New Hardness Results for Congestion Minimization and Machine Scheduling

JULIA CHUZHOY
CSAIL MIT and Dept. of CIS, University of Pennsylvania
and
JOSEPH (SEFFI) NAOR
Computer Science Dept., Technion

We study the approximability of two natural NP-hard problems. The first problem is *congestion minimization* in directed networks. In this problem, we are given a directed graph and a set of source-sink pairs. The goal is to route all the pairs with minimum congestion on the network edges. The second problem is *machine scheduling*, where we are given a set of jobs, and for each job, there is a list of intervals on which it can be scheduled. The goal is to find the smallest number of machines on which all jobs can be scheduled such that no two jobs overlap in their execution on any machine. Both problems are known to be $O(\log n/\log\log n)$-approximable via the randomized rounding technique of Raghavan and Thompson. However, until recently, only Max SNP hardness was known for each problem. We make progress in closing this gap by showing that both problems are $\Omega(\log\log n)$-hard to approximate unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log\log\log n)})$.

Categories and Subject Descriptors: G.2.2 [**Discrete Mathematics**]: Graph Theory—*Network problems*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*

General Terms: Theory

Additional Key Words and Phrases: Hardness of approximation, congestion minimization, network routing, scheduling, resource minimization

1. INTRODUCTION

In this paper we study the approximability of the congestion minimization and the machine scheduling problems. Both these problems deal with resource minimization, though in different contexts: congestion minimization aims at routing source-sink pairs in a network, so as to minimize edge congestion, while in the machine scheduling problem the goal is to schedule jobs on machines, while minimizing the number of machines used. Congestion minimization and machine scheduling are probably the most natural resource minimization problems in their respective areas (network routing / scheduling), and each one of the two problems is associated

Julia Chuzhoy, Computer Science and Artificial Intelligence Lab, MIT and Dept. of Computer and Information Science, University of Pennsylvania. Email: cjulia@csail.mit.edu. Work done while the author was a graduate student at the Computer Science Department at the Technion.
Joseph (Seffi) Naor, Computer Science Dept., Technion, Haifa 32000, Israel. E-mail: naor@cs.technion.ac.il. Research supported in part by the United States-Israel Binational Science Foundation Grant No. 2002-276 and by EU contract IST-1999-14084 (APPOL II).

with a rich and well-studied family of closely related problems. Though congestion minimization and machine scheduling might seem very different at the first sight, it turns out that there is an interesting connection between them, which is exploited in our inapproximability results.

In the congestion minimization problem, we are given a graph (directed or undirected) and a collection of source-sink pairs. The goal is to find the smallest integer $c \geq 1$ such that every input source-sink pair can be connected by a simple path, while the number of paths passing through any edge does not exceed c. The congestion minimization problem can be relaxed by formulating it in a natural way as a multicommodity flow linear program. A classical result of Raghavan and Thompson [Raghavan and Thompson 1987] shows that a randomized rounding of the multicommodity flow relaxation yields an approximation factor of $O(\log n/\log\log n)$ for both undirected and directed graphs, where n denotes the number of vertices. This is the best currently known approximation factor for congestion minimization, and to the best of our knowledge, no non-trivial hardness of approximation results have been known prior to this work.

In the machine scheduling problem, we are given a set of n jobs and for each job a set of time intervals is specified. A job is scheduled by choosing one of its associated time intervals. The goal is to schedule all the jobs using the minimum number of machines, such that no two jobs assigned to a machine overlap in time. There are two popular versions of machines scheduling: In the *discrete version*, the intervals belonging to each job are listed explicitly; in the *continuous version*, for each job j, a processing time p_j, a release date r_j and a deadline d_j are given, and job j can be scheduled on any time interval of length p_j that lies between its release date and deadline. The best currently known approximation factor for the discrete machine scheduling problem is $O(\log n/\log\log n)$, and it is achieved via the randomized rounding technique of Raghavan and Thompson [Raghavan and Thompson 1987]. However, for the continuous version of the problem, Chuzhoy et al. [Chuzhoy et al. 2004] obtained an $O\left(\sqrt{\log n}\right)$ approximation. They also showed a constant approximation for the special case of the continuous version, where the optimal solution uses a constant number of machines.

1.1 Our Results

We prove hardness of approximation for congestion minimization in directed networks and for the machine minimization problem with discrete input. For both problems, we show that there is no $c\log\log n$-approximation algorithm for some constant c, unless $NP \subseteq \mathsf{DTIME}(n^{O(\log\log\log n)})$. The hardness of discrete machine scheduling holds for the special case where all the jobs can scheduled on a single machine. Our results give the first non-trivial inapproximability bounds for the well-studied classical problem of congestion minimization. Subsequent to our work, Andrews and Zhang [Andrews and Zhang 2005b] proved an $\Omega\left(\log\log^{1-\epsilon} n\right)$-hardness of undirected congestion minimization, under the assumption that NP is not contained in ZPTIME $\left(n^{\text{poly}\log n}\right)$. Recently, Andrews and Zhang [Andrews and Zhang 2006] showed that directed congestion minimization is $\Omega(\log^{1-\epsilon} n)$-hard to approximate unless $\mathsf{NP} \subseteq$ ZPTIME $\left(n^{\text{poly}\log n}\right)$. This was improved by Chuzhoy and Khanna [Chuzhoy and Khanna 2006] to $\Omega\left(\frac{\log n}{\log\log n}\right)$-hardness, under the same complexity assumption, thus resolving the approximability of this problem up to a constant factor. We note that these results do not extend to the machine scheduling problem.

1.2 Related Problems

The edge disjoint paths problem (EDP) is closely related to congestion minimization. EDP is defined exactly like the congestion minimization problem, except that now, instead of routing all the source-sink pairs while minimizing the congestion, the objective is to connect the maximum number of source-sink pairs via edge-disjoint paths (i.e., no congestion is allowed). This is a fundamental problem, extensively studied, and the best known approximation factors are $\tilde{O}(\min(n^{2/3}, \sqrt{m}))$ for directed graphs [Chekuri and Khanna 2003; Kleinberg 1996; Kolliopoulos and Stein 1998; Srinivasan 1997; Varadarajan and Venkataraman 2004], and $O(\sqrt{n})$ for undirected and directed acyclic graphs [Chekuri et al. 2006b] (here n and m denote the number of vertices and edges respectively in the input graph). In directed graphs, this is matched by an $\Omega(m^{\frac{1}{2}-\epsilon})$-hardness due to Guruswami *et al.* [Guruswami et al. 2003; Chekuri and Khanna 2003]. For undirected EDP, Andrews and Zhang [Andrews and Zhang 2005c] showed an $\Omega(\log^{\frac{1}{3}-\epsilon} n)$ hardness, and this has recently been improved to $\Omega(\log^{\frac{1}{2}-\epsilon} n)$ [Chuzhoy and Khanna 2005].

An interesting problem, which can be seen as lying between the EDP and the congestion minimization problems, is EDP with congestion (EDPwC). This problem is defined exactly like EDP, except that an integer c bounding the allowed congestion is given as part of the input. The goal is again to route the maximum possible number of source-sink pairs, while the number of paths using any edge is restricted to be at most c. Notice that in this problem, the objective function is the same as in EDP - maximizing the number of routed source-sink pairs, while some fixed allowed congestion c is given as a part of problem input. The performance of approximation algorithms for EDPwC is usually measured with respect to the optimal solution with congestion 1. On the positive side, for constant congestion $c \geq 2$, an $O(n^{1/c})$ approximation is known [Azar and Regev 2001; Baveja and Srinivasan 2000; Kolliopoulos and Stein 1998]. When the congestion is allowed to be $O(\log n/\log\log n)$, we get a constant approximation via randomized rounding [Raghavan and Thompson 1987]. For planar graphs, there is an $O(\log n)$-approximation when congestion 2 is allowed [Chekuri et al. 2004; 2005], and a constant approximation with congestion 4 [Chekuri et al. 2006a]. On the negative side, it has been shown by [Andrews and Zhang 2005a; Chuzhoy and Khanna 2005; Guruswami and Talwar 2005; Andrews et al. 2005] that for any $c = o(\log\log n/\log\log\log n)$, the EDPwC problem is $\Omega(\log^{1/\gamma_c})$-hard to approximate, for some constant γ_c depending only on c. Moreover, it is shown that even when $c = O(\log n \log n/\log\log\log n)$, EDPwC is still hard to approximate up to some super-constant factor.

The throughput maximization problem is defined exactly like machine scheduling, except that here the number of machines is fixed and the goal is to find a maximum cardinality subset of jobs that can be scheduled on the given number of machines. Several constant factor approximation algorithms for this problem are known [Bar-Noy et al. 2001; Erlebach and Spieksma 2003; Bar-Noy et al. 2001; Berman and DasGupta 2000; Chuzhoy et al. 2001]. We note that this problem is

NP-hard in the strong sense for a single machine (even for continuous input) and in fact it is one of the original NP-hard problems in Garey and Johnson [Garey and Johnson 1979]. In the notation convention of [Lawler et al. 1993], the continuous version is denoted by $1|r_j|\sum_j \bar{U}_j$. For discrete input the problem on a single machine is also known as the job interval selection problem. Erlebach and Spieksma [Erlebach and Spieksma 2003] showed that it is Max-SNP hard, thus resolving the approximability of the discrete version to within constant factors. Interestingly, our results show that the discrete and the continuous versions of the machine minimization problem have different approximation factors (we show that the discrete version of machine minimization is $\Omega(\log\log n)$-hard to approximate, even if the optimal solution uses one machine, while the continuous version is known to have an $O(\sqrt{\log n})$-approximation, and the special case where the optimal solution schedules all the jobs on one machine has a constant approximation [Chuzhoy et al. 2004]). No such separation is known for the discrete and continuous versions of the throughput maximization problem.

1.3 Our Techniques

We start by defining a natural special case of the machine scheduling problem which we call the *restricted machine scheduling* problem. In this problem, we assume that for each job, all the time intervals associated with it are disjoint. We perform a simple reduction that shows that restricted machine scheduling can be viewed as a special case of the congestion minimization problem on directed graphs. The main technical part of the paper is devoted to proving that the restricted machine scheduling problem is hard to approximate.

The hardness of restricted machine scheduling is shown via a PCP reduction. We construct a problem instance consisting of $r = \Omega(\log\log n)$ layers, where each layer is a reduction from the gap version of the 3SAT(5) problem to the machine scheduling problem itself. To be more specific, we first construct a basic instance of the machine scheduling problem, which depends on the input 3SAT(5) formula and a number of parameters. Each layer in the final machine scheduling instance is actually a basic instance, which is constructed using different parameters for each layer. Interestingly, the basic instance by itself is not a hard machine minimization instance. In fact, all the jobs in the basic instance can easily be scheduled on one machine. However, the basic instance does have some interesting properties which are exploited to prove the hardness of the overall construction. Thus, it is the careful combination of layers of basic instances, together with their special properties, that makes the final problem instance hard to approximate.

We believe our result opens up several interesting avenues for future research. We study a very restricted special case of the congestion minimization problem, namely the restricted machine scheduling, and we prove that it is hard to approximate. Intuitively, the congestion minimization problem looks much richer and more complex than the scheduling problem we consider. It would be interesting to understand whether the machine scheduling problem indeed captures the hardness of the congestion minimization problem in its full generality, i.e., is machine scheduling as hard to approximate as congestion minimization.

Our techniques involve a recursive layered construction of the final instance. This approach has proved useful in many other hardness of approximation results.

However, a usual practice is to start with a single layer corresponding to an instance which is hard to approximate up to some constant factor, and then amplify this factor by recursively adding new layers. An interesting feature of our construction is that in our case, each layer, when viewed as a machine minimization problem instance, is easily solvable in polynomial time, while the final instance is hard to approximate.

2. PRELIMINARIES

The input to the *congestion minimization* problem is a graph $G = (V, E)$, either directed or undirected, with a collection $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ of source-sink pairs. The goal is to find the smallest integer c such that each input pair can be connected by a simple path, while the number of paths passing through any edge is at most c.

The input for the *discrete machine scheduling* problem is a set J of n jobs, where for each job $j \in J$ an explicitly specified set $\mathcal{I}(j)$ of time intervals is given. These time intervals are also called *job intervals*. A job j is scheduled by choosing one of the time intervals in $\mathcal{I}(j)$. The task is to schedule all the jobs using a minimum number of machines, such that no two jobs assigned to a machine overlap in time. In the *restricted machine scheduling* the time intervals $\mathcal{I}(j)$ associated with each job $j \in J$ are *disjoint*.

We show that the restricted machine scheduling problem is $\Omega(\log \log n)$-hard to approximate unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log \log n)})$. Clearly, the restricted machine scheduling problem is a special case of the general machine scheduling problem. We also show that the restricted discrete machine scheduling problem is a special case of the congestion minimization problem on directed networks. Thus the hardness result holds for both these problems.

3. FROM MACHINE SCHEDULING TO CONGESTION MINIMIZATION

In this section we show a reduction from the restricted machine scheduling problem to the directed congestion minimization problem. The main result of this section is summarized in the following theorem.

THEOREM 3.1. *For any instance of restricted machine scheduling, there is an instance of directed congestion minimization, which can be constructed in polynomial time, such that minimum congestion equals the minimum number of machines needed to schedule all the jobs.*

PROOF. Suppose we are given an instance of machine scheduling, where for each job, all its intervals are mutually disjoint. Let D be the set of all the points on the time line where some job interval starts or finishes. Denote these points by $d_1 \leq d_2 \leq \cdots \leq d_N$. The corresponding congestion minimization problem is defined as follows. For each job $j \in J$, there is a source s_j and a destination t_j. The vertex set V is:

$$V = D \cup \{s_j, t_j \mid j \in J\}$$

There are two sets of edges. The first set is $E_1 = \{(d_i, d_{i+1}) \mid 1 \leq i < N\}$. In order to define the second set of edges, E_2, consider some job $j \in J$ and one of its time intervals $I \in \mathcal{I}(j)$. Let $l(I) : 1 \leq l(I) \leq N$ denote the index of the left endpoint of I, and let $r(I) : 1 \leq r(I) \leq N$ denote the index of the right endpoint of I. Then there is a pair of edges $(s_j, d_{l(I)})$, $(d_{r(I)}, t_j)$ in E_2. More formally,

$$E_2 = \{(s_j, d_{l(I)}), (d_{r(I)}, t_j) \mid j \in J, I \in \mathcal{I}(j)\}.$$

The set of edges in our congestion minimization instance is $E = E_1 \cup E_2$. For each job interval I, let $P(I)$ denote the path $(d_{l(I)} \rightarrow d_{l(I)+1} \rightarrow \cdots \rightarrow d_{r(I)})$.

We can assume, without loss of generality, that in any solution to the congestion minimization problem, each demand j is routed via a path of the form $(s_j \rightarrow P(I) \rightarrow t_j)$, where $I \in \mathcal{I}(j)$. This is the case since after leaving s_j, the path must continue to some $d_{l(I)}$ for some $I \in \mathcal{I}(j)$. Then, it is impossible for the path to leave $P(I)$ before arriving at $d_{r(I)}$, since the only way to do so is via some $t_{j'}$, $j' \neq j$. But, as no edges are leaving $t_{j'}$, the path will not be able to arrive at t_j. Therefore, the path must be of the form $(s_j \rightarrow P(I) \rightarrow \cdots \rightarrow t_j)$. If the path does not go directly to t_j after leaving $P(I)$, we can reroute it, so that the path becomes $(s_j \rightarrow P(I) \rightarrow t_j)$, since this can only decrease the congestion.

Thus, for each demand j, the path $(s_j \rightarrow P(I) \rightarrow t_j)$, via which commodity j can be routed, translates to a job interval $I \in \mathcal{I}(j)$, where job j can be scheduled, and vice versa. It is therefore easy to see that the minimum number of machines needed to schedule all the jobs equals the minimum congestion needed to route all the demands. □

It is clear from Theorem 3.1 that restricted machine scheduling can be viewed as a special case of directed congestion minimization. Therefore, any hardness of approximation result for the restricted machine scheduling problem holds for the directed congestion minimization problem as well.

4. HARDNESS OF RESTRICTED MACHINE SCHEDULING

4.1 Preliminaries

The hardness of restricted machine scheduling involves a reduction from the gap version of Exact MAX 3SAT(5), which is defined as follows. The input is a CNF formula with n variables and $5n/3$ clauses. Each clause contains exactly 3 literals, and each variable appears in exactly 5 different clauses. The goal is to find an assignment maximizing the number of satisfied clauses. The following theorem is one of the several alternative statements of the famous PCP theorem [Arora and Safra 1998; Arora et al. 1998].

THEOREM 4.1. *There is some constant $\epsilon > 0$, such that it is NP-hard to distinguish between a satisfiable 3SAT(5) formula, and a formula where no assignment can satisfy more than a fraction $(1 - \epsilon)$ of the clauses.*

Definition 4.2. A 3SAT(5) formula φ is called a YES-INSTANCE if it is satisfiable. It is called a NO-INSTANCE if no assignment satisfies more than a fraction $(1 - \epsilon)$ of the clauses.

In our reduction, we start from a 3SAT(5) formula φ on n variables and produce an instance of restricted machine scheduling problem with at most $n^{O(\log\log\log n)}$ jobs. If φ is a YES-INSTANCE, then all the jobs can be scheduled on one machine. If φ is a NO-INSTANCE, then at least $c \log\log n$ machines are needed. The hardness result therefore follows from the reduction and Theorem 4.1. Our reduction uses the Raz verifier with $\ell = \Theta(\log \log \log n)$ repetitions, which is described next.

The Raz verifier for MAX 3SAT(5) with ℓ repetitions receives as input a 3SAT(5) formula φ, performs some interaction with two provers, at the end of which it either accepts or rejects. The actions of the verifier are as follows:

—Choose, uniformly and independently, ℓ clauses $C_1, \ldots, C_\ell$ from the formula φ and send the indices of these clauses to Prover 1.

—In each clause C_i, $1 \le i \le \ell$, choose a random variable $x_i \in C_i$, which is called a *distinguished* variable, and send the indices of these variables to Prover 2.

—Receive the answers of the provers to the queries. Prover 1 is expected to send an assignment to the variables that appear in the clauses $C_1, \ldots, C_\ell$. Prover 2 is expected to send an assignment to the variables $x_1, \ldots, x_\ell$.

—Check that for each clause C_i, $1 \le i \le \ell$, the assignment sent by Prover 1 satisfies the clause. If this is not true, reject.

—Check that for each i, $1 \le i \le \ell$, the assignments of Prover 1 and Prover 2 to x_i are identical, and accept or reject accordingly.

The following theorem follows from Theorem 4.1 combined with the Raz Parallel Repetition Theorem [Raz 1998].

THEOREM 4.3. *There is a constant $\alpha > 0$, such that:*

—*If φ is a* YES-INSTANCE, *then there is a strategy of the provers that makes the verifier accept always.*

—*If φ is a* NO-INSTANCE, *then no matter what the strategy of the provers is, the verifier accepts with probability at most $2^{-\alpha\ell}$.*

We view the Raz verifier with ℓ repetitions as the following constraint satisfaction problem. We have a set X of variables, containing one variable x for each possible query to Prover 1 (i.e., for each possible sequence of ℓ clauses). The range of variable x is denoted by $\mathcal{A}_x$, and it consists of all the possible answers of Prover 1 to the query corresponding to x, that satisfy all the clauses in the query. Clearly, $|\mathcal{A}_x| = 7^\ell$, and $|X| = (5n/3)^\ell$. Similarly, we also have a set Y of variables, that contains one variable y for each possible query to Prover 2 (i.e., for each possible sequence of ℓ variables of formula φ). The range of variable y, which is denoted by $\mathcal{A}_y$, is the set of all the possible answers of Prover 2 to the query. Thus, $|\mathcal{A}_y| = 2^\ell$, and $|Y| = n^\ell$.

We denote the set of constraints by Φ. For each random string r of the verifier, there is a constraint $(x, y) \in \Phi$, where $x \in X$ is the query sent to Prover 1 and $y \in Y$ is the query sent to Prover 2, if the verifier chooses random string r. Constraint (x, y) is satisfied, if and only if the assignments to x and y are consistent, i.e., the assignment to y and the projection of the assignment to x onto the distinguished variables are identical. Note that for each possible assignment to x, there is exactly one assignment to y that satisfies the constraint (x, y). Therefore, each constraint (x, y) defines a function $\pi_{x,y} : \mathcal{A}_x \to \mathcal{A}_y$.

The next corollary is an easy consequence of Theorem 4.3.

Corollary 4.4. *If φ is a* Yes-Instance, *then there is an assignment to $X \cup Y$, such that all the constraints in Φ are satisfied. If φ is a* No-Instance, *then no assignment satisfies more than a fraction $2^{-\alpha\ell}$ of the constraints.*

We call the constraints in Φ "type x-y constraints". We define another set Ψ of constraints, called "type x-x constraints". Consider some $x_1, x_2 \in X$. There is a constraint $(x_1, x_2) \in \Psi$ if and only if there is some $y \in Y$, such that $(x_1, y) \in \Phi$ and $(x_2, y) \in \Phi$. Let $a_1 \in \mathcal{A}_{x_1}$ and $a_2 \in \mathcal{A}_{x_2}$ be assignments to x_1 and x_2, respectively. Then, the constraint (x_1, x_2) is satisfied if and only if for every $y \in Y$, such that $(x_1, y) \in \Phi$ and $(x_2, y) \in \Phi$, assignments a_1 and a_2 imply the same assignment to y, i.e., $\pi_{x_1,y}(a_1) = \pi_{x_2,y}(a_2)$. In this case we say that a_1 and a_2 are consistent.

Note that each $x \in X$ participates in exactly 3^ℓ constraints in Φ, and each $y \in Y$ participates in exactly 5^ℓ constraints in Φ. Therefore, for each $x \in X$, there are at most 15^ℓ constraints in Ψ in which x participates.

4.2 The Construction

We show that there is no efficient algorithm, which distinguishes between the instances in which all the jobs can be scheduled on one machine, and the instances in which at least $c \log\log n$ machines are needed.

Our hardness proof is inspired and motivated by the hardness results for hypergraph covering which recently appeared in [Dinur et al. 2002; Dinur et al. 2003]. The starting point of our reductions is the Raz verifier for 3SAT(5). Given a 3SAT(5) formula φ on n variables, we construct a discrete machine scheduling instance where the size of the construction is $N = n^{O(\log\log\log n)}$. If φ is satisfiable, then all the jobs can be scheduled on a single machine. If no assignment can satisfy more than a fraction $(1-\epsilon)$ of the clauses in φ, then we need at least $\Omega(\log\log N)$ machines to schedule all the jobs. The reduction to the machine scheduling problem is performed in two stages. First, given the input formula φ, we construct a "basic instance" of discrete machine scheduling. In the second stage, we combine $\Theta(\log\log n)$ layers of the basic instance in a special way to obtain the final construction.

4.2.1 *The Basic Instance.* In this section we construct an instance of the restricted discrete machine scheduling problem, which is called "the basic instance" and discuss its properties. In our final reduction, we are going to construct a number of such basic instances, and combine them together to obtain the final scheduling problem.

A basic instance is determined by the input 3SAT(5) formula φ, and by the following parameters:

—Integer k.

—For each $x \in X$, there is a collection of $k(x) \le k$ subsets of assignments, $\mathcal{A}^x_1, \mathcal{A}^x_2, \ldots, \mathcal{A}^x_{k(x)} \subseteq \mathcal{A}_x$. For each i, $|\mathcal{A}^x_i| \ge |\mathcal{A}_x| - \log\log n$.

The parameter ℓ (number of repetitions in the Raz verifier) is always the same, $\ell = \frac{3}{\alpha}\log\log\log n$, where α is the constant from Theorem 4.3. We note that in our final construction where we combine several basic instances, the integer k and the sets $\mathcal{A}^x_1, \mathcal{A}^x_2, \ldots, \mathcal{A}^x_{k(x)}$ will be determined separately for each one of the basic instances.

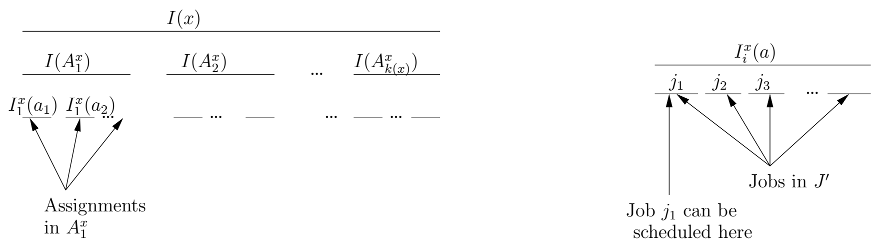

Fig. 1. The "virtual" intervals and the jobs

We now define the basic instance. Intuitively, the time intervals represent variables and their possible assigned values, while jobs correspond to constraints. Denote the set of jobs by J. For each job $j \in J$, denote by $\mathcal{I}(j)$ its set of its intervals, i.e., the time intervals in which the job can be scheduled.

First, we define a collection of "virtual" intervals on the time line, which are not part of the problem input, but which will be useful later when we define the jobs and their intervals.

—For each variable $x \in X$, there is an interval representing x and denoted by $I(x)$. This interval is called a *variable interval*. All variable intervals are equal-sized and non-overlapping.

—The variable interval $I(x)$ is further subdivided into $k(x)$ non-overlapping equal sized intervals representing the input subsets of assignments $\mathcal{A}_i^x$, $1 \leq i \leq k(x)$. An interval corresponding to $\mathcal{A}_i^x$ is denoted by $I(\mathcal{A}_i^x)$

—Finally, each interval $I(\mathcal{A}_i^x)$ is divided into $|\mathcal{A}_i^x|$ equal-sized non-overlapping intervals. Each such interval represents an assignment $a \in \mathcal{A}_i^x$. We denote this interval by $I_i^x(a)$. Note that the assignment a may appear in several subsets of assignments and thus will have several intervals.

We proceed to define the set of jobs J and the job intervals. The job intervals will be given implicitly, as follows. First, we will specify, for each assignment interval $I_i^x(a)$, a subset of jobs "belonging" to it. In order to define the final job intervals, suppose that for some assignment interval $I_i^x(a)$, the subset of jobs belonging to it is $J' \subseteq J$. Then, the "virtual" interval $I_i^x(a)$ is further divided into $|J'|$ equal-sized non-overlapping subintervals, where each subinterval corresponds to exactly one job $j \in J'$, and j can be scheduled in this subinterval. Thus, given a job j, for each assignment interval $I_i^x(a)$ to which j belongs, there is some subinterval of $I_i^x(a)$, such that j can be scheduled on this subinterval, i.e., the subinterval is in $\mathcal{I}(j)$ (See Figure 1).

We are now ready to define the set of jobs J and the job intervals. Consider variables $x, x' \in X$, such that there is a constraint $(x, x') \in \Psi$. Consider some $\mathcal{A}_i^x$ and $\mathcal{A}_{i'}^{x'}$, $1 \leq i \leq k(x), 1 \leq i' \leq k(x')$. There is a job $j = j(\mathcal{A}_i^x, \mathcal{A}_{i'}^{x'})$, if and only if there is **no** pair of assignments $a_1 \in \mathcal{A}_x \setminus \mathcal{A}_i^x$, $a_2 \in \mathcal{A}_x \setminus \mathcal{A}_{i'}^{x'}$, such that a_1 and a_2 are consistent. If job $j = j(\mathcal{A}_i^x, \mathcal{A}_{i'}^{x'})$ exists, then it belongs to all the intervals $I_i^x(a)$ and $I_{i'}^{x'}(a')$, for all $a \in \mathcal{A}_i^x$, $a' \in \mathcal{A}_{i'}^{x'}$ (see Figure 2).

This finishes the description of the basic instance construction. Observe that for

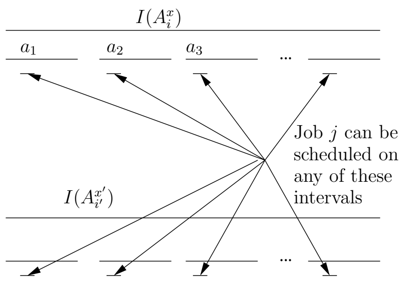

Fig. 2. The intervals in $\mathcal{I}(j)$

each job j, the intervals in $\mathcal{I}(j)$ are non-overlapping, and thus we indeed construct an instance of the restricted machine scheduling. Actually, all the job intervals in the above instance are non-overlapping, and thus we can easily schedule all the jobs on one machine, by choosing one arbitrary interval in $\mathcal{I}(j)$ for each job j. We next establish some structural properties of the solutions corresponding to the Yes and the No instances of the input 3SAT(5) formula. It might be not apparent to the reader at this point why these properties are useful. In particular, these properties will of course not show that the basic instance is hard to approximate. However, these properties will be crucial when proving hardness of the final instance.

Yes-Instance. Consider any feasible solution to the scheduling problem defined above. For any interval $I_i^x(a)$, we say the interval is *used* by the solution, if there is some job which is scheduled inside this interval.

Claim 4.5. *Suppose the initial 3SAT(5) formula φ is a* Yes-Instance. *For each $x \in X$, let $f(x)$ denote the assignment to x obtained from the satisfying assignment to φ. Then, there is a way to schedule all the jobs, such that the only intervals $I_i^x(a)$ that are used are those for which $a = f(x)$ (i.e., only intervals corresponding to the satisfying assignment are used).*

Proof. Consider some job $j = j(\mathcal{A}_i^x, \mathcal{A}_{i'}^{x'})$. Let $a = f(x)$ and $a' = f(x')$. Clearly, a and a' are consistent. So, by the definition of j, it is impossible that $a \notin \mathcal{A}_i^x$ and also $a' \notin \mathcal{A}_{i'}^{x'}$ (in such a case the job j would not have existed). Therefore, in case $a \in \mathcal{A}_i^x$, we can schedule j in its interval in $I_i^x(a)$, and in case $a' \in \mathcal{A}_{i'}^{x'}$, we can schedule j in its interval in $I_{i'}^x(a')$. $\square$

No-Instance. Consider any feasible solution to the scheduling problem defined above. We say that interval $I(\mathcal{A}_i^x)$ is *used* by the solution, if there is some job scheduled inside this interval. We say that the variable $x \in X$ is *good*, if *all* the intervals $I(\mathcal{A}_i^x)$, $1 \le i \le k(x)$, are used by the solution. The next claim states that in any solution of the No-Instance, at least half the variables are good. We note that this is not necessarily the case in the Yes-Instance: it is possible that for most variables x, the satisfying assignment $f(x)$ does not belong to some of the sets $\mathcal{A}_i^x$, and thus x would not be a good variable in the solution described above.

Claim 4.6. *Suppose the initial 3SAT formula φ is a* No-Instance. *Then, in any solution of the above scheduling problem, at least half the variables are good.*

PROOF. Assume for contradiction that there is a solution for which less than half the variables are good. Let X' be the subset of variables that are not good. For each $x \in X'$, there is at least one interval $I(\mathcal{A}_i^x)$ which is not used (if there are several such intervals, fix any of them). Denote the index of the corresponding subset of assignments by $i = i(x)$. Let $B(x) = \mathcal{A}_x \setminus \mathcal{A}_{i(x)}^x$. Recall that $|B(x)| \leq \log\log n$.

Intuitively, the idea of the proof is the following. Since more than half the variables are not good, we have a constant fraction of x-x-type constraints (x, x'), where $x, x' \in X'$. Consider any such constraint (x, x'). Since x, x' belong to X', there are two corresponding intervals $I(\mathcal{A}_{i(x)}^x)$, $I(\mathcal{A}_{i(x')}^{x'})$ that are not used by the solution. Since all the jobs are scheduled, this means that there is no job that corresponds to these two intervals, and therefore the complements of their assignment sets ($B(x)$ and $B(x')$) contain a pair of matching assignments. Furthermore, for each variable $x \in X'$, the size of $B(x)$ is small. Therefore, if we randomly choose, for each $x \in X'$, an assignment in $B(x)$, we will satisfy a large fraction of x-x constraints. Using a similar reasoning, we show that we can also satisfy a large fraction of x-y type constraints, thus contradicting Corollary 4.4.

We now provide a formal proof. Let $\Phi' \subseteq \Phi$ be the subset of x-y constraints in which the variables from X' participate. As $|X'| \geq \frac{1}{2}|X|$, and each variable participates in the same number of x-y constraints, $|\Phi'| \geq \frac{1}{2}|\Phi|$. We show an assignment to $X \cup Y$ that satisfies a large fraction of the constraints in Φ'. For each $x \in X'$, we randomly choose one of the assignments in $B(x)$, uniformly and independently.

Now, consider some $y \in Y$ that participates in some constraint $(x, y) \in \Phi'$ for some $x \in X'$. Denote x by x_y (if there are several such variables x, then fix any of them as x_y.) Let $a \in B(x)$ be the assignment chosen by x_y. Then, the assignment to y is $\pi_{x_y,y}(a)$. For all the other variables in $X \cup Y$, fix the assignments arbitrarily.

We now compute the expected fraction of constraints in Φ' which are satisfied by this assignment. Consider some constraint $(x, y) \in \Phi'$. If $x = x_y$, then clearly, the constraint is satisfied. Otherwise, suppose that $x' = x_y$, where $x' \neq x$, $x' \in X'$. Since the intervals $I(\mathcal{A}_{i(x)}^x)$ and $I(\mathcal{A}_{i(x')}^{x'})$ are not used by the solution, and since all the jobs are scheduled, there is no job $j(\mathcal{A}_{i(x)}^x, \mathcal{A}_{i(x')}^{x'})$. Therefore, there are two assignments, $a \in B(x)$ and $a' \in B(x')$, which are consistent. If these assignments are chosen by x and x', the constraint (x, y) is satisfied. The probability that this happens is at most $\frac{1}{|B(x)||B(x')|} \geq \frac{1}{(\log\log n)^2}$.

Thus, the expected fraction of satisfied constraints is at least $\frac{1}{2(\log\log n)^2}$, since $|\Phi'| \geq \frac{1}{2}|\Phi|$. As $\ell = \frac{3}{\alpha}\log\log\log n$, we have that the fraction of satisfied constraints is $\frac{1}{2(\log\log n)^2} > 2^{-\alpha\ell}$, contradicting Corollary 4.4. □

Construction size. We have $|X| = (5n/3)^\ell$, and each variable $x \in X$ has at most k subsets of assignments $\mathcal{A}_i^x$. Consider some $\mathcal{A}_i^x$. There are at most 15^ℓ variables x' such that there is a constraint $(x, x') \in \Psi$. For each such x', for each i', $1 \leq i' \leq k$, there might be a job $j(\mathcal{A}_i^x, \mathcal{A}_{i'}^{x'})$. Therefore, there are at most $(5n/3)^\ell k^2 15^\ell$ jobs.

Each job has at most 7^ℓ intervals which are contained inside the same variable interval. So the total number of job intervals is bounded by:

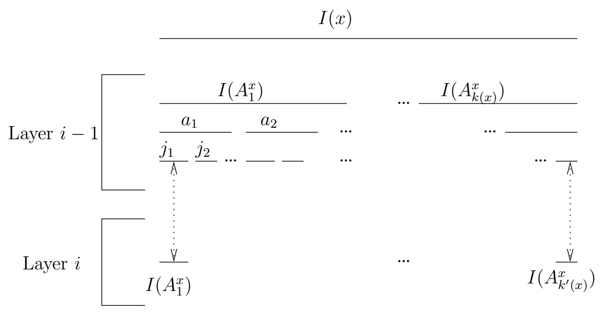

Fig. 3. Layer i construction

$$2 \cdot (5n/3)^\ell 15^\ell 7^\ell k^2 \leq n^{c' \log\log\log n} k^2$$

for some constants c'. Note that each variable interval $I(x)$ is subdivided into at most $k^2 15^\ell 7^\ell = k^2 (\log\log n)^{c''}$ job intervals, for some constant c''.

4.2.2 **The Final Construction**. We use $r = \frac{1}{\log 3} \cdot \log\log n$ copies of the basic construction, setting the parameters of the basic instances appropriately. We refer to the copies of the basic construction as layers. The layers are combined in the following way.

First, for each $x \in X$, we fix some interval $I(x)$ on the time line. All the intervals $I(x)$ are equal-sized and non-overlapping. Interval $I(x)$ serves as the variable interval representing x in each one of the r layers.

We now describe layer 1. The parameter k for layer 1 is $k_1 = 1$, and for each $x \in X$, $k_1(x) = 1$ and $\mathcal{A}_1^x = \mathcal{A}_x$.

Description of layer i: Consider some $x \in X$. Interval $I(x)$ in layer $(i-1)$ is subdivided into at most $k_{i-1}^2 (\log\log n)^{c''}$ job intervals. For each such job interval I, we define a subset of assignments to x, $\mathcal{A}_I(x)$. Interval I becomes the interval corresponding to set $\mathcal{A}_I(x)$ in the construction of layer i (see Figure 3). Thus, $k_i(x)$ is exactly the number of job intervals inside $I(x)$ in layer $(i-1)$, and $k_i = k_{i-1}^2 (\log\log n)^{c''}$.

Given job interval I in layer $(i-1)$, we now only need to define the corresponding subset of assignments $\mathcal{A}_I(x)$. Observe that I, being a job interval, is completely contained in some layer-$(i-1)$ assignment interval, representing some assignment a_{i-1}. This assignment interval is in turn contained in some layer-$(i-2)$ assignment interval representing some assignment a_{i-2}, and so on. Thus, we have a collection of $(i-1)$ assignments $a_1, \ldots, a_{i-1}$. The subset of assignments $\mathcal{A}_I(x)$ is defined as follows: $\mathcal{A}_I(x) = \mathcal{A}_x \setminus \{a_1, \ldots, a_{i-1}\}$. Note that since there are less than $\log\log n$ layers, $|\mathcal{A}_I(x)| \geq |\mathcal{A}_x| - \log\log n$ as required.

YES-INSTANCE

CLAIM 4.7. *If φ is a* YES-INSTANCE*, then it is possible to schedule all the jobs on one machine.*

PROOF. In each layer, we use the solution defined in the previous section, i.e., for each variable x, we only use intervals corresponding to the satisfying assignment $f(x)$.

Observe that the construction is defined in such a way, that if some interval in layer i representing some assignment $a \in \mathcal{A}_x$ overlaps with some interval in layer i' representing some assignment $a' \in \mathcal{A}_x$, and $i \neq i'$, then $a \neq a'$. (This follows from the definition of $\mathcal{A}_I(x)$.) Therefore, in our schedule, all the jobs are scheduled on non-overlapping intervals. $\square$

No-Instance. We show that if φ is a No-Instance, then any solution uses at least $\frac{1}{2\log 3}\log\log n$ machines. We start with the following claim.

Claim 4.8. *Consider a feasible solution to the scheduling problem. Let $x \in X$, and suppose x is a good variable in q layers. Then the schedule uses at least q machines for the jobs that are scheduled inside the time interval $I(x)$.*

Recall that if φ is a No-Instance, then at least half the variables are good in each layer. Therefore, there is at least one variable x which is good in at least half the layers. Combining this with Claim 4.8, we have the following corollary:

Corollary 4.9. *If ϕ is a No-Instance, then any solution to the scheduling problem uses at least $\frac{1}{2\log 3}\cdot\log\log n$ machines.*

Proof of Claim 4.8. We denote the layers in which variable x is good by $i_1 < i_2 < \cdots < i_q$. Recall that if x is good in some layer h, then we use all the intervals $I(\mathcal{A}_i^x)$, $1 \leq i \leq k_h(x)$.

Since x is good in layer i_1, there is at least one layer-i_1 job j scheduled in $I(x)$. Consider the interval I_1 on which it is scheduled. By the construction, there is at least one interval $I(\mathcal{A}_i^x)$ in layer i_2 that is completely contained in the interval I_1. Since variable x is good in layer i_2, the interval $I(\mathcal{A}_i^x)$ is used by the solution, and there is at least one layer i_2 job scheduled in it. Denote the corresponding job interval I_2. Continuing in the same fashion, we obtain a sequence of q intervals $I_q \subseteq \cdots \subseteq I_2 \subseteq I_1$, where for each $a : 1 \leq a \leq q$, I_a is a job interval in layer i_a, and there is a job scheduled in it.

Thus, we have a nested set of q job intervals, and therefore at least q machines are needed for scheduling them. $\square$

The Construction Size. The size of the construction is dominated by the size of the last layer, which is bounded by:

$$n^{c'\log\log\log n}(\log\log n)^{c''}k_r^2.$$

Recall that the recursive formula for k is: $k_1 = 1$; $k_i = k_{i-1}^2(\log\log n)^{c''}$. Clearly, $k_i \leq (\log\log n)^{c''3^i}$.

In total, the size of the construction is at most:

$$\begin{aligned} N &= n^{c'\log\log\log n}(\log\log n)^{c''}(\log\log n)^{c''3^r} \\ &\leq n^{c'\log\log\log n}\cdot 2^{O(\log n\log\log\log n)} \\ &= n^{O(\log\log\log n)} \end{aligned}$$

since $r = \frac{1}{\log 3} \log\log n$. Clearly, $r = \Theta(\log\log N)$. We have thus proved the following theorem.

THEOREM 4.10. *Let N denote the size of the discrete machine scheduling problem. Then there is no $c \log\log N$–approximation algorithm for machine scheduling, for some constant c, unless $NP \subseteq \mathsf{DTIME}(n^{O(\log\log\log n)})$.*

Combining Theorem 4.10 together with Theorem 3.1, we have the following corollary:

COROLLARY 4.11. *Let N denote the size of the directed congestion minimization problem. Then there is no $c \log\log N$–approximation algorithm for directed congestion minimization, for some constant c, unless $NP \subseteq \mathsf{DTIME}(n^{O(\log\log\log n)})$.*

ACKNOWLEDGMENTS

We would like to thank Sudipto Guha and Sanjeev Khanna for many fruitful discussions and helpful suggestions.

REFERENCES

ANDREWS, M., CHUZHOY, J., KHANNA, S., AND ZHANG, L. 2005. Hardness of the undirected edge-disjoint paths problem with congestion. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*. 226–244.

ANDREWS, M. AND ZHANG, L. 2005a. Hardness of the edge-disjoint paths problem with congestion.

ANDREWS, M. AND ZHANG, L. 2005b. Hardness of the undirected congestion minimization. In *Proceedings of the 37th ACM Symposium on Theory of Computing*. ACM, 284–293.

ANDREWS, M. AND ZHANG, L. 2005c. Hardness of the undirected edge-disjoint paths problem. In *Proceedings of the 37th ACM Symposium on Theory of Computing*. ACM, 278–283.

ANDREWS, M. AND ZHANG, L. 2006. Logarithmic hardness of the directed congestion minimization problem. In *Proceedings of the 38th ACM Symposium on Theory of Computing*. ACM, 517–526.

ARORA, S., LUND, C., MOTWANI, R., SUDAN, M., AND SZEGEDY, M. 1998. Proof verification and the hardness of approximation problems. *Journal of the ACM 45,* 3, 501–555.

ARORA, S. AND SAFRA, S. 1998. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM 45,* 1, 70–122.

AZAR, Y. AND REGEV, O. 2001. Strongly polynomial algorithms for the unsplittable flow problem. In *Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization*. 15–29.

BAR-NOY, A., BAR-YEHUDA, R., FREUND, A., NAOR, J., AND SCHIEBER, B. 2001. A unified approach to approximating resource allocation and scheduling. *JACM 48,* 5, 1069–1090.

BAR-NOY, A., GUHA, S., NAOR, J., AND SCHIEBER, B. 2001. Approximating the throughput of multiple machines under real-time scheduling. *SIAM Journal on Computing 31*, 331–352.

BAVEJA, A. AND SRINIVASAN, A. 2000. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research 25*, 255–280.

BERMAN, P. AND DASGUPTA, B. 2000. Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization 4,* 3 (Sept.), 307–323.

CHEKURI, C. AND KHANNA, S. 2003. Edge disjoint paths revisited. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*. 628–637.

CHEKURI, C., KHANNA, S., AND SHEPHERD, F. B. 2004. Edge disjoint paths in planar graphs. In *Proceedings of the 45th IEEE Annual Symposium on Foundations of Computer Science*. 71–80.

CHEKURI, C., KHANNA, S., AND SHEPHERD, F. B. 2005. Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the 37th ACM Symposium on Theory of Computing*. 183–192.

Chekuri, C., Khanna, S., and Shepherd, F. B. 2006a. Edge-disjoint paths in planar graphs with constant congestion. In *Proceedings of the 38th ACM Symposium on Theory of Computing.* 757–766.

Chekuri, C., Khanna, S., and Shepherd, F. B. 2006b. An $o(\sqrt{n})$-approximation for edp in undirected graphs and directed acyclic graphs. *Theory of Computing 2*, 137–146.

Chuzhoy, J., Guha, S., Khanna, S., , and Naor, J. 2004. Machine minimization for scheduling jobs with interval constraints. In *Proc. of the 45th Annual IEEE Symposium on Foundations of Computer Science.* 81–90.

Chuzhoy, J. and Khanna, S. 2005. New hardness results for undirected edge disjoint paths. Manuscript.

Chuzhoy, J. and Khanna, S. 2006. Congestion minimization in directed graphs. Manuscript.

Chuzhoy, J., Ostrovsky, R., and Rabani, Y. 2001. Approximation algorithms for the job interval selection problem and related scheduling problems. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science.* 348–356.

Dinur, I., Guruswami, V., and Khot, S. 2002. Vertex cover on k-uniform hypergraphs is hard to approximate within factor $(k - 3 - \epsilon)$. Tech. Rep. TR02-027.

Dinur, I., Guruswami, V., Khot, S., and Regev, O. 2003. A new multilayered PCP and the hardness of hypergraph vertex cover. In *Proceedings of the 35th ACM Symposium on Theory of Computing.* 595–601.

Erlebach, T. and Spieksma, F. C. R. 2003. Interval selection: Applications, algorithms, and lower bounds. *J. Algorithms 46,* 1, 27 – 53.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., and Yannakakis, M. 2003. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences 67,* 3, 473–496.

Guruswami, V. and Talwar, K. 2005.

Kleinberg, J. M. 1996. Approximation algorithms for disjoint paths problems. Ph.D. thesis, MIT, Cambridge, MA.

Kolliopoulos, S. G. and Stein, C. 1998. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In *Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization.* ACM, 153 – 168.

Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., and Shmoys, D. B. 1993. Sequencing and scheduling: Algorithms and complexity. In *Handbooks in Operations Research and Management Science, Vol.4: Logistics for Production and Inventory.* North Holland, 445–522.

Raghavan, P. and Thompson, C. D. 1987. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica 7*, 365–374.

Raz, R. 1998. A parallel repetition theorem. *SIAM J. Comput. 27,* 3, 763–803.

Srinivasan, A. 1997. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science.* 416–425.

Varadarajan, K. and Venkataraman, G. 2004. Graph decomposition and a greedy algorithm for edge-disjoint paths. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms.* 379–380.

Received Month Year; revised Month Year; accepted Month Year