An Approximation Algorithm for a Scheduling Problem

Tomás Feder[*] Rajeev Motwani[†]

July 24, 2008

1 Introduction

We consider the following scheduling problem. We are given a complete graph with n vertices $v_1, \ldots, v_n$ corresponding to n tasks to be completed. Each vertex v_i has a weight $w_i \geq 0$ indicating the time it takes to complete task i. In addition, we allow pairs of tasks i, j to be completed simultaneously, in time $0 \leq u_{ij} \leq w_i + w_j$.

The first question to be addressed is how to schedule tasks, pairing some and scheduling the remaining ones separately, to minimize the total completion time. This problem can be solved exactly in polynomial time as follows. Add n tasks $v_{n+1}, \ldots, v_{2n}$ with $w_j = 0$ and $u_{jj'} = 0$ among these, $n+1 \leq j, j' \leq 2n$, and for $1 \leq i \leq n$, let $u_{ij} = w_i$. The problem then reduces to finding a maximum matching of minimum weight, which can be solved in polynomial time.

The main question of this paper concerns the average completion time. Here we may add extra vertices as before, and assume the tasks will all be completed in pairs, for a total of n pairs. If the new problem with $2n$ tasks has optimal average completion time T', and the original problem has optimal average completion time T, then we have $T \leq T'$, as a solution to the new problem gives a less expensive solution to the original problem by ignoring the added tasks. In the other direction, we have $T' \leq 2T$, as scheduling an i task with a j task together adds for the j task the same time as for the i task, costing a factor of two.

We thus focus on the latter problem, which consists of finding a matching M with n edges on $2n$ vertices, so that if the edges in the matching M have weights $s_1 \geq s_2 \geq \cdots \geq s_n$, then the average completion time is S/n, where $S = \sum_{1 \leq i \leq n} i s_i$. The aim is thus to minimize S.

2 Approximation Algorithms

We consider first a special case.

[*]268 Waverley St., Palo Alto, CA 94301. E-mail: tomas@theory.stanford.edu.

[†]Computer Science Department, Stanford University, Stanford, CA 94305. E-mail: rajeev@cs.stanford.edu.

Theorem 1 *In the modified average completion time problem, suppose all u_{ij} are either a, b, or infinity, for $a < b$. Then the modified problem can be solved exactly in polynomial time, so the original average completion time problem can be approximated within a factor of two in polynomial time.*

PROOF: Remove the edges of infinite weight u_{ij}, and look for a perfect matching in the resulting graph that minimizes the total weight. This will maximize the number of task pairs finished in time a versus b. ■

We give an approximation algorithm for the modified problem.

Theorem 2 *In the modified average completion time problem, suppose all u_{ij} are $a_1 < a_2 < \cdots < a_k$ or infinity. Then the modified problem (and thus the original average completion time problem) can be approximated within a factor of $O(k^2)$ in time $n^{O(1)} \log^k n$. This gives without assumption on k a $\log^{2+1/c} n$ approximation in polynomial time $n^{O(c)}$ for any constant $c \geq 1$.*

PROOF: Suppose in the optimal solution we have d_i matched edges taking value a_i. If we let $t_i = d_1 + d_2 + \cdots d_i$, then the optimum is at least

$$\sum_{1 \leq i \leq k} d_i t_i a_i / 2.$$

Now replace all a_i by $1/d_i$, and find a matching of minimum cost

$$s_1 + s_2 + \cdots + s_n.$$

The optimal solution above will give cost at most k, as there are d_i values $1/d_i$. Therefore in the optimal solution $1/d_i$ will occur at most kd_i times. Back in the original problem, this optimum gives cost at most

$$\begin{aligned} &\sum 1 \leq i \leq k(kd_i)(kt_i)a_i \\ &= 2k^2 \sum_{i \leq i \leq k} d_i t_i a_i / 2, \end{aligned}$$

a factor of $2k^2$ from the optimum. As we must guess k values d_i from 1 to n, the running time is $n^{k+O(1)}$. If we choose $d_i = 2^j$, we only loose a constant factor in the approximation and the running time is improved to $(\log n)^k$ as j ranges from 1 to $\log n$. Now, if the largest weight to be used in a solution is W, we may assume that all weights at most W/n^2 are just W/n^2, and cost only a constant factor in the total cost. We may then use $k = 3c \log n / \log\log n$ values a_i by rounding to the nearest W/b^i, where $b = (n^2)^{1/k} = (\log n)^{1/c}$, giving in polynomial time $n^{O(c)}$ $bk^2 \leq \log^{2+1/c} n$ approximation. ■