New Hardness Results for Congestion Minimization and Machine Scheduling

Julia Chuzhoy [*]   Joseph (Seffi) Naor [†]

ABSTRACT

We study the approximability of two natural NP-hard problems. The first problem is *congestion minimization* in directed networks. We are given a directed capacitated graph and a set of source-sink pairs. The goal is to route all pairs with minimum congestion on the network edges. A special well-studied case of this problem is the edge-disjoint paths problem where all edges have unit capacities. The second problem is *discrete machine scheduling* problem where we are given a set of jobs, and for each job, a list of intervals in 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 a Max SNP hardness was known for each problem. We make some progress in closing this gap by showing that both problem are $\Omega(\log\log n)$-hard to approximate unless $NP \subseteq \mathsf{DTIME}(n^{O(\log\log\log n)})$. Our hardness for the congestion minimization holds for the special case of edge-disjoint paths itself.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.1 [**Discrete Mathematics**]: Combinatorics.

General Terms

Algorithms, Theory.

[*]Computer Science Dept., Technion, Haifa 32000, Israel. E-mail: cjulia@cs.technion.ac.il.

[†]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).

Keywords

Approximation Algorithms, Machine Scheduling, Congestion Minimization, Hardness of Approximation.

1. INTRODUCTION

This paper considers hardness of approximation for congestion minimization and machine scheduling problems. In the congestion minimization problem, we are given a graph with edge capacities and a collection of source-sink pairs. The goal is to find the smallest factor α such that each input pair can be connected by a simple path without exceeding the edge capacities by more than a factor of α. The special case where all edge capacities are unit is known as the *edge disjoint paths* (EDP) problem. 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 [19] shows that a randomized rounding of the multicommodity flow relaxation yields that $\alpha = O(\log n/\log\log n)$ on both undirected and directed graphs, where n denotes the number of vertices in the graph.

A closely related problem is the throughput maximization version of EDP, where the objective is to connect a maximum number of source-sink pairs via edge-disjoint paths. This is a fundamental problem, extensively studied, and the best known approximation factors are $O(\min\{n^{2/3}, \sqrt{m}\})$ for undirected graphs and $O(\min\{n^{4/5}, \sqrt{m}\})$ for directed graphs [6]. As for hardness of approximation, in directed graphs, an $\Omega(n^{1/2-\epsilon})$-hardness is known for any $\epsilon > 0$ [13, 6], and for undirected graphs, only a Max SNP-hardness result is known.

In the *discrete* machine scheduling problem, we are given a set of n jobs and for each job an *explicitly* specified set of time intervals. A job is scheduled by choosing one of its associated time intervals. 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. Alternatively, the input to the machine scheduling problem can be given by defining for each job j, a processing time p_j, and an allowed window of time W_j. Job j can be scheduled in any time interval of length p_j which is entirely contained in W_j. We refer to the latter type of input as *continuous*, in contrast to *discrete* input (previously defined), where the allowed set of time intervals is given explicitly.

A related problem is the real time scheduling problem or the throughput maximization problem, where the number of machines is fixed and the goal is to find a maximum revenue subset of jobs that can be scheduled on the given set of machines. Several constant factor approximation algorithms for this problem were given recently [4, 11, 3, 5, 8]. 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 problems in Garey and Johnson [12]. In the notation convention of [18], 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 (JISP). Spieksma [11] showed that it is Max SNP hard to approximate, thus resolving the approximability of the discrete version to within constant factors.

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 $NP \subseteq \mathsf{DTIME}(n^{O(\log \log \log n)})$. Our hardness result for congestion minimization holds even for the special case of edge disjoint paths (i.e., unit capacities). Similarly, the hardness of discrete machine scheduling holds for the special case when the optimal algorithm can schedule all jobs on a single machine.

Very recently, Chuzhoy et al. [7] obtained a a $\sqrt{\log n}$ approximation factor for machine scheduling with continuous input. They also showed a constant approximation for the special case where an optimal solution uses a constant number of machines. The hardness of machine scheduling with discrete input should be contrasted with this result. Interestingly, for the throughput maximization version, a separation between the approximability of continuous and discrete inputs is not known.

Our hardness proof is inspired and motivated by the hardness results for hypergraph covering which recently appeared in [9, 10]. 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. The construction of the basic instance uses ideas from [9, 10]. In the second stage, we combine $\Theta(\log \log n)$ layers of the basic instance in a special way to obtain the final construction. We then show that the hard instances for discrete machine scheduling that we construct have a special structure that allows us to reduce the problem to directed congestion minimization.

Our results give the first non-trivial inapproximability bounds for the well-studied classical problem of congestion minimization. We note that until recently, it has been a common belief that so-called natural NP-hard optimization problems fall within only very few approximability classes. For example, for minimization problems, they either have a constant approximation factor, or a logarithmic factor, or higher approximation factors. This assumption was supported to some extent by previous work on constraint satisfaction problems [15, 16, 17]. It turns out that this is an overly simplistic view of the terrain of approximation algorithms. Our results demonstrate two natural problems whose approximability threshold is sandwiched between $\Theta(\log \log n)$ and $\Theta(\log n / \log \log n)$.

2. PRELIMINARIES

The input to the *congestion minimization* problem, is a graph, either directed or undirected, with edge capacities and a collection of source-sink pairs. The goal is to find the smallest factor α such that each input pair can be connected by a simple path without exceeding the edge capacities by more than a factor of α.

The input for the *discrete machine scheduling* problem is a set of n jobs, where for each job an explicitly specified set of time intervals is given. The time intervals are also called *job intervals*. A job is scheduled by choosing one of the time intervals it is associated with. 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 discrete machine scheduling* the time intervals associated with each job are *disjoint*.

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

The hardness of the restricted discrete machine scheduling involves a reduction from the gap version of Exact MAX 3SAT(5).

Definition 1. *Exact MAX 3SAT(5) problem 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 which maximizes the number of satisfied clauses.*

The following well known theorem was proved by [1].

Theorem 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 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 1.

Our reduction uses a Raz verifier with $l = \Theta(\log \log \log n)$ repetitions, which is described next.

2.1 Raz Verifier

A 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, a random sequence of l clauses $C_1, \ldots, C_l$ from the formula φ and send the indices of these clauses to prover 1.

- In each clause C_i, $1 \le i \le l$, 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 $C_1, \ldots, C_l$. Prover 2 is expected to send an assignment to the variables $x_1, \ldots, x_l$.

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

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

The following well known theorem follows from [2, 1, 20].

THEOREM 2. *There is a universal 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 $\le 2^{-\alpha l}$.*

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, $|\mathcal{A}_x| = 7^l$, and $|X| = (5n/3)^l$. 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, $|\mathcal{A}_y| = 2^l$, and $|Y| = n^l$.

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 2.

COROLLARY 3. *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 $2^{-\alpha l}$ of the constraints.*

We call the constraints in Φ "type x-y constraints". Following [9, 10], 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 $a_1 \in A_{x_1}$ and $a_2 \in A_{x_2}$ 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 $(x_2, y) \in \Phi$, 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.

3. MACHINE SCHEDULING HARDNESS

In this section we show that a $c \log\log n$–approximation algorithm for restricted machine scheduling, for some constant c, does not exist unless $NP \subseteq \mathsf{DTIME}\left(n^{O(\log\log\log n)}\right)$. In particular, we show that it is impossible to distinguish in polynomial time, 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.

3.1 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 reduction, we are going to construct a number of such basic instances, and combine them together in a special way 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 $x \in X$, a collection of $k(x) \le k$ subsets of assignments, $\mathcal{A}_1^x, \mathcal{A}_2^x, \ldots, \mathcal{A}_{k(x)}^x \subseteq \mathcal{A}_x$. For each i, $|\mathcal{A}_i^x| \ge |\mathcal{A}_x| - \log\log n$.

The parameter l (number of repetitions in the Raz verifier) is always the same, $l = \frac{3}{\alpha} \log\log\log n$. We note that in our final construction where we combine several basic instances, the integer k and the sets $\mathcal{A}_1^x, \mathcal{A}_2^x, \ldots, \mathcal{A}_{k(x)}^x$ will be determined separately for each of the basic instances.

We now define the scheduling instance. Denote the set of jobs by J. For each job $j \in J$, denote by $I(j)$ its set of job intervals, i.e., the time intervals in which it 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 $\mathcal{A}_i^x$, $1 \le i \le k(x)$. 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

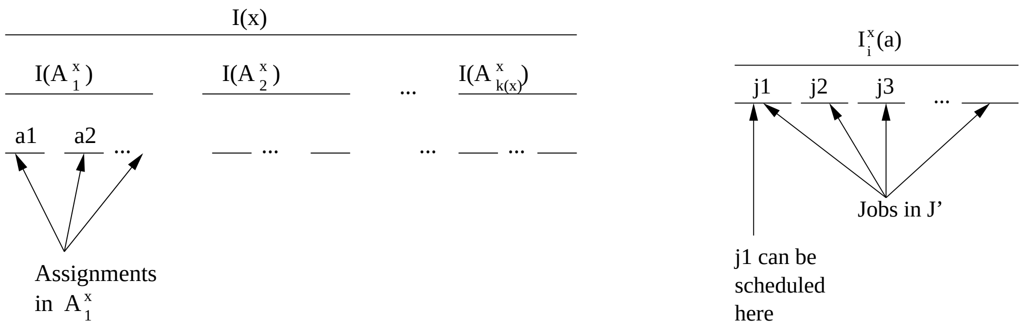

Figure 1: The "virtual" intervals and the jobs

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. Job intervals will be given implicitly, by defining, for each assignment interval $I_i^x(a)$, the jobs belonging to it, as follows. Suppose that for each interval $I_i^x(a)$, there is a subset of jobs $J' \subseteq J$, belonging to it. 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 interval $I_i^x(a)$ to which j is assigned, there is some subinterval of $I_i^x(a)$ in $I(j)$.

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 consistent assignments $a_1 \in \mathcal{A}_x \setminus \mathcal{A}_i^x$, $a_2 \in \mathcal{A}_x \setminus \mathcal{A}_{i'}^{x'}$. 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 $a \in \mathcal{A}_i^x$, $a' \in \mathcal{A}_{i'}^{x'}$.

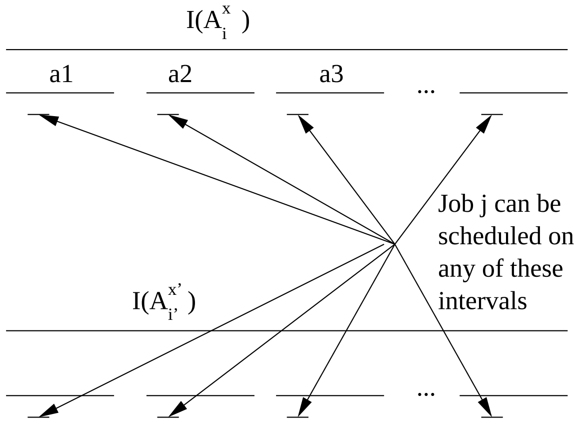

Figure 2: The intervals in $I(j)$

Observe that for each job j, the intervals in $I(j)$ are non-overlapping.

Yes Instances

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. Note that in the solution the jobs are scheduled in non-overlapping intervals, therefore it is always possible to schedule all the jobs on a single machine.

CLAIM 4. *Suppose the initial 3SAT(5) formula φ is a yes-instance, and let $f(x)$, for all $x \in X$, denote the assignment to x obtained from the satisfying assignment to φ. Then there is a schedule of all the jobs, that uses only such intervals $I_i^x(a)$, for which $a = f(x)$ (i.e., only intervals corresponding to the satisfying assignment).*

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 Instances

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 the variable interval $I(x)$ is *good*, if *all* the intervals $I(\mathcal{A}_i^x)$, $1 \leq i \leq k(x)$, are used by the solution.

CLAIM 5. *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.*

Note that in a yes-instance this is not necessarily the case. It is possible that for most of the variables x, the satisfying assignment $f(x)$ does not belong to some of the A_i^x, and thus x would not be good in the solution described above.

PROOF OF CLAIM 5. Let X' be the subset of variables that are not good, and 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 $x \in X'$, 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 $|B(x)| \leq \log\log n$. 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 with $x \in X'$ in a constraint in Φ', and 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 $x' = x_y$. 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 $\geq \frac{1}{(\log\log n)^2}$.

Thus, the expected fraction of satisfied constraints is at least $\frac{1}{2(\log\log n)^2}$, since $|\Phi'| \geq \frac{1}{2}|\Phi|$. Since $l = \frac{3}{\alpha}\log\log\log n$, $\frac{1}{2(\log\log n)^2} > 2^{-\alpha l}$, contradicting Corollary 3. ☐

Size of the Construction

We have $|X| = (5n/3)^l$, 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 $(5n/3)^l k^2 15^l$ 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:

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

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

3.2 The Overall 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 a special 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 $\mathcal{A}_I(x)$ in the construction of layer i. Thus, $k_i(x)$ is exactly the number of job intervals inside $I(x)$ in layer $(i-1)$, and $k_i = k_{i-1}^2(\log\log n)^{c''}$.

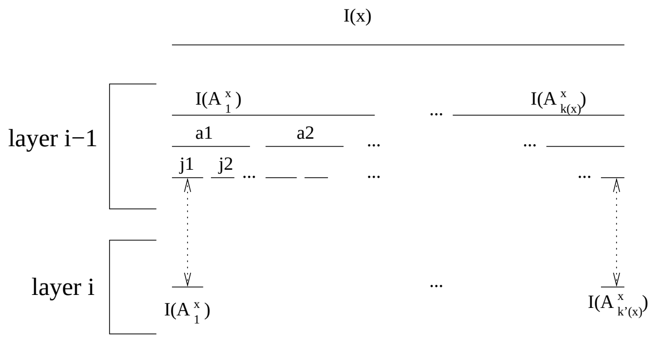

Figure 3: Layer i construction

Given job interval I in layer $(i-1)$, we only need now 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 Instances

Claim 6. *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 j representing some assignment $a' \in \mathcal{A}_x$, and $i \neq j$, then $a \neq a'$. (This follows from the definition of $\mathcal{A}_I(x)$.) Therefore, in our schedule, all the jobs are scheduled in non-overlapping intervals. ☐

No Instances

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 7. *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. Thus, the schedule uses at least $\frac{1}{2\log 3} \cdot \log\log n$ machines.

Proof of Claim 7. We denote the layers in which x is good by $i_1 < i_2 < \cdots < i_q$. Recall that if x is good in some layer j, then we use all the intervals $I(\mathcal{A}_i^x)$, $1 \leq i \leq k_j(x)$.

Since x is good in layer i_1, there is at least one layer 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. □

The Construction Size

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

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

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 8. *There is no $c \log\log n$–approximation algorithm for machine scheduling, for some constant c, unless $NP \subseteq \mathsf{DTIME}(n^{O(\log\log\log n)})$.*

4. HARDNESS OF DIRECTED CONGESTION MINIMIZATION

In this section we show that congestion minimization for the directed edge-disjoint paths is $\Omega(\log\log n)$-hard to approximate, unless $NP \subseteq \mathsf{DTIME}(n^{O(\log\log\log n)})$. The hardness result follows from the result for machine scheduling, and a simple reduction from the restricted machine scheduling.

THEOREM 9. *Suppose we are given an instance of restricted machine scheduling. Then there is an instance of congestion minimization, which can be constructed in polynomial time, where the minimum congestion equals the minimum number of machines needed to schedule all the jobs.*

Observe that in our hardness result for machine scheduling, the intervals belonging to each job are mutually disjoint (in fact, all the job intervals from the same layer are mutually disjoint). Therefore, the next corollary follows from Theorem 8 and Theorem 9.

COROLLARY 10. *There is no $c \log\log n$–approximation algorithm for congestion minimization in directed networks, for some constant c, unless $NP \subseteq \mathsf{DTIME}(n^{O(\log\log\log n)})$.*

PROOF OF THEOREM 9. 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 defined as follows:

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

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 $I \in I(j)$. 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,

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

The set of edges in our congestion minimization problem is $E = E_1 \cup E_2$, and all the edges have unit capacity.

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 w.l.o.g., that in any solution for the congestion minimization problem, each commodity j is routed via a path of the form $(s_j \rightarrow P(I) \rightarrow t_j)$. This is because after leaving s_j, the path must continue to some $d_{l(I)}$ for some $I \in I(j)$. After that, it is impossible that the path leaves $P(I)$ 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 $t_{j'}$, it is impossible for the path 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 the path becomes $(s_j \rightarrow P(I) \rightarrow t_j)$. This can only decrease the congestion.

Thus, for each commodity j, each path $(s_j \rightarrow P(I) \rightarrow t_j)$ via which commodity j can be routed translates to a job interval $I \in I(j)$ 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 commodities. □

Acknowledgements

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

5. REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[2] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[3] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[4] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines under real-time scheduling. *SIAM Journal on Computing*, 31:331-352, (2001).

[5] P. Berman and B. DasGupta. Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization*, 4(3):307–323, 2000.

[6] C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 628–637, 2003.

[7] J. Chuzhoy, S. Guha, S. Khanna, and J. Naor. Approximation algorithms for machine scheduling. Manuscript, 2003.

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

[9] I. Dinur, V. Guruswami, and S. Khot. Vertex cover on k-uniform hypergraphs is hard to approximate within factor $(k-3-\epsilon)$. ECCC Report TR02-027, Electronic Colloquium on Computational Complexity, 2002.

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

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

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

[13] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 19–28, 1999.

[14] E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 585–594, 2003.

[15] S. Khanna, M. Sudan, L. Trevisan and D.P. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing (SICOMP)*, 30(6), pages 1863-1920, 2000.

[16] S. Khanna, M. Sudan, and L. Trevisan. Constraint satisfaction: the approximability of minimization problems. in *Proc. 12th Annual IEEE Computational Complexity Conference (CCC)*, pages 282-296, 1997.

[17] S. Khanna, M. Sudan and D.P. Williamson. A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11-20, 1997.

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

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

[20] R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.