Improved Scheduling Algorithms for Minsum Criteria

(Extended Abstract)

Soumen Chakrabarti* Cynthia A. Phillips** Andreas S. Schulz***
David B. Shmoys† Cliff Stein‡ Joel Wein§

Abstract. We consider the problem of finding near-optimal solutions for a variety of $\mathcal{NP}$-hard scheduling problems for which the objective is to minimize the total weighted completion time. Recent work has led to the development of several techniques that yield constant worst-case bounds in a number of settings. We continue this line of research by providing improved performance guarantees for several of the most basic scheduling models, and by giving the first constant performance guarantee for a number of more realistically constrained scheduling problems. For example, we give an improved performance guarantee for minimizing the total weighted completion time subject to release dates on a single machine, and subject to release dates and/or precedence constraints on identical parallel machines. We also give improved bounds on the power of preemption in scheduling jobs with release dates on parallel machines.

We give improved on-line algorithms for many more realistic scheduling models, including environments with parallelizable jobs, jobs contending for shared resources, tree precedence-constrained jobs, as well as shop scheduling models. In several of these cases, we give the first constant performance guarantee achieved on-line. Finally, one of the consequences of our work is the surprising structural property that there are schedules that simultaneously approximate the optimal makespan and the optimal weighted completion time to within small constants. Not only do such schedules exist, but we can find approximations to them with an on-line algorithm.

1 Introduction

Recently there has been significant progress in giving approximation algorithms to minimize average weighted completion time for a variety of $\mathcal{NP}$-hard scheduling problems [16, 11, 18]. Constructing a schedule to minimize average completion time in a one-machine or parallel machine scheduling environment has long

* soumen@cs.berkeley.edu. Computer Science Division, U. C. Berkeley, CA 94720. Supported partly by ARPA/DOD (DABT63-92-C-0026), DOE (DE-FG03-94ER25206), and NSF (CCR-9210260, CDA-8722788 and CDA-9401156). Part of the work was done while visiting IBM T. J. Watson Research Center.

** caphill@cs.sandia.gov. Sandia National Labs, Albuquerque, NM. This work was performed under U.S. Department of Energy contract number DE-AC04-76AL85000.

*** schulz@math.tu-berlin.de. Department of Mathematics, Technical University of Berlin, 10623 Berlin, Germany. Supported by the graduate school Algorithmische Diskrete Mathematik (DFG), grant We 1265/2-1.

† shmoys@cs.cornell.edu. School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853. Research partially supported by NSF grant CCR-9307391.

‡ cliff@cs.dartmouth.edu. Department of Computer Science, Sudikoff Laboratory, Dartmouth College, Hanover, NH. Research partially supported by NSF Award CCR-9308701, a Walter Burke Research Initiation Award and a Dartmouth College Research Initiation Award.

§ wein@mem.poly.edu. Department of Computer Science, Polytechnic University, Brooklyn, NY, 11201. Research partially supported by NSF Research Initiation Award CCR-9211494 and a grant from the New York State Science and Technology Foundation, through its Center for Advanced Technology in Telecommunications.

been known to be polynomial-time solvable [21, 4, 12]; when one adds release dates, precedence constraints, or weights, essentially all versions of the problem become $\mathcal{NP}$-hard, and until [16, 11, 18] very little was known about approximation algorithms with good performance guarantees.

Recent progress on these problems follows from two basic approaches. In the first approach a linear programming relaxation of the scheduling problem is solved, and then a schedule is constructed simply by list scheduling in a natural order dictated by an optimum solution to the linear program. The second approach is a general on-line framework, in which one attempts to pack the most profitable jobs into successive intervals of geometrically increasing size.

In this paper we improve and extend these techniques. We develop new analytical tools that lead to improved off-line and on-line approximation algorithms for many of the problems considered in [16, 11, 18]. By *on-line*, we mean that the algorithm constructs the schedule in time, and that the existence of a job is known only at its release date. We also extend the on-line techniques to a number of different scheduling models. Finally, we extend the on-line technique to yield *bicriteria* scheduling algorithms: we give very general algorithmic and structural results for schedules that simultaneously come within a small factor of both optimal schedule length and average weighted completion time.

Notation and models. We will consider a number of scheduling models; in the most basic we have a set J of n jobs and m identical parallel machines. Each job j has a positive integral processing requirement (size) p_j and must be processed for that amount of time on one of the machines. In *preemptive schedules*, a job may be interrupted and continued later on another machine. In *nonpreemptive schedules*, a job must be processed in an uninterrupted fashion.

We will be interested in constrained scheduling problems, in which each job j may have a release date r_j before which it cannot be processed, and/or there may be a partial order $\prec$ on the jobs, where $j \prec k$ means that job k cannot start before job j completes. For most of our models a job is processed on only one machine at a time, and a machine can process only one job at a time. We will at times consider *parallelizable jobs*, where job j can run on a number of processors simultaneously, with some specified speedup.

We denote the completion time of a job j in a schedule S as C_j^S; the S will be dropped when clear from the context. Most often we seek to minimize the total completion time $\sum_{j \in J} C_j$, or the total *weighted* completion time: a weight w_j is associated with each job and the goal is to minimize $\sum_{j \in J} w_j C_j$. If we divide either objective function by n, we obtain the average completion time and average weighted completion time objectives. In contrast, the makespan is $\max_j C_j$. All of the problems we consider are $\mathcal{NP}$-hard, and thus we seek approximation algorithms. We define a ρ-approximation algorithm to be a polynomial-time algorithm that delivers a solution of quality at most ρ times optimal.

Discussion of results. The first part of the paper has its roots in the observation that one may construct a nonpreemptive parallel machine schedule by list scheduling in order of the completion times of a preemptive schedule while at most losing a factor of $(3 - \frac{1}{m})$ in average weighted completion time [16]. This proved to be an important observation, for a generalization of this idea proved to be a powerful tool in rounding solutions to linear programming formulations of a number of scheduling problems [11, 18].

In this paper we give improved rounding techniques for these problems; specifically, we give an algorithm that takes a preemptive parallel machine schedule of average completion time C and converts it to a nonpreemptive schedule of average completion time $\frac{7}{3}C$. As a corollary, this gives the same bound on the power of preemption in this scheduling environment; allowing preemption improves the average completion time by at most a factor of $\frac{7}{3}$. When applied to the solution of a linear programming formulation considered by Schulz [18] and independently by Queyranne and by Hall, Shmoys & Wein, this technique yields a 3.5-approximation algorithm for the nonpreemptive scheduling of parallel machines subject to release dates to minimize average completion time; this improves on the previous best bound of $(4 - \frac{1}{m})$.

In the second part of the paper we give a framework for designing on-line algorithms that minimize the average weighted completion time, by improving and extending a result of Hall, Shmoys, & Wein [11]. By incorporating an idea of Goemans & Kleinberg [8] that exploits randomization in an elegant way, we can improve the performance guarantee of the resulting algorithms; the resulting bounds are quite strong, and in certain cases even improve upon off-line bounds achieved via the linear programming formulations previously discussed. Furthermore, we show that this framework actually produces a schedule that is simultaneously near-optimal with respect to both the average weighted completion time objective, and with respect to the makespan objective.

The on-line framework requires a dual ρ-approximation algorithm for the problem of scheduling a maximum weight subset of jobs to complete by a given deadline D. We show that, given a dual ρ-approximation algorithm for this problem, there is an algorithm that yields a schedule that is simultaneously within a factor of 4ρ and 2.89ρ of the minimum makespan and total weighted completion time, respectively. We also give a structural theorem, that shows that, for a very general class of scheduling models, there exist schedules that are simultaneously within a factor of 2 of the minimum total weighted completion time, and within a factor of 2 of the minimum makespan. Such simultaneous bounds were only known in very restrictive scheduling environments.

In the third part of the paper we design the above-mentioned dual approximation routine for a number of scheduling problems. As a result, we give the first (off-line or on-line) constant-approximation algorithms for open shop scheduling and job shop scheduling with a fixed number of machines. We also give algorithms for a number of models that capture realistic elements of parallel computing, including resource constraints, forest precedence constraints, and parallelizable jobs. Many of our on-line results are the first constant approximation algorithms for the corresponding problems, on-line or off-line, while others are on-line algorithms whose performance is close to the best known off-line results.

2 Tighter bounds for parallel machines with release dates

In this section, we consider preemptive and nonpreemptive schedules for minimizing average completion time on parallel machines with release dates. We will show that the ratio between the preemptive and nonpreemptive average completion times is at most $\frac{7}{3}$ by giving an algorithm to convert any preemptive schedule to a nonpreemptive schedule with average completion time at most $\frac{7}{3}$ times greater. This improves on the bound of $(3-\frac{1}{m})$, given by Phillips, Stein and Wein [16]. We then use this technique to obtain a 3.5 approximation algorithm for $P|r_j|\sum C_j$, improving upon the best previous bound of $(4-\frac{1}{m})$ [18].

We use the algorithm CONVERT, introduced by [16], which takes a preemptive schedule P and list schedules the jobs nonpreemptively in the order of their completion times in P. Each job in turn is scheduled as early as possible without violating its release date, and without disturbing the jobs that have already been scheduled. We will show that this produces a schedule N that has average completion time at most $\frac{7}{3}$ that of P. For any schedule S, let $C^S = \sum_{j\in J} C_j^S$.

Theorem 1. *Given a preemptive schedule P for scheduling jobs with release dates on parallel machines, algorithm* CONVERT *produces a nonpreemptive schedule whose average completion time is at most $\frac{7}{3}C^P$.*

Proof. Let $\mathcal{L}$ be the last m jobs to finish in P and let $\mathcal{K}$ be the remaining jobs. In P, multiple jobs in $\mathcal{L}$ can finish on the same machine. Simple movement of job pieces, however, yields a new schedule of no greater total completion time where the last work done on each machine is the last piece of some job in $\mathcal{L}$. We will assume, for this proof, that P is of this form. The jobs in $\mathcal{L}$ are the last m jobs to start in N. Still, some machines may not process any jobs in $\mathcal{L}$. We define a modified schedule, N', that guarantees that each job in $\mathcal{L}$ runs last on some machine in N'. To obtain N' from N, for each machine that has more than one job from $\mathcal{L}$ in N, all but the first to run are removed and each one is appended to the schedule of some machine that has no such job. Since N is a list schedule, job completion times cannot decrease, i.e., we have $C_j^{N'} \geq C_j^N$, for all jobs j. We further divide $\mathcal{L}$ into two sets $\mathcal{L}_1$ and $\mathcal{L}_2$. The set $\mathcal{L}_2$ contains those jobs $j \in \mathcal{L}$ for which $C_j^{N'} = r_j + p_j$ and the set $\mathcal{L}_1 = \mathcal{L} - \mathcal{L}_2$. We let $C^{L_1} = \sum_{j\in\mathcal{L}_1} C_j^P$, $C^K = \sum_{j\in\mathcal{K}} C_j^P$, and $C^{L_2} = \sum_{j\in\mathcal{L}_2} C_j^P$.

Next we show that given a preemptive schedule P for parallel machine scheduling with release dates, algorithm CONVERT produces a nonpreemptive schedule N in which $C^N \leq 2C^K + 3C^{L_1} + 2C^{L_2}$. Phillips, Stein, and Wein[16] show that $C_j^N \leq 2C_j^P + p_j$. If we apply this bound to the jobs in $\mathcal{L}_1 \cup \mathcal{K}$, apply the bound $C_j^N \leq C_j^{N'} = r_j + p_j$ to the jobs in $\mathcal{L}_2$, and sum over all jobs, we get that

$$\begin{aligned} C^N &\leq \sum_{j\in\mathcal{K}\cup\mathcal{L}_1} (2C_j^P + p_j) + \sum_{j\in\mathcal{L}_2} (r_j + p_j) \leq 2(C^K + C^{L_1}) + \sum_{j\in J} p_j + \sum_{j\in\mathcal{L}_2} r_j \\ &\leq 2(C^K + C^{L_1}) + \sum_{j\in J} p_j + C^{L_2} \quad . \end{aligned} \tag{1}$$

But $\sum_{j \in J} p_j \leq C^{L_1} + C^{L_2}$, because each unit of processing contributes to the completion time of at most one of the jobs in $\mathcal{L}_1 \cup \mathcal{L}_2$ and so plugging into (1), we obtain the desired bound.

Now we show that $C^{N'} \leq 3C^K + C^{L_1} + 2C^{L_2}$. Since $C_j^{N'} = C_j^N$, for jobs in $\mathcal{K}$, we can, as above, apply the bound $C_j^{N'} \leq 2C_j^P + p_j$ for jobs $j \in \mathcal{K}$. We can also, by definition, apply the bound $C_j^{N'} = r_j + p_j$ for jobs $j \in \mathcal{L}_2$.

Now we bound the completion times of jobs in $\mathcal{L}_1$, which are each on a different machine. Consider a particular job $j \in \mathcal{L}_1$ on some machine M. Let t be the last time that machine M was idle before running j. Let k be the job that ran immediately after that idle time. Note that $r_k = t$, or else we could have run k earlier. Let $j_1, \ldots, j_\ell$ be the jobs that run on M between k and j. Then

$$C_j^{N'} = r_k + p_k + p_{j_1} + \ldots + p_{j_\ell} + p_j \quad . \tag{2}$$

Since j is in $\mathcal{L}_1$ and not in $\mathcal{L}_2$, we know that there is a job running immediately before j and hence $k \neq j$, and $k \in \mathcal{K}$. If we sum (2) over all jobs in $\mathcal{L}_1$, each job in $\mathcal{K} \cup \mathcal{L}_1$ contributes to the right-hand side at most once, since each job in $\mathcal{L}_1$ is on a different machine. The jobs in $\mathcal{L}_2$ don't contribute at all, since they are run on different machines than the ones which run jobs in $\mathcal{L}_1$. Thus, summing over $\mathcal{L}_1$, we get $\sum_{j \in \mathcal{L}_1} C_j^{N'} \leq \sum_{k \in \mathcal{K}} r_k + \sum_{j \in \mathcal{L}_1 \cup \mathcal{K}} p_j$. Combining this with the bounds above for the remaining jobs we get $C^N \leq C^{N'} \leq 3C^K + 2C^{L_2} + C^{L_1}$. Balancing the two cases proves the theorem. □

This theorem improves upon the previous best bound of $(3 - \frac{1}{m})$ for the ratio of average completion time in preemptive vs. nonpreemptive schedules; however, the best known preemptive algorithm is a 2-approximation algorithm [16]; therefore a direct application of this theorem does not give an approximation algorithm with an improved performance guarantee. We can, however, apply it to a different relaxation of the nonpreemptive schedule and obtain an improved approximation bound; we also believe that the ideas will prove useful in sharpening the analysis of other linear programming formulations for scheduling problems.

Schulz [18], and independently Queyranne and Hall, Shmoys and Wein, have given a $(4 - \frac{1}{m})$-approximation algorithm for nonpreemptively scheduling parallel machines with release dates to minimize average weighted completion time, which is based on the LP-relaxation of a formulation in *completion time variables:* with each job j is associated a variable C_j. We can show that by letting the solution to the LP-relaxation play the role of the preemptive completion time, we obtain an improved approximation algorithm.

Theorem 2. *There is a 3.5-approximation algorithm to minimize the average completion time of jobs with release dates on parallel machines.*

In Section 3 we give different techniques that yield better randomized performance guarantees.

3 An on-line framework for bicriteria scheduling

In this section, we will improve and extend a result of Hall, Shmoys, & Wein [11], that gives a framework for designing on-line algorithms that minimize the total weighted completion time. By incorporating an idea of Goemans & Kleinberg [8] that exploits randomization in an elegant way, we can improve the performance guarantee of the resulting algorithms. Furthermore, we show that this framework actually produces a schedule that is simultaneously near-optimal with respect to both the total weighted completion time objective, and the maximum completion time objective. The result of Goemans & Kleinberg improves upon a result of Blum et al., who present a similar bicriteria result for traveling-salesman type problems [3]; we show that their approach applies to a very general class of scheduling problems.

Our framework Greedy-Interval is based on algorithms for the *maximum scheduled weight problem:* given a deadline D, a set of jobs available at time 0, and a weight for each job, construct a schedule that maximizes the total weight of jobs completed by time D. We require a *dual ρ-approximation algorithm*, DualPack, which produces a schedule of length $\leq \rho D$ and whose total weight is at least the optimal weight for the deadline D.

Greedy-Interval uses DualPack in the following way. Let $\tau_\ell = \alpha 2^\ell$, where we shall set the constant $\alpha \in [1/2, 1)$ later. In iteration $\ell = 1, 2, \ldots$ we construct the schedule for the time interval $(\rho\tau_\ell, \rho\tau_{\ell+1}]$. Let J_ℓ denote the set of jobs that have been released by τ_ℓ, but not scheduled in the previous iterations. We call DualPack for the set of jobs J_ℓ (modified so that each is available at time 0) and the deadline $D = \tau_\ell$. Since $\rho\tau_{\ell+1} - \rho\tau_\ell = \rho\tau_\ell$, we can translate the schedule produced by the dual approximation algorithm into the specified interval; furthermore, each job is scheduled no earlier than its true release date.

Fix an optimal schedule with respect to $\sum w_j C_j$, in which each job j completes at time C_j^*; let B_j denote the start of the interval $(\tau_{\ell-1}, \tau_\ell]$ in which job j completes. Hall, Shmoys, & Wein [11] show that Greedy-Interval finds a schedule of total weighted completion time at most $4\rho \sum_{j=1}^n w_j B_j \leq 4\rho \sum_{j=1}^n w_j C_j^*$.

The value B_j, for a particular $j \in \{1, \ldots, n\}$, is determined by the choice of α and the value of C_j^*. If, for our choice of α, $\sum_j w_j B_j = \beta \sum_j w_j C_j^*$, where $\beta < 1$, we get an improved performance guarantee of $4\beta\rho$. In fact, we shall simply choose α at random: choose X uniformly in the interval (0,1] and set $\alpha = 2^{-X}$. Consequently, the value β is a random variable. A routine calculation gives that

$$E[B_j] = C_j^* \int_0^1 2^{-x} dx = \frac{1}{2 \ln 2} C_j^* \quad . \tag{3}$$

By linearity of expectation, this implies that, for any instance, the expectation of the ratio between the total weighted completion time for the schedule found and for the optimum is at most $\frac{2\rho}{\ln 2} < 2.89\rho$.

Next consider the makespan of the Greedy-Interval schedule. Let $C^*_{\max}$ denote the optimal makespan, and let $B_{\max}$ denote the start of the interval that contains $C^*_{\max}$; i.e., $B_{\max} = \alpha 2^K$. We know that the algorithm must terminate by iteration

$K+1$, since the dual approximation algorithm must return all of the jobs when $D = 2B_{\max} > C^*_{\max}$. Hence, the makespan of the schedule is at most $\rho\alpha 2^{K+2} = 4\rho B_{\max} \leq 4\rho C^*_{\max}$. That is, we have shown a performance guarantee of 4ρ. If we choose α as above, then the expected performance ratio is also similarly improved, since $E[B_{\max}] = \frac{1}{2\ln 2} C^*_{\max}$.

Theorem 3. *Given a dual ρ-approximation algorithm* DualPack*, the randomized framework* Greedy-Interval *is, simultaneously, an on-line 4ρ-approximation algorithm to minimize the total weighted completion time, and an on-line 4ρ-approximation algorithm to minimize the makespan. Furthermore, for every instance, each objective function is expected to be within a factor of $\frac{2\rho}{\ln 2}$ of its respective optimum.*

The use of randomization is a relatively mild one. If we are interested in off-line results, then we can run the algorithm with many choices of α, select the best output, and thereby achieve the following deterministic results.

Corollary 4. *Given a dual ρ-approximation algorithm* DualPack*, the framework* Greedy-Interval *yields a 2.89ρ-approximation algorithm to minimize the total weighted completion time, and to minimize the makespan.*

This result yields the best known performance guarantee for minimizing the total weighted completion time subject to release date constraints in either a single-machine or a parallel-machine environment, improving results of [11, 18, 10]. Observe that in Corollary 4, we did not state that the two bounds could be achieved by the same schedule. Indeed, we do not know how to achieve these simultaneously, since a choice of α that is good for the first criterion might be bad for the latter. Nonetheless, if we weaken either one of the two bounds to 4ρ, then they can be achieved simultaneously by a deterministic polynomial-time algorithm.

We now compare our algorithmic results to existence theorems about schedules that are simultaneously near-optimal for these two objective functions. We can also show a general result that holds for any of the scheduling environments we consider in this paper, and a wide variety of other models: let S and T denote optimal schedules with respect to the makespan and total weighted completion time objectives, respectively; there exists a schedule with makespan at most $2C^S_{\max}$ and total weighted completion time at most $2\sum_j w_j C^T_j$.

We shall show how to construct the desired schedule from S and T. Let T' be the schedule induced from T by considering only those jobs j for which $C^T_j \leq C^S_{\max}$. Let S' be the schedule induced from S by considering only jobs for which $C^T_j > C^S_{\max}$. Construct the schedule N by first scheduling according to T' and then according to S'. Clearly, the makespan of N is at most $2C^S_{\max}$. Furthermore, each job j scheduled in T' completes at the same time in N as in T. For each job j scheduled in S', $C^T_j > C^S_{\max}$, and yet $C^N_j \leq 2C^S_{\max}$. Hence, $C^N_j < 2C^T_j$ for each job j.

This proof can be refined to yield somewhat better constants; the details will be given in the complete version of the paper. In contrast to this result, Hurkens

& Coster [13] have given a family of unrelated parallel machine instances for which all minimum total completion time schedules have makespan that is an $\Omega(\log n)$ factor greater than the optimal makespan.

3.1 Applying randomization to other interval-based algorithms

We can also use the randomized technique of Goemans and Kleinberg [8] to improve LP-based results of Hall, Schulz, Shmoys, & Wein [10]. In particular, we improve upon a 7-approximation algorithm for the problem of minimizing the total weighted completion time on scheduling parallel machines subject to precedence constraints and release dates. The algorithm of [10] first solves an LP relaxation of this problem to obtain an optimal solution $\widetilde{C}_j$. Assume that the jobs are indexed by nondecreasing LP value; the LP solution satisfies the following properties: (1) for each $j = 1, \ldots, n$, $\sum_{k=1}^{j} p_k/m \leq 2\widetilde{C}_j$; (2) for each $j = 1, \ldots, n$, $r_j \leq \widetilde{C}_j$; (3) $\widetilde{C}_j + p_k \leq \widetilde{C}_k$ whenever $j \prec k$; (4) $\sum_j w_j \widetilde{C}_j \leq \sum_j w_j C_j^*$.

Given this solution, the algorithm partitions the time horizon into intervals. Set $\tau_\ell = \alpha 2^\ell$, $\ell = 1, \ldots, L$, where we will judiciously choose $\alpha \in [0.5, 1)$. (In [10], this value was, in essence, 0.5.) We partition the time horizon into the intervals $(\tau_{\ell-1}, \tau_\ell]$, $\ell = 1, 2, \ldots, L$, where L is chosen so that each $\widetilde{C}_j$ value is contained in some interval. Partition the jobs into sets such that J_ℓ is defined to be the set of jobs j for which $\widetilde{C}_j$ lies within the ℓth interval, $\ell = 1, \ldots, L$. We construct disjoint schedules for each set J_ℓ by performing ordinary list scheduling on each set, using any list ordering that is consistent with the precedence constraints and ignoring any release dates. Set $\overline{\tau}_\ell = \tau_{\ell+1} + \sum_{k=1}^{\ell} t_k$, where $t_\ell = \sum_{j \in J_\ell} p_j/m$, $\ell = 1, \ldots, L$. We schedule the fragment for J_ℓ between $\overline{\tau}_{\ell-1}$ and $\overline{\tau}_\ell$, $\ell = 1, \ldots, L$.

It is relatively straightforward to show that this yields a feasible schedule. For each $j \in J_\ell$, we can bound the completion time of j in this schedule by $\beta_j + 6\tau_{\ell-1}$, where β_j is the length of some chain that ends with job j. This yields the performance guarantee of 7 of [10]. Let $B_j = \tau_{\ell-1}$ as in the on-line case. Thus, if α is set equal to 2^{-X} where X is selected uniformly from $(0, 1]$, then the expected value of $6B_j$ is equal to $\frac{3}{\ln 2}\widetilde{C}_j$. This implies that the expected ratio between the total weighted completion of the schedule found and the optimum is at most $1 + \frac{3}{\ln 2} < 5.328$. Of course, we can derandomize the algorithm as well.

Theorem 5. *There is a 5.33-approximation algorithm for nonpreemptively scheduling on parallel machines with release dates and precedence constraints.*

Applying techniques used in [10], it is also quite straightforward to generalize this theorem to the setting in which machine i runs at speed s_i. In that model, the resulting algorithm has a performance guarantee that is at most $\min\{2.89 + 2.45\frac{\max_i s_i}{\min_i s_i}, 2.89 + 5.31\sqrt{m-1}\}$.

4 Dual packing algorithms and applications

In order to apply our on-line framework of Section 3, we need to construct the corresponding subroutine $\mathsf{DualPack}(J, D)$. In this section, we shall present this subroutine for a number of scheduling environments.

Several of our results will involve parallelizable jobs. A parallelizable job is described by a processing time p_j and a number of machines m_j. A *non-malleable* job j must be run on exactly m_j processors, and has a specified running time p_j. A *perfectly malleable* job j may be run on μ processors, where $\mu = 1, \ldots, m_j$, and then has running time $p_j m_j/\mu$. A *malleable* job j is a common generalization where, for each possible number of processors $\mu = 1, \ldots, m$, there is a specified running time $p_j(\mu)$.

Our implementation of $\mathsf{DualPack}(J, D)$ has the same outline in each of our applications. First, we prune from J the jobs that are impossible to complete within D time units, either because of precedence constraints or because their processing time is too large. We shall restrict our attention to precedence constraints $\prec$ that are out-trees; in-trees can be handled analogously. To prune based on precedence constraints, define $\mathsf{PathToRoot}(j) = p_j$ if j is a root, and $\mathsf{PathToRoot}(j) = \mathsf{PathToRoot}(\mathsf{Parent}(j)) + p_j$ otherwise. These quantities can be computed in linear time, and the appropriate jobs can then be eliminated. Second, we use a routine $\mathsf{Knapsack}$ to solve a modified knapsack problem in order to find a set of jobs of sufficiently large weight to schedule. Third, we schedule those jobs selected using a known makespan algorithm.

The input to the routine $\mathsf{Knapsack}(J, S)$ consists of a set J of n items (jobs), where item j has weight w_j and size s_j, and a knapsack of size S; in addition, we might be given precedence constraints on the items; if $j_1 \prec j_2$ we forbid the packing of j_2 unless j_1 is also packed. The knapsack problem is to find the maximum weight set that can be packed into the knapsack; let the optimal value be denoted W^*. The routine $\mathsf{Knapsack}(J, S)$ finds a set of total weight at least W^* that has total size at most $(1 + \epsilon)S$, where $\epsilon > 0$ is an arbitrarily small constant. To achieve this, we round down each s_j by units of $\epsilon S/n$ and then use dynamic programming as in [14].

In each of the following subsections, we will give an implementation of $\mathsf{DualPack}$ for a specific problem; throughout, we denote the deadline by D and the set of jobs from which we choose by J. All of these are bicriteria results; we report only the min-sum result and omit proofs for brevity.

4.1 Malleable jobs

In this subsection we give an algorithm for malleable parallelizable jobs without precedence constraints. The best off-line performance guarantee known for the non-malleable special case, without release dates, is 8.53, due to Turek *et al* [22]; by applying an idea of Ludwig & Tiwari [15], this can be extended to the malleable case. Our *on-line* min-sum algorithm with release dates has a performance guarantee of $12 + \epsilon$, and if we allow randomization, a nearly identical guarantee of 8.67.

We implement $\mathsf{DualPack}$ as follows: for each job j, we find the value of μ such that $p_j(\mu) \leq D$ for which $\mu p_j(\mu)$ is minimized. Jobs for which there is no such μ are removed from J; otherwise, let m_j be the value for which this minimum is attained. If job j is scheduled by $\mathsf{DualPack}$, it will be run on m_j machines. We set the weight and size of job $j \in J$ to be w_j and $m_j p_j$, respectively, and then call

Knapsack, setting $J' = \textsf{Knapsack}(J, mD)$. Finally, we adapt the list scheduling algorithm of Garey & Graham [7] to schedule J'.

Theorem 6. *The above* DualPack *routine is a dual* $(3+\epsilon)$*-approximation algorithm for the maximum scheduled weight problem. This gives a deterministic on-line* $(12+\epsilon)$*-approximation algorithm for scheduling malleable jobs on parallel machines, and a randomized on-line algorithm with expected performance within* 8.67 *of optimal.*

Using the multidimensional knapsack routine in §4.4, together with adaptations of graph labeling results of [7], we can also generalize non-malleable jobs to resource constrained jobs. In this model, each job j holds $r_{ij} \le 1$ units of resource type i, $i = 1, \ldots, m$, while it runs for duration p_j. At most 1 unit of each resource is available at any time. We get an on-line $O(m)$-approximation algorithm for the min-sum problem; only makespan results were known previously.

4.2 Sequential jobs with tree precedence

Whereas Hall, Shmoys and Wein gave versions of DualPack for m parallel machines, they were not able to handle precedence-constrained jobs. We present here a routine DualPack that can handle forest precedence constraints. In contrast to the previous subsection, we now consider sequential tasks; each job must be scheduled to run on exactly one machine. Our DualPack routine is as follows. We remove from J all jobs j with $\textsf{PathToRoot}(j) > D$, and let $J' = \textsf{Knapsack}(J, mD)$. We then list schedule J'.

Theorem 7. *The above* DualPack *routine is a dual* $(2+\epsilon)$*-approximation algorithm for the maximum scheduled weight problem. This gives a deterministic on-line* $(8+\epsilon)$*-approximation algorithm to minimize the average weighted completion time of sequential jobs with release dates and forest precedence on parallel machines, and a randomized on-line algorithm with expected performance within* 5.78 *of optimal.*

4.3 Perfectly malleable jobs with tree precedence

We shall consider perfectly malleable jobs, as in Feldmann *et al* [6], and out-tree precedence constraints. (A study of precedence and non-malleability is initiated in [5].) Our DualPack routine is as follows. We remove from J any j with $\textsf{PathToRoot}(j) > D$, set $J' = \textsf{Knapsack}(J_\ell, mD)$, and list schedule J' as in [6]: let $\phi = (\sqrt{5}-1)/2$ be the golden ratio; whenever there is a job j with all of its predecessors completed and the number of busy processors is less than ϕm, schedule the job on the minimum of m_j and the number of free processors.

Theorem 8. *The above* DualPack *routine is a dual* $(2+\phi+\epsilon)$*-approximation algorithm for the maximum scheduled weight problem. This gives a deterministic on-line* 10.48*-approximation minsum algorithm for perfectly malleable jobs with forest precedence constraints, and a randomized algorithm with expected performance within* 7.58 *of optimal.*

4.4 Minsum shop scheduling

In shop scheduling each job consists of *operations*, each of which must run on a specified machine; for each job, no two of its operations may run concurrently. In the *job shop* problem the operations must proceed in a specified sequence, while in the *open shop* problem they can be scheduled in any order. Minimizing the makespan of job shops is perhaps the most notorious of difficult $\mathcal{NP}$-hard scheduling problems; even small instances are difficult to solve to optimality [1] and the best approximation algorithms give polylogarithmic performance guarantees [19].

We give the first constant-factor approximations for min-sum shop scheduling with a fixed number of machines m. No approximation algorithms were known for minimizing average completion time in shop scheduling, except for the recent results by Schulz that give an m-approximation algorithm for *flow shop scheduling*, the special case of job shop in which the order is the same for each job [18]. We assume for ease of presentation that each job has at most one operation on a machine; let p_{ij} be the size of the operation of job j on machine i.

We again give a version of DualPack for this problem. Let the maximum weight of jobs that can be scheduled by D be W^*, and the optimal set of jobs be J^*. We note that two lower bounds on the makespan of a shop schedule (job or open) are the maximum job size $P_{\max} = \max_{j \in J^*}\{\sum_i p_{ij}\}$, and the maximum machine load, $\Pi_{\max} = \max_i\{\sum_{j \in J^*} p_{ij}\}$. Let W' be the optimum of the following multidimensional knapsack problem: maximize $\sum_j w_j x_j$ such that $\sum_j x_j p_{ij} \leq D$ for $i = 1, \ldots, m$, and $\sum_i x_j p_{ij} \leq D$ for $j = 1, \ldots, n$, where $x_j \in \{0, 1\}$; note that $W' \geq W^*$.

Lemma 9. *For fixed m there is an algorithm for the multidimensional knapsack problem that produces a solution of weight $W'' \geq W'$ for which $\sum_j x_j p_{ij} \leq (1+\epsilon)D$ and $\sum_i x_j p_{ij} \leq D$.*

We now show that having found a set of jobs of total weight $W'' \geq W^*$, we can schedule this set within makespan $(2+\epsilon)D$. For an open shop instance a nonpreemptive schedule of length $P_{\max} + \Pi_{\max} \leq (2+\epsilon)D$, and a preemptive schedule of length $\max(P_{\max}, \Pi_{\max}) \leq (1+\epsilon)D$ can be constructed [9, 2]. For job shop scheduling we can adapt the known $(2+\epsilon)$-approximation makespan algorithm for fixed m [19].

Theorem 10. *There is a randomized $(2.89+\epsilon)$-approximation algorithm for preemptive open shop scheduling with a fixed number of machines to minimize average weighted completion time, and there are randomized $(5.78+\epsilon)$-approximation algorithms for nonpreemptive open shop and job shop scheduling on a fixed number of machines to minimize average weighted completion time.*

Acknowledgments We thank Jon Kleinberg for pointing out that the randomized interval selection technique could be applied to Greedy-Interval, and Leslie Hall for helpful discussions.

References

1. D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing*, 3:149–156, 1991.
2. I. Bárány and T. Fiala. Többgépes ütemezési problémák közel optimális megoldása. *Szigma-Mat.-Közgazdasági Folyóirat*, 15:177–191, 1982.
3. A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In STOC 26, pages 163–172, 1994.
4. J.L. Bruno, E.G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17:382–387, 1974.
5. S. Chakrabarti and S. Muthukrishnan. Resource scheduling for parallel database and scientific applications. To appear in SPAA 96, June 1996.
6. A. Feldmann, M. Kao, J. Sgall, and S. Teng. Optimal online scheduline of parallel jobs with dependencies. In STOC 25, pages 642–653, 1993.
7. M.R. Garey and R.L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4:187–200, 1975.
8. M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In SODA 7, pages 152–157, 1996.
9. T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23:665–679, 1976.
10. L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. Joint journal version of [11] and [18]; in preparation.
11. L.A. Hall, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In SODA 7, pages 142–151, 1996.
12. W. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21:846–847, 1973.
13. C.A.J. Hurkens and M.J. Coster. On the makespan of a schedule minimizing total completion time for unrelated parallel machines. Unpublished manuscript.
14. D.S. Johnson and K.A. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research*, 8:1–14, 1983.
15. W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. in SODA 5, pages 167–176, 1994.
16. C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. In WADS 4, pages 86–97, 1995.
17. M. Queyranne and A.S. Schulz. Polyhedral approaches to machine scheduling. Preprint 408/1994, Dept. of Mathematics, Technical University of Berlin, 1994.
18. A.S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. Preprint 474/1995, Dept. of Mathematics, Technical University of Berlin. To appear in IPCO V, June 1996.
19. D.B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23:617–632, 1994.
20. D.B. Shmoys and É. Tardos. Scheduling parallel machines with costs. In SODA 4, pages 448–455, 1993.
21. W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
22. J. Turek, U. Schwiegelshohn, J. Wolf, and P. Yu. Scheduling parallel tasks to minimize average response time. In SODA 5, pages 112–121, 1994.