time.

Table 2 **Quality of the Lower Bounds with Respect to the Weighted Flow Time Given by y_{jt}, BPP2, and x_{jt} Relaxations**

	Optimal	Hard	Synthetic 1	Synthetic 2
y_{jt}(BPP1)	4.506	19.169	1.729	1.848
BPP2	2.969	15.539	0.941	0.000
x_{jt}	1.322	0.000	0.013	N/A

Note. We report on (BEST − LB)/LB × 100, where BEST is the best available lower bound and LB is the corresponding lower bound. The values reported are averaged over all the instances in each case.

We next examine the quality of the lower bound delivered by each of the relaxations. Table 2 reports on the relative performance of the different lower bounds. We note that the BPP2 lower bound does provide some improvement over the y_{jt} bound at modest additional computational cost ($O(n \log n)$ to $O(n^2)$), but because both are relaxations of the optimal preemptive schedule, on the *Hard* data set the BPP2 bound is still far from the x_{jt}-lower bound. Furthermore, note that on the *Optimal* and *Synthetic* 1 data sets sometimes BPP2 is better than the x_{jt}-relaxation, and this explains why none of the numbers in those columns is 0—the comparison is always with respect to the best lower bound for that instance. We note that the maximum improvement observed by BPP2 over y_{jt} on any instance was 9.492%, on an instance in the *Hard* data set.

We next study the performance of the upper-bounding techniques (both approximation algorithms and heuristics) on the different data sets.

Optimal and Synthetic Sets. For the instances in the *Synthetic* and *Hard* data sets we do not have the optimal solutions (as $n = 50$ was beyond the range of problems solvable optimally by the branch-and-cut code of van den Akker et al. (1999). However, because the algorithms will solve the linear-programming relaxations, yielding a lower bound, and then construct a schedule based on this relaxation, yielding an upper bound, we report as an upper bound on performance the ratio of the upper bound to the lower bound. We report these ratios with respect to the x_{jt}-based relaxation if we know its solution and otherwise with respect to the C_j-based relaxation. The *Optimal* and *Synthetic* data sets give similar results qualitatively, both with respect to the general size of factors of approximation achieved by the algorithms and their relative performance. Here we present results for the *Synthetic* 1 data set.

Table 3 Performance of Algorithms Applied to Solution of C_j-Relaxation for $\sum w_j F_j$

	Schedule-by-$\overline{C}_j$			Schedule-by-Fixed-α			Schedule-by-Best-α		
(n, a)	Mean	Std. dev.	Max	Mean	Std. dev.	Max	Mean	Std. dev.	Max
(50, 2)	1.271	0.144	1.729	1.184	0.092	1.495	1.066	0.047	1.282
(50, 5)	1.107	0.045	1.245	1.073	0.032	1.197	1.018	0.013	1.071
(100, 2)	1.268	0.125	1.786	1.190	0.096	1.686	1.071	0.043	1.271
(100, 5)	1.079	0.030	1.152	1.056	0.018	1.104	1.014	0.010	1.055

Note. We report on ratio of algorithm performance to x_{jt}-relaxation lower bound.

Although we considered nine combinations of distributions for generating the processing times and weights (three possibilities for each), in the hope of generating different sorts of behaviors, there were not significant qualitative differences in the performance on different distributions. Therefore, in reporting results we group into one set all (w_j, p_j) distribution combinations for a particular (n, a) pair.

Despite the worst-case results on the difficulty of the $\sum w_j F_j$ objective, the performance of these algorithms is quite reasonable, as illustrated in Table 3. The basic algorithms are observed performing as badly as 1.786 times optimal, but Schedule-by-Best-α is observed to perform only as badly as 1.282 times optimal. We also note that on average and in its maximum observed values Schedule-by-Fixed-α is better than Schedule-by-$\overline{C}_j$, and Schedule-by-Best-α gives significant improvement over both. However, this need not be true on every instance—on about 15% (over the entire *Synthetic* data set, which is the superset of the *Synthetic* 1 data set) Schedule-by-$\overline{C}_j$ gave a better solution than Schedule-by-Fixed-α (and on two of the 1440 instances it was better than Schedule-by-Best-α). Note that the ordering of average performance corresponds to the respective quality of their current worst-case performance guarantees, which are, respectively, 3 (Hall et al. 1997), $(\sqrt{2}+1)$ (Goemans 1997) and 1.745 (Goemans et al. 2002). Thus the ideas that yielded improved worst-case analysis yield significantly improved empirical performance.

To gain a sense of the distribution of the various performance factors achieved by these algorithms, we constructed a histogram of this information in Figure 3.

We now compare this performance to that of the algorithms applied to the solution of the x_{jt}-based relaxations. In Table 4 we observe that the C_j-based algorithms give performance very close to that of the x_{jt} algorithms, at greatly reduced computational cost; the best x_{jt}-algorithms are on average less than 2% better than the best C_j-based algorithms. We conclude that on data of this sort the C_j-based relaxations and algorithms deliver surprisingly strong experimental performance.

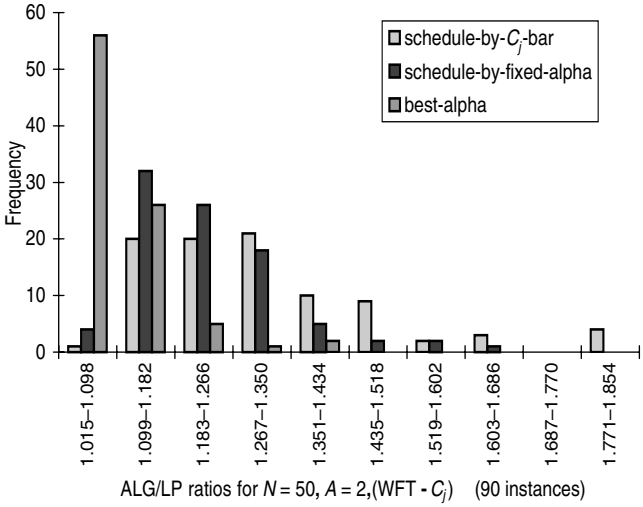

Figure 3 Distribution of Ratios of ALG/LP

Note. ALG is the cost of the solution obtained from running the corresponding algorithm on the C_j-based relaxation and LP is the solution to the C_j-relaxation with respect to $\sum w_j F_j$ criterion for the case $(n, a) = (50, 2)$ (a total of 90 instances).

Hard Sets. Next we discuss some hard data sets, on which these techniques were expected to exhibit lower-quality performance. All of these instances attempt to exploit the fact that the C_j-based relaxation is also a valid relaxation of the optimal preemptive schedule (Hall et al. 1997), and that therefore on instances for which the optimal preemptive schedule is much better than the optimal nonpreemptive schedule, the C_j-based relaxation should perform poorly.

Motivated by this and by Kellerer et al. (1999) we constructed instances with one or several very large

Table 4 Comparison of the x_{jt}-Based Algorithms to the C_j-Based Algorithms for $\sum w_j F_j$

	$N = 50$		$N = 100$	
	Mean	Std. dev.	Mean	Std. dev.
Schedule-by-$\overline{C}_j$	0.966	0.076	0.991	0.060
Schedule-by-Fixed-α	0.993	0.074	1.011	0.057
Schedule-by-Best-α	0.982	0.032	0.995	0.025

Note. The values given are the mean and standard deviation of the ratio of the performance of the x_{jt}-based algorithm to the C_j-based algorithm (averaged over 180 instances) in the *synthetic* set.

jobs, and a large number of tiny jobs that are released regularly at small intervals. For such an instance, its average flow time with preemption allowed is much smaller than its nonpreemptive average flow time because with preemption, each tiny job can be snuck in preemptively without overly disturbing the processing of the large job. We would expect the x_{jt}-formulations to give stronger lower bounds, as it is only a relaxation of the nonpreemptive problem, but, with respect to average weighted flow time they will still be weak on such instances because the small jobs can be snuck in in a fractional sense.

To generate difficult instances of this form we need processing times in a larger range than $[1, 10]$ and thus we experimented with instances with p_j generated in $[1, 100]$. We generated job sizes in a bimodal distribution, with the modes at 5 and 85; the size 5 jobs were generated on average 9 times more frequently. On this set of instances, which we investigated for $n \in \{50, 100, 200, 500\}$, the worst performance (upper bound/lower bound ratio) recorded over all algorithms exceeds 4 and the average performance measure of the best algorithm approaches 2. We note that the algorithms are actually doing better than this because the C_j-based lower bound to which we compare is weak.

Based on this experiment, we experimented with instances with a very large spread—for example, jobs of size 1 and 200. On one such instance we recorded a ratio for Schedule-by-Best-α of 16.502, which was in part due to the weak lower bound. However, the actual approximation factor, when the instance was solved optimally, was still 7.110. In this instance we have one large job released at time 0 and a sequence of small jobs that arrive periodically over time. In the optimal schedule the large job should be processed last. However, in the C_j-based relaxation it is completed, preemptively, much earlier, and therefore for all α its α-point is earlier than a large number of small jobs. As a result, many small jobs are delayed unduly by all of our algorithms. We note that these particular instances were very similar in spirit to the instances given by Queyranne (1993) and Wang (1996) that demonstrate a lower bound of $e/(e-1)$ on the worst-case ratio of (Optimal Schedule value)/(Value of C_j-based relaxation).

Unfortunately, instances with $p_j \in [1, 100]$ are well beyond our ability to solve the x_{jt} time-indexed relaxation. Therefore, in an attempt to understand the relative performance of the lower bound given by the x_{jt}-relaxation for such hard instances we created the *Hard* data set with $p_j \in \{1, 20\}$ and $\{1, 30\}$ (with processing times 1 and 20 or 30). For these we could solve the x_{jt}-relaxation except for the 100 job instances with $p_j \in \{1, 30\}$. On this set, on average the x_{jt}-relaxation is about 19% stronger than the C_j-based relaxation. The Schedule-by-Best-α algorithm is observed to perform as badly as 2.428 times optimal and incidentally, this performance was obtained when scheduled using the stronger x_{jt}-relaxation than the weaker y_{jt}-relaxation. Later we will discuss more about the impact of the quality of the underlying relaxation to the algorithms' performance.

Conclusions. To conclude, the observed quality of the lower bounds directly correspond to the time spent on computing them, with y_{jt}-relaxation giving the weakest bounds and x_{jt}-relaxation giving the strongest bounds. (Hence, if the quality of the lower bounds is not highly critical to the end application, then using y_{jt}-relaxation is better than using x_{jt}-relaxation because of the additional time it takes to solve the x_{jt}-relaxation.) Likewise, there is a direct correspondence of the quality of the schedule given by an approximation algorithm with its performance guarantee. In general, Schedule-by-Best-α outperforms all the other approximation algorithms.

3.2.2. Randomness and Sampling. On most instances, we see that the Schedule-by-Best-α algorithm yields significant improvement over the other algorithms. However, it requires extra computation, as one must check n different values of α, and thus is an $O(n^2)$ algorithm. Note that Schedule-by-Best-α actually arose as a derandomization of the Schedule-by-Random-α algorithm (Goemans 1997, Chekuri et al. 2000). To what extent does randomization afford the performance guarantees of Schedule-by-Best-α at a reduced computational cost? Qualitatively we summarize that both Schedule-by-Random-α and Schedule-by-Random-α_j, on average, give performance better than Schedule-by-$\overline{C}_j$ (although occasionally they can do worse) but they do not do nearly as well as Schedule-by-Best-α.

It is, however, a natural idea to run the randomized algorithm several times and choose the best answer it gives. In a similar spirit, perhaps we need not check all α, as does Schedule-by-Best-α; maybe checking just a few will be sufficient. Thus, we introduce three additional algorithms. Schedule-by-k-Best-α divides $[0, 1]$ into k equal subintervals and tries each of the resulting endpoints as candidate α, returning the best answer. Goemans (1997) has shown that this is a $(2 + 1/k)$-approximation algorithm. Schedule-by-k-Random-α and Schedule-by-k-Random-α_j run the respective randomized algorithms k times.

We conclude that it is necessary to test relatively few values of α to achieve performance very close to Schedule-by-Best-α. For example, in the *Synthetic* data set Schedule-by-k-Best-α, with $k = 5$, on average is less than one percent from Schedule-by-Best-α, with standard deviation 1.2%; on 48% of instances it achieved exactly the same bound. Even on the *Hard*

data set, with $k = 10$ we observed performance within 3% to 4% of SCHEDULE-BY-BEST-α.

As a general qualitative rule, the randomized algorithms that made k random choices were somewhat worse than the algorithm that chose k values of α deterministically for $k \leq n$. However, SCHEDULE-BY-k-RANDOM-α_j, for k large enough, yields on average moderate performance improvements. For example on the *Synthetic* set with $(n, a) = (50, 2)$, it on average improved on SCHEDULE-BY-BEST-α around $k = 50$ and improved gradually until $k = 200$ or so. The overall performance gain was approximately 3%. For some of the hardest problems significant improvements (20% to 30%) were seen by running SCHEDULE-BY-k-RANDOM-α_j a large number of times.

3.2.3. Algorithm Performance vs. Quality of Linear-Program Solution. Overall, our results suggest that the higher the quality of the LP relaxation solution, the better the algorithms perform. This is true across all the data sets, when we compare the performance of the algorithms based on the BPP2 relaxation and the y_{jt}-relaxation, as is evident from Table 5. Recall that both BPP2 and y_{jt} are preemptive relaxations and BPP2 is stronger than y_{jt}. But this is not always the case when we consider the x_{jt}-relaxation also. For example, on the $(50, 5)$ *Synthetic* data SCHEDULE-BY-FIXED-α does on average slightly worse when it is based on the x_{jt}-relaxation, and on the *Hard* instances for which we had solutions to both relaxations the algorithms based on the C_j relaxation were also better on average. We also explored this relationship from a different perspective. Specifically, recall that a branch-and-cut code for the solution of an integer program produces a sequence of solutions to the linear-programming relaxation that are of increasing quality (i.e. increasingly close to the optimal integer solution). For 10 instances we have generated the sequence of solutions to the time-indexed linear-programming relaxation produced by the branch-and-cut code at the root node, and run the algorithms on them. Somewhat surprisingly, the algorithms' performance does not monotonically improve with improved LP solution. For instance, see Figure 4 where we have shown the plots for one of the 10 instances. It is an interesting question to understand this phenomenon better and its implications. If the goal is only approximately optimal performance, these results suggest that there may be little benefit in doing the most intensive computations.

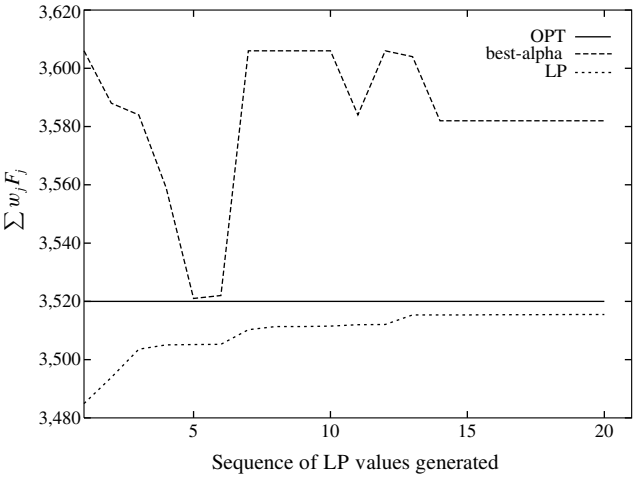

Figure 4 **Performance of the SCHEDULE-BY-BEST-α Heuristic with Successive LP's That were Generated Compared with the Optimal Value**

3.2.4. SWPT vs. LP-based Heuristics. Of greater interest is to compare the performance of all this LP "stuff" to the simple and naive heuristic used by Belouadah et al. (1992) in their combinatorial branch-and-bound code. This heuristic, SWPT, simply, when idle, selects for processing the unprocessed available job with the smallest p_j/w_j ratio (or equivalently the largest w_j/p_j ratio) and schedules it nonpreemptively. It is trivial to see that the worst-case performance of this heuristic is unbounded. Table 5 provides a summary of the performance of the various heuristics.

Table 5 **Performance of the Algorithms Based on the Three (y_{jt}, BPP2, x_{jt}) Formulations**

	Hard			Synthetic 1			Synthetic 2		
	Mean	Std. dev.	Max	Mean	Std. dev.	Max	Mean	Std. dev.	Max
SCHEDULE-BY-FIXED-α (y_{jt})	2.207	0.623	3.906	1.126	0.093	1.686	1.334	0.126	1.892
SCHEDULE-BY-BEST-α (y_{jt})	1.683	0.263	2.164	1.042	0.042	1.282	1.155	0.067	1.371
SCHEDULE-BY-5-BEST-α (y_{jt})	1.775	0.301	2.244	1.048	0.046	1.358	1.171	0.079	1.419
SCHEDULE-BY-FIXED-α (BPP2)	1.985	0.507	3.902	1.116	0.084	1.643	1.292	0.109	1.615
SCHEDULE-BY-BEST-α (BPP2)	1.628	0.245	2.122	1.040	0.039	1.233	1.144	0.059	1.311
SCHEDULE-BY-5-BEST-α (BPP2)	1.710	0.274	2.296	1.046	0.043	1.233	1.157	0.067	1.356
SCHEDULE-BY-BEST-α (x_{jt})	1.854	0.320	2.428	1.048	0.045	1.303	—	—	—
SWPT	1.874	0.372	3.015	1.033	0.032	1.148	1.121	0.048	1.307

Note. We report the mean, standard deviation, and maximum of the ratio of the performance of the algorithm to the best lower bound.

It is striking that except on the *Hard* data set the SWPT heuristic is superior to all the LP relaxation-based approaches. Across the entire *Synthetic* data sets SWPT yields a better solution than do the LP-approaches 70% to 80% of the time. On specific instances of the *Hard* data set it is up to 27% better than the best performance of the other heuristics.

In some sense this performance is quite surprising, but in another it is not very surprising at all. SWPT, while lacking any worst-case performance guarantee, is a natural choice that should work well "most of the time." It is likely that when one generates synthetic data one is generating instances that one would think of "much of the time." It is on the harder sorts of instances that the potential problems with such a heuristic arise.

Although it would not be difficult to generate instances that would yield horrendous performance for SWPT, the purpose of an experimental study is to yield insight into algorithm performance on instances that may be somewhat natural. To this end we generated a spectrum of problems in the framework of "a few jobs of size P and lots of jobs of size 1," with P ranging from 1 to 1,000. The results of the experiment, plotted in Figure 5, gives a qualitative sense of the range in which SWPT is competitive with algorithms with worst-case performance guarantee, which is essentially up to $P = 100$ or so.

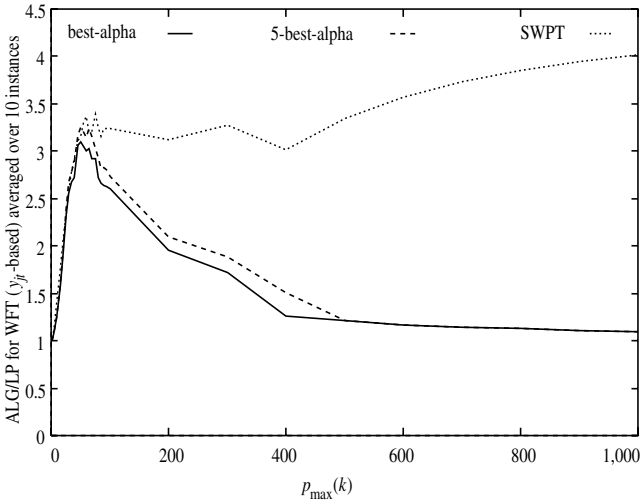

Figure 5 Plot of ALG/LP Ratios

Note. ALG is the cost of the solution obtained from running the corresponding algorithm on the C_j-based relaxation, if applicable, and LP is the solution to the C_j-relaxation for $\sum w_j F_j$ for SCHEDULE-BY-BEST-α, SCHEDULE-BY-5-BEST-α and SWPT. Each data point is an average of 10 instances. On the x-axis we plot the maximum job size p_{max}. $p_{max} = k$ means that the processing time was generated so that $p = 1$ with a probability of 0.9 and $p = k$ with a Probability of 0.1. The arrival times were generated uniformly on [0..500].

3.2.5. Effect of Local Improvement. Finally, we experimented with the impact of local improvement, which can be a very powerful algorithmic technique for certain scheduling problems. Specifically, given a solution, we considered all pairs and all triples of jobs, switching any that led to an improved solution and iterating until no improving switch existed. We applied these to the solutions of all the heuristics. For the heuristics that considered several possible schedules we applied local improvement to all of them and took the best resulting schedule. The results, reported in Table 6, indicate that the combination of local improvement applied to a collection of high-quality schedules yield the best results. When combined with local improvement techniques the SCHEDULE-BY-BEST-α heuristics invariably outperform the SWPT heuristic. We note further that applying this local improvement from scratch (random orderings) was not competitive with its application to a good initial schedule. The latter approach on average yielded solutions of quality 40% to 50% better.

If we switch all pairs of jobs, then local search runs in $O(n^2)$ time. Instead of switching all pairs of jobs, if we randomly choose $O(n)$ pairs of jobs to switch, then local search can be made to run in $O(n)$ time. On the other hand, solving the y_{jt}-relaxation takes $O(n \log n)$ time and solving the x_{jt}-relaxation takes pseudopolynomial time. Therefore, if we are given a fixed amount of time and if the quality of the schedule is not highly critical, then solving the y_{jt}-relaxation followed by

Table 6 Performance of the Algorithms Based on the Three (y_{jt}, BPP2, x_{jt}) Formulations After Some Local Improvement

	Hard			Synthetic 1			Synthetic 2		
	Mean	Std. dev.	Max	Mean	Std. dev.	Max	Mean	Std. dev.	Max
SCHEDULE-BY-FIXED-α (y_{jt})	1.594	0.264	2.137	1.088	0.061	1.330	1.200	0.073	1.438
SCHEDULE-BY-20-BEST-α (y_{jt})	1.402	0.148	1.679	1.022	0.022	1.138	1.096	0.035	1.196
SCHEDULE-BY-5-BEST-α (y_{jt})	1.420	0.164	1.770	1.024	0.023	1.140	1.100	0.037	1.213
SCHEDULE-BY-FIXED-α (BPP2)	1.591	0.270	2.141	1.081	0.058	1.339	1.175	0.066	1.371
SCHEDULE-BY-20-BEST-α (BPP2)	1.398	0.133	1.624	1.022	0.022	1.143	1.096	0.035	1.191
SCHEDULE-BY-5-BEST-α (BPP2)	1.413	0.142	1.652	1.024	0.023	1.143	1.100	0.036	1.213
SCHEDULE-BY-20-BEST-α (x_{jt})	1.429	0.197	1.871	1.021	0.022	1.125	—	—	—
SWPT	1.746	0.284	2.321	1.031	0.030	1.147	1.110	0.042	1.241

Note. We report the mean, standard deviation, and maximum of the ratio of the performance of the algorithm to the best lower bound.

local search is a better approach than trying to get a better lower bound by solving the x_{jt}-relaxation.

4. Extending the Techniques: A Real-Life Scheduling Problem

In this section, we demonstrate that the approximation algorithms evaluated in this paper also have practical value by illustrating their effectiveness on a real-life complex scheduling problem arising at BASF AG in Ludwigshafen, Germany. The problem is a resource-constrained project-scheduling problem and is described, together with methods that have been proposed for its solution, in Kallrath and Wilson (1997).

A number of orders have to be scheduled. Each order is broken down and produced in several identical pieces. Each piece in turn consists of several tasks, representing processing steps. Each task is characterized by a specific personnel requirement and a specific duration. The production of some of the orders are linked, i.e., the output of one order serves as input for another. Labor is the limiting resource. A fixed number of workers are available. The objective is minimizing the schedule length.

The problem is modeled by introducing a job for each piece of an order. As all pieces of an order are identical, the ordering of identical pieces is fixed using precedence constraints to avoid symmetries. Due to the fact that some orders are connected, additional precedence relations between jobs exist. Each job has, in addition to a processing time, a labor profile (duration and personnel requirement of the tasks). The jobs have to be scheduled on a single machine with limited capacity (number of workers). The objective is to minimize the makespan.

Various time-indexed formulations for the above problem have been proposed and investigated. We have used the so-called block formulation, which aggregates certain jobs into composite jobs, for all our computational experiments. The block formulation also formed the basis for an LP based branch-and-bound algorithm developed by Cavalcante et al. (2001). The algorithm employs the SCHEDULE-BY-$\overline{C}_j$ and the SCHEDULE-BY-FIXED-α (with $\alpha = 0.5$) heuristics as well as some local improvement techniques to obtain feasible schedules at each node of the search tree. The solutions obtained for the four available data sets, each corresponding to a different number of available workers (18, 20, 24, 27), were 469, 418, 366, and 363, respectively.

We investigated whether the use of improved heuristics would result in improved overall performance. Therefore, we replaced the heuristics SCHEDULE-BY-$\overline{C}_j$ and SCHEDULE-BY-FIXED-α by BEST-α and SCHEDULE-BY-RANDOM-α_j. The new solutions obtained for the four available data sets (with an imposed time limit on cpu time of 4 hours), were 457, 401, 363, and 363, respectively, which are the best known solutions for all these instances. We note that the biggest advantage of these multiple-α heuristics is that they supply a number of high-quality schedules for which to apply local improvement. We form all the schedules, one for each α, apply the local improvement to each schedule, and then choose the best one.

Table 7 provides a little more insight into the value of the various heuristics. For each of the four data sets, we present the value of the makespan obtained by the heuristics, as well as the value of the makespan after local search techniques have been applied to improve the initial schedule, at the root node of the search tree.

We note that the quality of the initial schedule is important. We have conducted a small experiment in which we applied the local improvement techniques to 1,000 randomly generated feasible schedules, but were unable to produce schedules of comparable quality.

5. Discussion and Future Directions

Our main goal in this paper is simply to understand the empirical impact of ideas and techniques that arose in the quest for improved worst-case performance guarantees for algorithms for $1|r_j|\sum w_jC_j$. Our results demonstrate that theoretical progress can lead

Table 7 Manufacturing Application from BASF AG, Germany

	18		20		24		27	
Best known prior to this work	469		418		363		363	
Best known through this work	457		401		363		363	
SCHEDULE-BY-$\overline{C}_j$	509	480	452	433	406	369	376	363
SCHEDULE-BY-α ($\alpha = 0.5$)	524	469	449	429	388	366	396	363
BEST(ALL)-α	500	469	427	419	372	365	374	363
SCHEDULE-BY-RANDOM-α_j	481	469	423	416	376	363	364	363

Note. The column headings indicate the number of available workers (18, 20, 24, 27). The first row gives the best known makespan of the schedule prior to our work and the second row gives the best known makespan of the schedule through our work. The last four rows give the makespan before local improvement and after doing some local improvement at the root node.

to progress in practice. Our goal is not to attempt to develop the best possible heuristics or to compare our work with all previous computational work on the problem. Therefore, the experimental results in this paper, taken in conjunction with a number of previous papers on both enumerative and approximation approaches, represent an example of good synergy between theory, experimentation, and practice.

In this study we touch on several approaches to scheduling problems: Linear-programming, approximation, combinatorial relaxations and branch-and-bound, and local improvement, and made a modest contribution to further our understanding of their relationship. We feel that it is an important direction to continue to try to understand, in a unified fashion, the different roles that these elements can play in the theory and practice of scheduling.

Acknowledgments

The authors are grateful to Jan Karel Lenstra, Chris Potts, Maurice Queyranne, David Shmoys, and Cliff Stein for helpful discussions. Martin W. P. Savelsbergh's research was partially supported by NSF Grant DMI-9410102. R. N. Uma's research was partially supported by NSF Grant CCR-9626831 and NSF Grant DMI-9970063. Joel Wein's research was 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. This work was done while the second author was a graduate student at Polytechnic University. Some of this work was done while the third author was visiting the IBM T. J. Watson Research Center.

References

Anderson, E. J., C. A. Glass, C. N. Potts. 1997. Machine scheduling. E. Aarts, J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*. John Wiley and Sons, Chichester, UK, 361–414.

Belouadah, H. 1985. Scheduling to minimize total cost. M.Sc. thesis, University of Keele, Staffordshire, UK.

Belouadah, H., M. E. Posner, C. N. Potts. 1992. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Appl. Math.* **36** 213–231.

Bianco, L., S. Ricciardelli. 1982. Scheduling of a single machine to minimize total weighted completion time subject to release dates. *Naval Res. Logist. Quart.* **29** 151–167.

Cavalcante, C. C. B., C. Carvalho de Sousa, M. W. P. Savelsbergh, Y. Wang, L. A. Wolsey. 2001. Scheduling projects with labour constraints. *Discrete Appl. Math.* **112** 27–52.

Chakrabarti, S., C. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, J. Wein. 1998. Improved bounds on relaxations of a parallel machine scheduling problem. *J. Combin. Optim.* **1** 413–426.

Chandra, R. 1979. On $n/1/\overline{F}$ dynamic deterministic systems. *Naval Res. Logist. Quart.* **26** 537–544.

Chekuri, C., R. Motwani, B. Natarajan, C. Stein. 2000. Approximation techniques for average completion time scheduling. *SIAM J. Comput.* **31**.

De Sousa, J. P., L. A. Wolsey. 1992. A time-indexed formulation of non-preemptive single-machine scheduling problems. *Math. Programming* **54** 353–367.

Dessouky, M. I., J. S. Deogun. 1981. Sequencing jobs with unequal ready times to minimize mean flow-time. *SIAM J. Comput.* **10** 192–202.

Dyer, M. E., L. A. Wolsey. 1990. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Appl. Math.* **26** 255–270.

Goemans, M. X. 1996. A supermodular relaxation for scheduling with release dates. *Proc. 5th MPS Conf. on Integer Programming and Combin. Optim., LNCS* Vol. 1084, Springer-Verlag, Berlin, Germany, 288–300.

Goemans, M. X. 1997. Improved approximation algorithms for scheduling with release dates. *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms, LNCS* Vol. 1099, Springer-Verlag, Berlin, Germany, 591–598.

Goemans, M. X., M. Queyranne, A. Schulz, M. Skutella, Y. Wang. 2002. Single machine scheduling with release dates. *SIAM J. Discrete Math.* **15** 165–192.

Hall, L. A., D. B. Shmoys, J. Wein. 1996. Scheduling to minimize average completion time: Off-line and on-line algorithms. *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, ACM, New York, 142–151.

Hall. L. A., A. S. Schulz, D. B. Shmoys, J. Wein. 1997. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.* **3** 513–544.

Hariri, A. M. A., C. N. Potts. 1983. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Appl. Math.* **5** 99–109.

Kallrath, J., J. M. Wilson. 1997. *Business Optimisation Using Mathematical Programming*. Macmillan Publishers, Hampshire, UK.

Kellerer, H., T. Tautenhahn, G. J. Woeginger. 1999. Approximability and nonapproximability results for minimizing total flow-time on a single machine. *SIAM J. Comput.* **28** 1155–1166.

Phillips, C., C. Stein, J. Wein. 1998. Minimizing average completion time in the presence of release dates. *Math. Programming B* **82** 199–224.

Posner, M. E. 1985. Minimizing weighted completion times with deadlines. *Oper. Res.* **33** 562–574.

Potts, C. N., L. N. van Wassenhove. 1983. An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *Eur. J. Oper. Res.* **12** 379–387.

Queyranne, M. 1993. Structure of a simple scheduling polyhedron. *Math. Programming* **58** 263–285.

Queyranne, M., A. S. Schulz. 1994. Polyhedral approaches to machine scheduling. Technical report 408/1994, Institut für Mathematik, Technische Universität Berlin, Berlin, Germany.

Rinaldi, G., A. Sassano. 1977. On a job scheduling problem with different ready times: Some properties and a new algorithm to determine the optimal solution. Tech. report R.77–24, Istituto di Automatica, Università di Roma, Rome, Italy.

Schulz, A. S. 1996. Scheduling to minimize total weighted completion time: Performance guarantees of LP based heuristics and lower bounds. *Proc. 5th MPS Conf. Integer Programming Combin. Optim., LNCS* Vol. 1084. Springer-Verlag, Berlin, Germany, 301–315.

Schulz, A. S., M. Skutella. 1997a. Scheduling LPs bear probabilities: Randomized approximations for min-sum criteria. R. Burkard, G. Woeginger, eds. *Algorithms–ESA'97, LNCS* Vol. 1284, *Proc. 5th Annual Eur. Sympos. Algorithms.* Springer, Berlin, Germany, 416–429.

Schulz, A. S., M. Skutella. 1997b. Random-based scheduling: New approximations and LP lower bounds. J. Rolim, ed. *Randomization and Approximation Techniques in Computer Science, LNCS* Vol. 1269, *Proc. Internat. Workshop RANDOM'97*. Springer, Berlin, Germany, 119–133.

Uma, R. N., J. Wein, D. P. Williamson. 2003. On the relationship between combinatorial and LP-based approaches to NP-hard scheduling problems. Technical report, Department of Computer Science, University of Texas at Dallas, Richardson, TX. (Preliminary version appeared in *IPCO: 6th Integer Programming Combin. Optim. Conf.* 1998.)

van den Akker, M. 1994. LP-based solution methods for single-machine scheduling problems. Ph.D. thesis, Department of Mathematics and Computing Science, Eindhoven Univ. of Technology, Eindhoven, The Netherlands.

van den Akker, M., C. A. J. Hurkens, M. W. P. Savelsbergh. 2000. A time-indexed formulation for single-machine scheduling problems: Column generation. *INFORMS J. Comput.* **12** 111–124.

van den Akker, M., C. P. M. Van Hoesel, M. W. P. Savelsbergh. 1999. A polyhedral approach to single-machine scheduling problems. *Math. Programming* **85** 541–572.

Wang, Y. 1996. Bicriteria job scheduling with release dates. Tech. Rep., Max-Planck-Institut für Informatik, Saarbrücken, Germany.

Wolsey, L. A. 1985. Mixed integer programming formulations for production planning and scheduling problems. *12th Internat. Sympos. Math. Programming*, MIT, Cambridge, MA.