Algorithms for Total Weighted Completion Time Scheduling

Ivan D. Baev* Waleed M. Meleis* Alexandre Eichenberger†

Abstract

We consider the total weighted completion time scheduling problem for parallel identical machines and precedence constraints, $P|prec|\sum w_i C_i$. We describe a family of natural scheduling algorithms that optimally solve the single machine problem, and show that they can be used to achieve good performance for the multiple machine problem. These algorithms are efficient and find schedules that are on average within 1.9% of optimal over a large synthetic benchmark.

We then consider the class of linear programming relaxations that have recently been successfully used to produce approximation algorithms for scheduling problems. We describe a new linear programming relaxation for the multiple machine problem that finds schedules that are within 0.39% of optimal over our benchmark.

This research represents the first experimental evaluation of various algorithms for the general WCT problem, and represents an extension of the results in Savelsbergh(1998) to multiple machines and in-tree, chains, and empty precedence constraints.

1 Introduction

A basic scheduling problem is $P|prec|\sum w_i C_i$, the problem of scheduling n jobs under precedence constraints on m identical parallel machines to minimize total weighted job completion time. Each job i has a positive weight w_i and processing time p_i, $i = 1, \ldots, n$. The start time of job i in a schedule is denoted S_i and the completion time is denoted C_i. A directed acyclic graph describes the precedence constraints among the jobs such that an edge from job i to job j in the graph implies that $C_i \leq S_j$. In a feasible schedule, no more than one job executes on any machine at any time, each job is scheduled nonpreemptively, and the precedence constraints are satisfied. The goal of the total weighted completion time (WCT) problem is to find a feasible schedule of the n jobs that minimizes $\sum w_i C_i$. The WCT problem is NP-hard, even for fixed $m = 2$ and the empty precedence graph, but it can be solved efficiently if $m = 1$ for certain classes of precedence constraints.

2 Algorithms

The following algorithms each consist of two stages. They first assign priorities to the jobs, and then use list scheduling to assign each job a start time.

*Department of ECE, Northeastern University, Boston MA, 02115 (`baev,meleis@ece.neu.edu`).

†Department of ECE, NC State University, EGRC, Box 7914, Raleigh, NC 27695 (`alexe@eos.ncsu.edu`).

The LRF algorithm

1. The priority of every job i is $\rho_i = w_i/p_i$.
2. Apply list scheduling.

The following algorithm described by Sidney [2] applies if the precedence constraints form a set of r chains. We let an ordered pair (i, j) represent a job, where i is the chain number, j is the job's position within the chain, and n_i is the number of jobs in the ith chain. We also define $Pred(i, j) = \{(i, k) | 1 \leq k \leq j\}$. We let C_i equal the set of jobs in the ith chain. For a set of jobs U we define the ratio $\rho(U) = \sum_{i \in U} w_i / \sum_{i \in U} p_i$.

The ρ-max algorithm for chains

1. Begin with an empty priority list β.
2. Repeat the following until the chains are all empty:
   (a) For every nonempty chain C_i, let $\delta_i = \max_j\{\rho(Pred(i, j))\}$.
   (b) Let $\delta = \max_i\{\delta_i\}$, and let i^* and j^* be the largest values such that $\rho(Pred(i^*, j^*) = \delta$.
   (c) Add the jobs in $Pred(i^*, j^*)$ in order to the end of β, and remove these jobs from C_{i^*}.
3. Apply list scheduling, using β as the priority list.

A variation of this chain algorithm can also be used to find optimal schedules when the precedence graph is a tree where each job has at most one immediate successor (i.e. an intree) [2]. Jobs that have more than one immediate predecessor are called *branch jobs*, and a branch job that does not have a direct or indirect predecessor that is a branch job is called an *initial branch job*.

The ρ-max algorithm for trees

1. Repeat the following until there are no remaining branch jobs:
   (a) Select an initial branch job i.
   (b) Use the chain algorithm to optimally order the jobs in $Pred(i)$ and set β equal to this order.
   (c) Add a precedence arc (j, k) to the precedence graph if job j precedes job k in β.

	prec	No precedence			Chains			Trees		
	m	2	4	6	2	4	6	2	4	6
LRF	mean	0.01	0.03	0.06	8.80	8.62	8.25	11.20	14.34	13.07
	max	0.09	0.31	0.42	18.54	18.22	18.43	31.25	30.80	26.50
ρ-max	mean	0.01	0.03	0.06	0.19	0.45	0.55	2.33	6.55	7.31
	max	0.09	0.31	0.42	9.57	6.66	4.59	11.32	17.45	16.76
LP-based	mean	0.00	0.00	0.00	0.11	0.15	0.11	1.29	1.05	0.76
	max	0.00	0.03	0.04	1.56	1.96	1.03	7.04	3.46	3.67

Table 1: Quality of LRF, ρ-max and LP-based algorithms vs. lower bound for large instances ($1.0 = 1\%$).

2. Create a priority list based on the total ordering of jobs, and apply list scheduling.

The tree algorithm applies the chain algorithm to initial branch jobs, replacing two chains with a single chain. This entire procedure can be implemented to run in $O(n\,log n)$ time.

The Linear Programming-based algorithm

The WCT problem is implemented using a straightforward integer programming formulation where the binary variable $x_{n,t}$ equals 1 if job n has a start time of t. We solve the relaxation of this formulation and interpret non-integral variable values as indicating that a job's start time is distributed over multiple time periods. A priority list is created based on the average start time of each job, and list scheduling is used to create a feasible schedule.

3 Experiments

We evaluate the algorithms described in the previous section by applying them to a set of 160 synthetic WCT instances. An instance consists of a set of n jobs, each with a processing time and a weight, a precedence graph on the jobs, and a set of m machines. Instances range in size from $n = 35$ to $n = 160$. The processing time of each job is uniformly distributed in $[1, 5]$, and the weight of each job is uniformly distributed in $[1, 100]$. Twenty random instances are generated for each n, with $m = 2, 4$, or 6 for each. Precedence constraints for each instance are randomly selected such that every unique graph is equally likely to occur.

We compute solutions for each instance using the CPLEX integer programming solver running on an array of 166MHz Sun 4 workstations with 128Mb of memory. We use the relaxed (nonintegral) version of the IP formulation to compute a lower bound on the cost of the optimal schedules for the instances. We then apply the LRF and ρ-max algorithms to the instances. The percentage difference between the heuristics and the lower bound is shown in Table 1.

The results indicate that without precedence, the LRF algorithm finds schedules that are on average 0.03% from the lower bound. When this algorithm is applied to chains and trees, the average difference rises to 8.56% and 12.87%, respectively, for the large instances. In contrast the equivalent differences for the ρ-max algorithms are .40% and 5.40%. The ρ-max algorithms, which give optimal results for the WCT problem with $n = 1$, reduce the average cost for chain instances by 95% and the average cost of tree instances by 58% compared to LRF over the entire benchmark.

The LP-based algorithm uses the non-integral solution the the relaxed IP formulation to construct the priority list. This algorithm finds schedules that are on average .39% above the lower bound. Much of this difference can be attributed to the gap between the lower bound and the optimal value. We note that over all our experiments there is no straightforward connection between the number of machines, $m \geq 2$, and the quality of the schedules.

4 Discussion

Our results indicate that the quality of the multiple machine schedules is closely related to the quality of the single machine priority list, and that it gets harder to achieve good results as the complexity of the precedence constraints increases. The good results produced by the LP-based algorithm suggest that such approaches may be useful for solving other difficult scheduling problems. Extensive research has been done on a range of LP relaxations, and these formulations may allow even better results to be obtained.

References

[1] M. W. P. Savelsbergh, R. N. Uma, and J. Wein. An experimental study of LP-based approximation algorithms. In *Proc. 9th Annual ACM-SIAM Symp. on Disc. Algorithms*, pages 453–462, 1998.

[2] J. B. Sidney. Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research*, 23:283–298, 1975.