Minimum Weighted Completion Time (1999; Afrati et al.) [*]

V.S. Anil Kumar, Virginia Tech
Madhav V. Marathe, Virginia Tech
Srinivasan Parthasarathy, IBM T.J. Watson Research Center
Aravind Srinivasan, University of Maryland, College Park
entry editor: Sanjeev Khanna

INDEX TERMS: Machine scheduling, Weighted completion time, Resource allocation.
SYNONYMS: Average Weighted Completion Time.

1 PROBLEM DEFINITION

The minimum weighted completion time problem involves (i) a set J of n jobs, a positive weight w_j for each job $j \in J$, and a release date r_j before which it cannot be scheduled; (ii) a set of m machines, each of which can process at most one job at any time, and (iii) an arbitrary set of positive values $\{p_{i,j}\}$, where $p_{i,j}$ denotes the time to process job j on machine i. A schedule involves assigning jobs to machines and choosing an order in which they are processed. Let C_j denote the completion time of job j for a given schedule. The *weighted completion time* of a schedule is defined as $\sum_{j \in J} w_j C_j$, and the goal is to compute a schedule that has the minimum weighted completion time.

In the scheduling notation introduced by Graham et al. [7], a scheduling problem is denoted by a 3-tuple $\alpha \mid \beta \mid \gamma$, where α denotes the machine environment, β denotes the additional constraints on jobs, and γ denotes the objective function. In this article, we will be considered with the α-values 1, P, R, and Rm which respectively denote one machine, identical parallel machines (i.e., for a fixed job j and for each machine i, $p_{i,j}$ equals a value p_j that is independent of i), unrelated machines ($p_{i,j}$'s are dependent on both job i and machine j), and a fixed number m (not part of the input) of unrelated machines. The field β takes on the values r_j which indicates that the jobs have release dates, and the value *pmtn* which indicates that preemption of jobs is permitted. Further, the value *prec* in the field β indicates that the problem may involve precedence constraints between jobs which poses further restrictions on the schedule. The field γ is either $\sum w_j C_j$ or $\sum C_j$ which denote total weighted and total (unweighted) completion times respectively.

Some of the simpler classes of the weighted completion time scheduling problems admit optimal polynomial time solutions. They include the problem $P \mid \mid \sum C_j$ for which the *shortest-job-first* strategy is optimal, the problem $1 \mid \mid \sum w_j C_j$ for which Smith's rule [13] (scheduling jobs in their non-decreasing order of $\frac{p_j}{w_j}$ values) is optimal, and the problem $R \mid \mid \sum C_j$ which can be solved via matching techniques [2, 9]. With the introduction of release dates, even the simplest classes of the weighted completion time minimization problem becomes strongly NP-hard. In this article, we focus on the work of Afrati et al. [1] whose main contribution is the design of polynomial-time approximation schemes (PTASs) for several classes of scheduling problems to minimize weighted completion time *with release dates*. Prior to this work, the best solutions for minimizing weighted completion time with release dates were all $O(1)$-approximation algorithms (for example, [4, 5, 11]); the only known PTAS for a strongly NP-Hard problem involving weighted completion time was

[*]The research of V. S. Anil Kumar and M. V. Marathe was supported in part by NSF Award CNS-0626964. A. Srinivasan's research was supported in part by NSF Award CNS-0626636.

due to Skutella and Woeginger [12] who developed a PTAS for the problem $P \mid \mid \sum w_j C_j$. For an excellent survey on the minimum weighted completion time problem, we refer the reader to Chekuri and Khanna [3].

2 KEY RESULTS

Afrati et al. [1] were the first to develop PTASs for weighted completion time problems involving release dates. We now summarize the running times of these PTASs in Table 1.

Problem	**Running Time of PTAS**
$1 \mid r_j \mid \sum w_j C_j$	$O(2^{poly(\frac{1}{\epsilon})} n + n \log n)$
$P \mid r_j \mid \sum w_j C_j$	$O((m+1)^{poly(\frac{1}{\epsilon})} n + n \log n)$
$P \mid r_j, pmtn \mid \sum w_j C_j$	$O(2^{poly(\frac{1}{\epsilon})} n + n \log n)$
$Rm \mid r_j \mid \sum w_j C_j$	$O(f(m, \frac{1}{\epsilon}) poly(n))$
$Rm \mid r_j,\ pmtn \mid \sum w_j C_j$	$O(f(m, \frac{1}{\epsilon}) n + n \log n)$
$Rm \mid \mid \sum w_j C_j$	$O(f(m, \frac{1}{\epsilon}) n + n \log n)$

Table 1: Summary of results of Afrati et al. [1]

The results presented in Table 1 are obtained through a careful sequence of input-transformations followed by dynamic programming. The input-transformations ensure that the input becomes *well structured* at a slight loss in optimality, while dynamic programming allows efficient enumeration of all the near-optimal solutions to the well structured instance.

The first step in the input transformation is *geometric rounding*, in which the processing times and release dates are converted to powers of $1+\epsilon$, with at most $1+\epsilon$ loss in the overall performance. More significantly, this step (i) ensures that there are only a small number of distinct processing times and release dates to deal with; (ii) allows time to be broken into geometrically increasing intervals, and (iii) aligns release dates with start and end-times of intervals. These are useful properties that can be exploited by dynamic programming.

The second step in the input transformation is *time stretching*, in which small amounts of idle time are added throughout the schedule. This step also changes completion times by a factor of at most $1 + O(\epsilon)$, but is useful for *cleaning up* the scheduling. Specifically, if a job is *large* (i.e., occupies a large portion of the interval where it executes), it can be pushed into the idle time of a later interval where it is small. This ensures that most jobs have small sizes compared to the length of the intervals where they execute, which greatly simplifies schedule computation.

The next step is *job shifting*. Consider a partition of the time interval $[0, \infty)$ into intervals of the form $I_x = [(1+\epsilon)^x, (1+\epsilon)^{x+1})$, for integral values of x. The job shifting step ensures that there is a slightly suboptimal schedule in which every job j gets completed within $O(\log_{1+\epsilon}(1+\frac{1}{\epsilon}))$ intervals after r_j. This has the following nice property: if we consider blocks of intervals $\mathcal{B}_0, \mathcal{B}_1, \ldots$, with each block $\mathcal{B}_i$ containing $O(\log_{1+\epsilon}(1+\frac{1}{\epsilon}))$ consecutive intervals then a job j starting in block $\mathcal{B}_i$ completes within the next block. Further, the other steps in the job shifting phase ensure that there are not too many *large* jobs which spill over to the next block; this allows the dynamic programming to be done efficiently.

The precise steps in the algorithms and their analysis are subtle, and the above description is clearly an over-simplification. We refer the reader to [1] or [3] for further details.

3 APPLICATIONS

A number of optimization problems in parallel computing and operations research can be formulated as machine scheduling problems. When precedence constraints are introduced between jobs, the weighted completion time objective can generalize the more commonly studied makespan objective, and hence is important.

4 OPEN PROBLEMS

Some of the major open problems in this area are to improve the approximation ratios for scheduling on unrelated or related machines for jobs with precedence constraints. The following problems in particular merit special mention. The best known solution for the $1 \mid prec \mid \sum w_j C_j$ problem is the 2-approximation algorithm due to Hall et al. [8]; improving upon this factor is a major open problem in scheduling theory. The problem $R \mid prec \mid \sum_j w_j C_j$ in which the precedence constraints form an arbitrary acyclic graph is especially open - the only known results in this direction are when the precedence constraints form chains [6], or trees [10].

The other open direction is inapproximability - there are significant gaps between the known approximation guarantees and hardness factors for various problem classes. For instance, the $R \mid \mid \sum w_j C_j$ and $R \mid r_j \mid \sum w_j C_j$ are both known to be APX-hard, but the best known algorithms for these problems (due to Skutella [11]) have approximation ratios of $\frac{3}{2}$ and 2 respectively. Closing these gaps remain a significant challenge.

5 EXPERIMENTAL RESULTS

None is reported.

6 DATA SETS

None is reported.

7 URL to CODE

None is reported.

8 CROSS REFERENCES

See chapters on Flow Time Minimization, List Scheduling, Minimum Flow Time, Minimum Makespan on Unrelated Machines, Online Load Balancing for other scheduling models and algorithms.

9 RECOMMENDED READING

References

[1] Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In Proc. of *Foundations of Computer Science*, pages 32–44, 1999.

[2] 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.

[3] C. Chekuri and S. Khanna. *Approximation algorithms for minimizing weighted completion time, Handbook of Scheduling.* 2004.

[4] Chandra Chekuri, Rajeev Motwani, B. Natarajan, and Clifford Stein. Approximation techniques for average completion time scheduling. *SIAM J. Comput.*, 31(1):146–166, 2001.

[5] M. Goemans, M. Queyranne, A. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics*, 15:165192, 2002.

[6] L.A. Goldberg, M. Paterson, A. Srinivasan and E. Sweedyk. *Better approximation guarantees for job-shop scheduling.* SIAM Journal on Discrete Mathematics, Vol. 14, 67-92, 2001.

[7] R.L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, pages 287–326, 1979.

[8] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, 1997.

[9] W. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21:846–847, 1973.

[10] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy and Aravind Srinivasan. Scheduling on unrelated machines under tree-like precedence constraints. In *APPROX-RANDOM*, pages 146–157, 2005.

[11] M. Skutella. Convex quadratic and semidefinite relaxations in scheduling. *Journal of the ACM*, 46(2):206–242, 2001.

[12] M. Skutella and G.J. Woeginger. A PTAS for minimizing the weighted sum of job completion times on parallel machines. In Proc. of 31^{st} *Annual ACM Symposium on Theory of Computing (STOC '99)*, pages 400-407, 1999.

[13] W. E. Smith. Various optimizers for single-stage production. *Nav. Res. Log. Q.*, pages 59–66, 1956.