Operations Research Letters 39 (2011) 423–427

Contents lists available at SciVerse ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

An improved monotone algorithm for scheduling related machines with precedence constraints

Anke van Zuylen

Max Planck Institute for Informatics, Campus E 1 4, 66123 Saarbrücken, Germany

ARTICLE INFO

Article history:
Received 17 July 2009
Accepted 28 July 2011
Available online 5 August 2011

Keywords:
Scheduling
Algorithmic mechanism design
Precedence constraints
Monotone algorithms

ABSTRACT

We answer an open question posed by Krumke et al. (2008) [6] by showing how to turn the algorithm of Chekuri and Bender for scheduling related machines with precedence constraints into an $O(\log m)$-approximation algorithm that is monotone in expectation. This significantly improves on the previously best known monotone approximation algorithms for this problem, from Krumke et al. [6] and Thielen and Krumke (2008) [8], which have an approximation guarantee of $O(m^{2/3})$.

1. Introduction

In this paper we consider the problem of scheduling jobs on related machines when the machines are controlled by selfish agents.

The classical version of the problem, in which the machines are not controlled by selfish agents, is called $Q|\text{prec}|C_{\max}$ in the notation introduced by Graham et al. [4]. We are given a set of n jobs with processing times $p_1, \ldots, p_n$, precedence constraints on the jobs in the form of a partial order, and m machines with speeds $s_1, \ldots, s_m$. If job j is processed on machine i, it takes p_j/s_i time units to process. A schedule is an assignment of jobs to machines, plus a starting time for each job, so that a job starts after all jobs that must precede it have been processed, and a machine works on only one job at a time. The goal is to minimize the makespan of the schedule, i.e. the finishing time of the last job.

Motivated by applications on the Internet, in recent years researchers have considered scheduling problems in which the machines are controlled by selfish agents. In our setting, this means that we do not know the speeds of the machines. Our goal is to design a mechanism which asks the agents for their speeds, assigns the jobs to the agents so as to minimize the makespan, and computes payments to pay each agent. Each agent, on the other hand, is trying to maximize her profit, which is equal to the payment received minus the time it takes to process the jobs assigned to the agent. Hence, agents may choose to misrepresent their speed. A *truthful* mechanism is a mechanism in which reporting the true speed is a dominant strategy for each agent, i.e. no matter what the speeds are that are reported by the other agents, the strategy which yields the maximum profit for a given agent is to report its true speed.

The problem that we consider belongs to the class of mechanism design problems for one-parameter agents. For a problem in this class, there are m agents, and each agent i holds some piece of private data consisting of a single parameter $t_i \in \mathbb{R}_{\geq 0}$. All other input is public knowledge. The vector $t = (t_1, \ldots, t_n)$ combined with the rest of the input defines a (classical) optimization problem, for which a feasible solution takes the form of a set of loads $w_i(t)$ to be assigned to the agents. A mechanism for a problem in this class solicits a bid b_i from each agent, and based on $b = (b_1, \ldots, b_m)$, it computes feasible loads $w_i(b)$ and payments $p_i(b)$ for agents $i = 1, \ldots, m$. The goal of agent i is to maximize $\text{profit}_i = p_i(b) - w_i(b)t_i$. Let b_{-i} be the bids of the other agents, and let $b = (b_{-i}, b_i)$ denote the vector containing all bids. The mechanism is truthful if for each agent i, reporting her true private value t_i dominates any other bid b_i, i.e., $\text{profit}_i(b_{-i}, t_i) \geq \text{profit}_i(b_{-i}, b_i)$, for all b_{-i}, b_i.

It was shown by Archer and Tardos [1] that a necessary and sufficient condition for obtaining a truthful mechanism for this class of mechanism design problems is to have a *monotone* algorithm for the underlying problem, in which t is part of the known input. An algorithm is monotone if $w_i(t_{-i}, t_i)$ is a non-increasing function of t_i. Given such an algorithm, Archer and Tardos show how to compute payments that induce truth telling. In our setting, an agent's private information is $t_i = \frac{1}{s_i}$, and a monotone algorithm means that the amount of work assigned to a machine should not increase if the speed of the machine is decreased (and all other input remains the same). Since the problem $Q|\text{prec}|C_{\max}$ is NP-hard, we are concerned with finding monotone approximation algorithms for $Q|\text{prec}|C_{\max}$. Combined

E-mail address: anke@mpi-inf.mpg.de.

0167-6377/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.orl.2011.07.009

with the payment scheme of [1], a monotone α-approximation algorithm results in a truthful mechanism that finds a schedule with makespan at most α times optimal.

The problem of finding a monotone approximation algorithm for $Q|\text{prec}|C_{\max}$ was first considered by Krumke et al. [6]. They showed how to turn an $O(\sqrt{m})$-approximation algorithm of Jaffe [5] into a monotone $O(m^{2/3})$-approximation algorithm. Their approach was generalized by Thielen and Krumke [8] to a general scheme for designing monotone algorithms. Their scheme also yields a monotone algorithm for the problem $Q|\text{prec}, r_j|C_{\max}$, in which the existence of a job j is unknown until its release time r_j. The best known (not necessarily monotone) approximation algorithms for $Q|\text{prec}|C_{\max}$ and $Q|\text{prec}, r_j|C_{\max}$ are $O(\log m)$-approximation algorithms of Chekuri and Bender [2] and Chudak and Shmoys [3]. An open question posed by Krumke et al. [6] is whether one can get better monotone approximation algorithms using these approaches.

In this paper, we slightly weaken the monotonicity requirement and answer the question of Krumke et al. affirmatively under this weakened definition. In particular, we show that a slightly modified and randomized version of the algorithm of Chekuri and Bender [2] is an $O(\log m)$-approximation algorithm that is monotone *in expectation*: the *expected* amount of work assigned to a machine does not increase if the machine's speed is decreased. Using the result of Archer and Tardos [1], this implies a mechanism that is truthful in expectation: an agent cannot improve its expected payoff by being untruthful. Using a result of Shmoys et al. [7], it is straightforward to extend our approach to $Q|\text{prec}, r_j|C_{\max}$ and obtain an $O(\log m)$-approximation algorithm that is monotone in expectation.

2. The problem definition

In the problem $Q|\text{prec}|C_{\max}$, we are given a set of n jobs with processing times $p_1, \ldots, p_n$, precedence constraints on the jobs in the form of a partial order $\prec$, and machines with speeds $\{s_1, \ldots, s_m\}$. If job j is processed on machine i, it takes p_j/s_i time units to process. A schedule is an assignment of jobs to machines, plus a starting time for each job, so that a job k starts after all jobs j such that $j \prec k$ have been processed, and a machine works on only one job at a time. We assume without loss of generality that $s_1 \geq s_2 \geq \cdots \geq s_m$. The makespan of a schedule is the time when the last job is finished. We will denote by $C^*_{\max}$ the minimum makespan over all feasible schedules.

We define the amount of work assigned to machine i as the sum of p_j over all jobs j that are assigned to machine i. An algorithm for $Q|\text{prec}|C_{\max}$ is monotone if the following holds: if we consider two instances that are identical except for the speed of machine i, which is s in the first instance and $s' > s$ in the second instance, then the amount of work assigned to machine i in the schedule produced for the second instance is not smaller than the amount of work assigned to it in the first schedule. A randomized algorithm is monotone in expectation if the expected amount of work assigned to machine i in the second instance is not smaller than the expected amount of work assigned to it in the first schedule.

The problem $Q|\text{prec}, r_j|C_{\max}$ is defined similarly to $Q|\text{prec}|C_{\max}$, except that the existence of a job is not known until the time r_j when it is released.

3. The Chekuri–Bender algorithm

We begin by describing the algorithm proposed by Chekuri and Bender [2]. We will need to make only a few changes to achieve monotonicity, which we will describe in the next section.

The algorithm of Chekuri and Bender [2] begins by computing a lower bound B on the makespan of any feasible schedule. To do so, they find a maximal chain decomposition of the jobs: a chain P is a subset of jobs $j_1, \ldots, j_k$ such that $j_i \prec j_{i+1}$ for all $i \in \{1, \ldots, k-1\}$. The length of P is denoted by $|P| = \sum_{i=1}^{k} p_{j_i}$. A chain decomposition is a partition of the precedence order into an ordered collection of chains $(P_1, \ldots, P_r)$ such that P_1 is a longest chain and $(P_2, \ldots, P_r)$ is a maximal chain decomposition with the jobs in P_1 removed. If we consider the first k chains in the chain decomposition, then at any point in time at most k different machines are working on the jobs in these chains. The total speed of these k machines is at most $\sum_{i=1}^{k} s_i$ where we recall that speeds are assumed to be ordered so that $s_i \geq s_{i+1}$. Hence there must be some job in these k chains which completes at $\frac{\sum_{i=1}^{k} |P_i|}{\sum_{i=1}^{k} s_i}$ or later. This observation gives rise to the following lower bound on the makespan:

$$B = \max \left\{ \frac{\sum_{j=1}^{n} p_j}{\sum_{i=1}^{m} s_i}, \max_{1 \leq k \leq \min\{r,m\}} \left\{ \frac{\sum_{i=1}^{k} |P_i|}{\sum_{i=1}^{k} s_i} \right\} \right\} \leq C^*_{\max}.$$

The Chekuri–Bender algorithm uses an idea of Chudak and Shmoys [3] to reduce an instance with an arbitrary number of speeds to an instance with only $K = O(\log m)$ speeds while losing only a constant factor in the approximation ratio: ignore machines with speed less than $\frac{1}{m}$ times the speed of the fastest machine, and divide the remaining machine speeds into speed classes, by rounding down all speeds to the nearest power of 2.

Let $\bar{s}_1, \ldots, \bar{s}_K$ be the remaining distinct speeds. The algorithm iterates through the chains in the maximal chain decomposition and assigns them to the speed classes. See Fig. 1 for a description of the algorithm Chain-Alloc that allocates the chains. For a given assignment, let $k(j)$ be the speed class that (the chain containing) job j is assigned to. Given the output of Chain-Alloc, the jobs are scheduled according to speed-based list scheduling [3]: if a job j is available and there is a free machine i such that machine i is in speed class $k(j)$, then schedule job j on machine i.

We repeat a key point in the analysis by Chekuri and Bender, which we will be using in the next section to show monotonicity. The following lemma is (contained in) Lemma 2 in [2].

Lemma 1 (*Chekuri and Bender [2]*). *Let $P_{\ell(k)}, \ldots, P_r$ be the chains remaining when Chain-Alloc considers speed k. Then $|P_{\ell(k)}|/\bar{s}_k \leq 2B$.*

4. A monotone algorithm

Recall that our goal is to show that if a machine changes its speed from s to $s' > s$, then the amount of work assigned to it does not decrease. We begin by analyzing the original Chekuri–Bender algorithm. Our analysis will highlight certain cases in which monotonicity is not guaranteed, and we show two small adaptations which do ensure monotonicity.

Let m_k be the number of machines with rounded speed $\bar{s}_k$, and define

$$D_k = \frac{1}{m_k \bar{s}_k} \sum_{j:k(j)=k} p_j.$$

We will assume that in the speed-based list scheduling phase of the algorithm, all machines in one speed class are indistinguishable for the algorithm. To be precise, we simulate the speed-based list scheduling where all machines have speed equal to their true speed rounded down to the nearest power of 2, and machines within one speed class are considered in random order. This gives an assignment of jobs to machines, plus a starting time for each job, which is feasible for the real instance. The expected amount of work assigned to a machine in speed class k is equal to $D_k \bar{s}_k$.

The following lemma allows us to bound the expected amount of work assigned to a machine.