Algorithmic Aspects of Bandwidth Trading

RANDEEP BHATIA

Bell Labs

JULIA CHUZHOY

MIT

ARI FREUND

IBM Haifa Research Lab

AND

JOSEPH (SEFFI) NAOR

Technion

Abstract. We study algorithmic problems that are motivated by bandwidth trading in next-generation networks. Typically, bandwidth trading involves sellers (e.g., network operators) interested in selling bandwidth pipes that offer to buyers a guaranteed level of service for a specified time interval. The buyers (e.g., bandwidth brokers) are looking to procure bandwidth pipes to satisfy the reservation requests of end-users (e.g., Internet subscribers). Depending on what is available in the bandwidth exchange, the goal of a buyer is to either spend the least amount of money so as to satisfy all the reservations made by its customers, or to maximize its revenue from whatever reservations can be satisfied.

We model this as a real-time nonpreemptive scheduling problem in which machine types correspond to bandwidth pipes and jobs correspond to end-user reservation requests. Each job specifies a time interval during which it must be processed, and a set of machine types on which it can be executed.

The work of the J. Naor was supported in part by a Foundational and Strategic Research Grant from the Israeli Ministry of Science and Arts and by the United States-Israel Binational Science Foundation Grant 2002–276.

A preliminary version of this article appeared in the *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP)*, Eindhoven, The Netherland, 2003, 751–766.

Authors' addresses: R. Bhatia, Bell Labs, 600 Mountain Ave., Murray Hill, NJ 07974, e-mail: randeep@research.bell-labs.com; J. Chuzhoy, School of Mathematics, Institute for Advanced Study, 1 Einstein Drive, Princeton, NJ, e-mail: cjulia@math.ias.edu; A. Freund, IBM Haifa Research Lab, Haifa 31905, Israel, e-mail: arief@il.ibm.com; J. (S) Naor, Computer Science Department, Technion, Haifa 32000, Israel, e-mail: naor@cs.technion.ac.il.

If necessary, multiple machines of a given type may be allocated, but each must be paid for. Finally, each job has associated with it a revenue, which is realized if the job is scheduled on some machine.

There are two versions of the problem that we consider. In the cost minimization version, the goal is to minimize the total cost incurred for scheduling all jobs, and in the revenue maximization version the goal is to maximize the revenue of the jobs that are scheduled for processing on a given set of machines. We consider several variants of the problems that arise in practical scenarios, and provide constant factor approximations.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures, sequencing and scheduling*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Approximation algorithms for NP-hard problems, dynamic storage allocation, general caching, scheduling, resource allocation

ACM Reference Format:

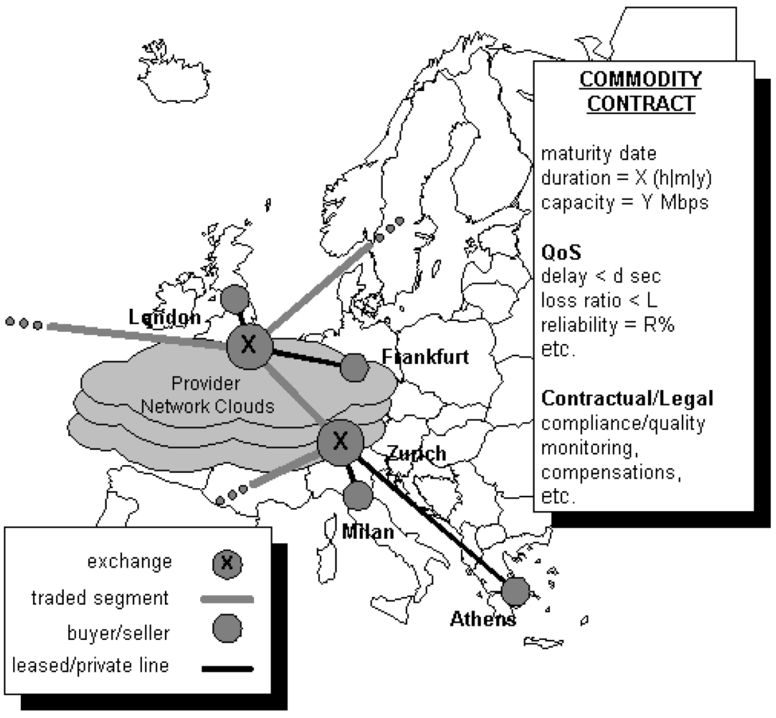
Bhatia, R., Chuznoy, J., Freund, A., and Naor, J. (S). 2007. Algorithmic aspects of bandwidth trading. ACM Trans. Algor. 3, 1, Article 10 (Feb. 2007), 19 pages. DOI = 10.1145/1186810.1186820 http://doi.acm.org/10.1145/1186810.1186820.

1. *Introduction*

We study algorithmic problems involving *bandwidth trading* in next-generation networks. As network operators build new high-speed networks, they look for new ways to sell or lease their plentiful bandwidth. At the same time, there are emerging potential buyers of bandwidth, such as virtual network carriers, who would like to be able to expand capacity easily and rapidly to meet the ever-changing demands of their customers. Similarly, many companies are looking for ways to reserve bandwidth for on-off events such as videoconferences. Finally, there are network subscribers who would like to be able to buy bandwidth for the duration of a webcast, pay per movie on the web, etc. In this article we consider some of the algorithmic problems arising in this context.

1.1. BANDWIDTH TRADING IN PRACTICE. Our work is motivated by the emerging business model being discussed in the networking community, which we briefly describe here. Although bandwidth exchange/trading is not new, the traditional methodology is often marred by long periods of binding contracts and slow provisioning time. With the recent advances in network technologies, however, there has been a tremendous leap forward both in network capacity and provisioning times. It is now possible to quickly provision end-to-end protocol-independent light paths with a specified service-level agreement (SLA) that takes into account QoS, bandwidth, restoration level, etc., in order to rapidly meet the changing bandwidth demands of network users. In addition, many identical low-bandwidth data streams can be multiplexed over a single light path in the core network, thus enabling a core network operator to sell bandwidth at smaller granularities.

Driven to meet demands for high-speed data-centric applications, various upstart network carriers have been rolling out networks with vast amounts of excess capacity. With all this capacity up for grabs, a new generation of resellers, wholesalers, and online bandwidth brokers are poised to resell it to customers. Leading the pack in the bandwidth commodity effort are a host of real-time online trading centers pioneered by the likes of Band-X and RateXchange.



Typically, bandwidth trading involves a *bandwidth exchange* which includes a marketplace for suppliers and buyers of bandwidth, and a set of *pooling points* which are used for actually providing the bandwidth upon settlement. Physically, a pooling point may be a fiber interconnection and switching site in a particular geographical location, with co-located points of presence for buyers (e.g., ISPs) and suppliers (e.g., long-haul carriers). In the pooling point, the buyer's network interfaces with the supplier's high-speed optical network, and data passing between the two is converted from electrical packets to optical signal and vice versa. The assumption is that a bandwidth exchange trades well-defined bandwidth contracts, for example, Cheliotis [2000] and Kenyon and Cheliotis [2001]. Each contract refers to a *bandwidth segment* between two pooling points, where a bandwidth segment is an abstraction of one or more high-capacity networks providing connectivity between the two pooling points. Each bandwidth contract describes the duration for which connectivity will be made available, as well as a service-level agreement (SLA) that takes into account QoS, bandwidth, restoration level, etc. The illustrated inset, taken from the website of IBM Zürich Research Lab (`http://www.zurich.ibm.com/bandwidth/concepts.html`), summarizes the situation by showing an example of a bandwidth segment being offered between two pooling points connected to buyers' networks.

Optical technologies play a central role in next-generation networks. A single strand of optical fiber is now capable of carrying a large number of high-bandwidth data streams, each of which can be individually managed. Typically, a low-bandwidth data stream is dedicated to a single end-user at any given time, but may be shared over time by multiple end-users, where the switch from one user to another is provisioned almost instantaneously. A core optical network operator is therefore able to trade a large number of identical bandwidth contracts (with identical attributes), each corresponding to a low-bandwidth data stream, all of which are multiplexed over a single high-bandwidth light path. For practical purposes, we can therefore assume that an unlimited number of "copies" of each contract are available.

Finally, buyers are themselves service providers to their clients—the end-users. End, users generate bandwidth requests, which are called *forward reservations*. Each forward reservation specifies two endpoints (which translate into two pooling points) between which the bandwidth reservation is required, the time interval for which it is required, any other attributes of the connection (QoS, restoration level, etc.), and the revenue obtained by honoring the reservation. Buyers in the bandwidth exchange, who may be ISPs, bandwidth brokers, etc., are looking to procure bandwidth contracts to satisfy at least cost the forward reservation requests made by their clients (e.g., network subscribers, companies, or virtual network operators). A single procured bandwidth contract can be used to serve a set of "nonoverlapping" forward reservation requests. Depending on what is available in the bandwidth exchange, the goal of a buyer is to either spend the least amount of money to buy enough bandwidth contracts so as to satisfy all the reservations, or, if the reservations cannot be all satisfied, to maximize its revenue from whatever reservations are possible.

Combining contracts. A *contract graph* [Cheliotis 2000; Kenyon and Cheliotis 2001] is defined, given a bandwidth exchange, to be a graph whose nodes are the pooling points and edges represent the traded contracts. Several point-to-point segments (on a path in the graph) can be assembled to connect any two geographical locations. This leads to a new (path) contract whose attributes depend on the choice of path in the contract graph. We stress that the new contract is *indivisible*. For example, consider three pooling points A, B, and C. Suppose that an (A, B) contract and a (B, C) contract are combined into an (A, C) contract. Then, the (A, C) contract cannot be used to also route traffic from A to B or from B to C, since this would require an optical-to-electrical followed by electrical-to-optical conversion at point B. Such conversions introduce substantial delays at intermediate points and deteriorate end-to-end QoS, thus defeating the purpose of high-speed optical routing.

In general, we can assume that if two pooling points have a point-to-point contract between them, then this is the cheapest way to connect the two points (for these attributes), since we consider a highly liquid bandwidth market in which arbitrage opportunities [Chiu and Crametz 2000] are instantaneously removed (a geographic arbitrage arises, for example, if the price of an indivisible point-to-point contract between New York and London is more than the price of a path contract with the same attributes that goes via Los Angeles).

1.2. Wavelength Assignment in Optical Line Systems.

Another motivation for the problems we consider comes from wavelength assignment in optical line systems [Winkler and Zhang 2003] such as those involving DWDMs (*dense wavelength division multiplexing*). An optical line system is a collection of nodes called mesh optical-add-and drop multiplexers (MOADM) arranged in a line, with adjacent nodes connected by optical fibers. A demand enters the line system at one node, exits at some other, and is routed on the same wavelength on the fibers connecting all intermediate nodes. The set of wavelengths available on each fiber connecting two adjacent MOADM (say, node i and $i + 1$) may differ from fiber to fiber, and is a function of the fiber characteristics, and also of the wavelengths which have been used up by previously provisioned demands. Given a set of demands, the problem is to assign them wavelengths so that no two demands use the same wavelength on the same fiber.

We note that any optical line system as previously described can be viewed as a set of windows $\mathcal{I}$, where each $I \in \mathcal{I}$ is an interval corresponding to a single wavelength which is available between the two endpoints of I. Given a set of demands D (i.e., D is a set of intervals), the wavelength assignment problem corresponds to packing the intervals belonging to D into $\mathcal{I}$ such that intervals packed into a window $I \in \mathcal{I}$ do not overlap.

1.3. MODEL DESCRIPTION, NOTATION, AND TERMINOLOGY. We model bandwidth trading as a real-time scheduling problem. As explained earlier, a bandwidth contract can only be used for routing traffic between its two endpoints. Therefore, we only need to consider the bandwidth trading problem for a single pair of pooling points. We view the bandwidth contracts as a set of machines of different types, where identical bandwidth contracts (with the same attributes) correspond to machines of the same type. Each machine type has a *cost* per machine. We can assume that an unlimited number of machines are available from each type, since the available bandwidth on light paths in the core network is greater by many orders of magnitude than the end-user bandwidth requirement for any single data stream. The jobs correspond to reservation requests made by end users. Each job needs to be processed during a specified time interval, and can only be processed on a machine of one of several specified types. Each job has associated with it a value corresponding to the revenue obtained for processing the job. At most, one job can be scheduled on any machine at any given time (since no two overlapping reservation requests can be served by the same bandwidth contract).

In the *cost minimization* version of the problem, the goal is to find a set of machines of minimum cost so as to be able to schedule all the jobs on the machines. In this version of the problem, we ignore the job revenues. In the *revenue maximization* version, the goal is to maximize the total revenue by selecting a subset of the jobs that can be scheduled using a given set of machines. In this version of the problem we ignore the machine costs.

Formally, we have a set of m machine types $\mathcal{T} = \{T_1, \ldots, T_m\}$. A cost, or *weight*, $w(T_i) \geq 0$, is incurred for allocating a machine of type T_i. There are n jobs belonging to an input set of jobs J, where each job j is associated with the following parameters: a revenue $w(j) \geq 0$, a set $S(j) \subseteq \mathcal{T}$ of machine types on which this job can be processed, and a time interval $I(j)$ during which the job is to be processed. For simplicity, the job intervals are half-open, that is, closed on the left and open on the right. We sometimes refer to a job and its interval interchangeably. At most, one job can be processed on a given machine at any given moment.

1.3.1. *The Problems.* The general version of the cost minimization problem, where the sets $S(j)$ of machine types are arbitrary, is essentially equivalent (approximation-wise) to *set cover* (hardness can be shown by a simple reduction in which set elements become nonoverlapping jobs and each set becomes a machine capable of processing only those jobs corresponding to the set's elements. Logarithmic approximability was shown by Jansen [1994]). In practice, however, the definition of the sets $S(j)$ is usually based on properties of the machines and thus has a computationally more convenient structure. We consider two variants arising naturally in bandwidth trading and other real-world applications.

Cost minimization with machine time intervals. Machine types are defined by time intervals. Each type T_i is associated with a time interval $I(T_i)$ during which machines of this type are available. A job can be processed by every machine that is available throughout its processing interval. Thus the sets $S(j)$ are defined by $S(j) = \{T_i \in \mathcal{T} \mid I(j) \subseteq I(T_i)\}$. In the *unweighted* case, all types have unit cost ($w(T_i) = 1$ for all i), and in the *weighted* case, costs are arbitrary.

Cost minimization with machine strengths. There is a linear order of *strength* defined on the machine types, $T_1 \prec T_2 \prec \cdots \prec T_m$, such that a job that may be processed by a machine of a given type may also be processed on every stronger machine, that is, for all jobs j, $S(j)$ has the form $\{T_{i_j}, T_{i_j+1}, \ldots, T_m\}$. We also assume that the stronger a machine is, the higher its cost (otherwise there is no point in using weaker machines). The linear order models a situation in which the SLAs are contained in each other in terms of the capabilities they specify.

We comment that no bounded approximation factors are possible for these problems (unless P = NP) if only a limited number of machines is available from each type, since it is NP-hard even to decide whether all the jobs can be scheduled on all available machines (ignoring types). We show this by demonstrating a reduction from the *circular arc coloring* problem,[1] which is known to be NP-hard [Garey et al. 1980]. Given a circular arc graph and an integer k, pick a point on the circle, and consider the set A of arcs containing the point. These arcs constitute a clique in the graph (and so must use $|A|$ different colors). Let B be the set of remaining arcs and let $\bar{A}$ be the set of complements of the arcs in A. The arcs $\bar{A} \cup B$ induce an interval graph in the obvious manner, and we identify the arcs with these intervals. Construct an instance of the scheduling problem as follows. For each interval $I \in B$, define a job whose interval is I. For each interval $I \in \bar{A}$, define a machine which is available in I. Finally, add another $k - |A|$ machines that are always available (for a total of k machines). Clearly, scheduling all jobs is feasible if and only if coloring the original circular arc graph by k colors is possible (the machines correspond to colors. Each arc in A is colored by the machine corresponding to its complement, and each arc in B is colored by the machine to which its corresponding job is assigned).

We also consider the revenue maximization version.

Revenue maximization. We are given a collection $\mathcal{M}$ of machines (presumably already paid for) and wish to select a maximum revenue subset of the jobs that can be scheduled on these machines. Every job j specifies an arbitrary set $S(j) \subseteq \mathcal{M}$ of machines on which it can be scheduled.

1.4. Our Contribution.

The problems we consider are NP-hard. We present the first polynomial-time constant-factor approximation algorithms for both versions of the problem (cost minimization and revenue maximization).

In Section 2 we consider the cost minimization problem with machine time intervals. We describe a three-approximation algorithm for the weighted case and a

[1] The circular arc coloring problem is the following: Given a circular arc graph (in circular arc representation) and an integer k, determine whether the vertices can be colored by k colors (such that no two adjacent vertices share the same color).

two-approximation algorithm for the unweighted case. We also show how to extend our algorithm (for the weighted case) to a *prize collecting* version of the problem, where it is not necessary to schedule all the jobs, but there is a fine to be paid for each unscheduled job.

Our algorithm for the weighted case is based on a linear programming relaxation and is a variant of the (combinatorial) primal-dual schema [Vazirani 2001]. It is rather unique in that it copes with the difficulty posed by a constraint matrix containing both positive and negative entries. It is an interesting fact that the primal-dual schema is often incapable of dealing with both positive and negative coefficients. Our algorithm is also unconventional in that it departs from the common dual-ascent, reverse-delete paradigm. Rather than generating a minimal solution via a simple reverse-delete stage, we iteratively improve our schedule by a selective rescheduling of jobs which obviates some of the machines in our schedule. While our algorithm is not the only primal-dual algorithm with these traits, there haven't been many such proposed to date (the only other such algorithm we are aware of is Jain and Vazirani's [2001] algortihm for *facility location*).

In Section 3, we show a two-approximation algorithm for the cost minimization problem with machine strengths. Both this algorithm and for the unweighted version of the cost minimization problem with machine time intervals employ simple combinatorial lower bounds on the problems.

We conclude with the revenue maximization problem in Section 4. We present a $(1 - 1/e)$-approximation algorithm for this problem. This result improves on the approximation factor of $1/2$ implicit in Bar-Noy et al. [2001] for this problem.

1.5. RELATED WORK.

Kolen and Kroon [1991, 1994] and Jansen [1994] considered the same scheduling model as ours in the context of aircraft maintenance. When aircraft arrive at the airport, they must be inspected by ground engineers between their arrivals and departures. There are different types of aircraft, and different types of engineer licenses, and there is an aircraft-type/engineer-license-type matrix specifying which license type permits an engineer to work on which aircraft type. In addition, the engineers work in shifts. The cost of assigning an engineer to an aircraft inspection job depends on the engineer's license type and the shift during which the inspection must be carried out. The goal is to enlist a minimum-cost multiset of "engineer instances" to handle all aircraft inspections, where an engineer instance is defined by a (*license*,*shift*) pair. In our model, jobs correspond to inspections of aircraft, and machine types correspond to (*license*,*shift*) pairs.

Kolen and Kroon [1991] study the computational complexity of this problem with respect to different aircraft/engineers matrices when all the shifts are the same. In particular, their work implies that the cost minimization problem we consider in Section 3 is NP-hard. In Kolen and Kroon [1994], author study another version of this problem, where all the aircraft and license types are the same, and there are different time shifts. They show that the problem is NP-hard even for unit costs, implying that the problems we consider in Sections 2.1 and 2.2 are NP-hard, as well. Jansen [1994] gives an $O(\log n)$-approximation algorithm for the general problem, with both aircraft/license types and time shifts. When all shifts are the same and all aircraft types identical, the problem reduces to optimal coloring of interval graphs, and has a polynomial-time algorithm [Arkin and Silverberg 1987].

Maximizing the throughput (*revenue*, in our terminology) in real-time scheduling was studied extensively in Bar-Noy et al. [2001], Bar-Yehuda et al. [2001], Spieksma

[1999], Berman and DasGupta [2000], and Chuzhoy et al. [2001]. They focused on the case where for each job, more than one time interval in which it can be performed is specified, while machines are available continuously. As here, jobs are scheduled nonpreemptively and at each point of time, only one job can be scheduled on a given machine. This model captures many applications, for example, scheduling a space mission, bandwidth allocation, and communication in a linear network. The results of Bar-Noy et al. [2001] on maximizing the throughput of unrelated parallel machines imply an approximation factor of $1/2$ for our revenue maximization problem. This result was improved upon Chuzhoy et al. [2001] to $(1 - 1/e - \epsilon)$ (for any constant ϵ) for the unweighted version of the problem. The revenue maximization problem with machine time intervals was studied by Kolen and Kroon [1993] (see also Kolen and Lenstra [Kolen and Kroon 1995, pp. 1901–1903]). They solved the problem optimally with a dynamic programming algorithm whose running time is $O(n^m)$. This implies that the problem is polynomial-time solvable for a constant number of machines.

The wavelength assignment problem in optical line systems is studied in Winkler and Zhang [2003]. Their result implies that the resulting interval packing problem (which is a decision version of our revenue maximization problem), as described in Section 1.2, is NP-complete.

2. *Cost Minimization with Machine Time Intervals*

In this section we develop approximation algorithms for the special case of the cost minimization problem where each machine type T_i has a time interval $I(T_i)$ during which the machines of this type are available. The sets $S(j)$ of machine types allowable for job j are defined as follows: $S(j) = \{T_i \in \mathcal{T} \mid I(j) \subseteq I(T_i)\}$. We present a three-approximation algorithm for the weighted case and its prize collecting version, and a two-approximation algorithm for the unweighted case.

2.1. The Weighted Case.

Our algorithm for the weighted case is based on the primal-dual schema for approximation algorithms. The linear programming (LP) formulation of the problem contains two sets of variables: $\{x_i\}$ and $\{y_{ij}\}$. For each machine type T_i, variable x_i represents the number of machines allocated of type T_i, and for every pair of machine type T_i and job j such that $I(j) \subseteq I(T_i)$, variable y_{ij} indicates whether job j is assigned to a machine of type T_i. We also use the following notation: E is the set of left endpoints of the job intervals, and for $t \in E$, $J(t)$ is the set of jobs containing t (but recall that job intervals are closed on the left and open on the right). The linear program is:

$$\text{Min} \quad \sum_{i=1}^{m} w(T_i)x_i \qquad \text{such that}$$

$$\sum_{i=1}^{m} y_{ij} \geq 1, \qquad \forall j \in J; \tag{1}$$

$$x_i - \sum_{j \in J(t)} y_{ij} \geq 0, \qquad \forall 1 \leq i \leq m,\ \forall t \in E \cap I(T_i); \tag{2}$$

$$x, y \geq 0. \tag{3}$$

(The sums in constraints (1) and (2) should be understood to include only variables y_{ij} that are defined.) The dual variables are $\{\alpha_j\}$ and $\{\beta_i^t\}$, corresponding to constraints (1) and (2), respectively. The dual program is:

$$\text{Max} \qquad \sum_{j\in J} \alpha_j \qquad \text{s.t.}$$

$$\sum_{t\in E\cap I(T_i)} \beta_i^t \le w(T_i), \qquad \forall 1 \le i \le m; \tag{4}$$

$$\alpha_j - \sum_{t\in E\cap I(j)} \beta_i^t \le 0, \qquad \forall 1 \le i \le m,\ \forall j \text{ s.t. } I(j) \subseteq I(T_i); \tag{5}$$

$$\alpha, \beta \ge 0. \tag{6}$$

Our algorithm proceeds in two phases. In the first phase, it constructs a feasible schedule by iteratively allocating machines and scheduling jobs on them. In the second phase, it improves the solution by considering the allocated machines in reverse order and (possibly) eliminating some, rescheduling jobs as necessary.

Phase 1: *Dual Ascent*. As previously mentioned, the first phase allocates machines and schedules jobs. Accordingly, at a given moment during this phase, there are *scheduled* and *unscheduled* jobs, in addition to *allocated* and *unallocated* machines and machine types. Initially, all jobs are unscheduled, all machines and machine types are unallocated, and all dual variables are set to 0.

Phase 1 iterates as long as there are unscheduled jobs. The kth iteration proceeds as follows. Let $t_k \in E$ be such that a maximum number of unscheduled jobs contain t_k, and let n_k denote the number of these jobs. Let $\mathcal{T}_k$ be the set of all unallocated machine types whose intervals contain t_k. We increase $\beta_i^{t_k}$ for all i such that $T_i \in \mathcal{T}_k$ uniformly at the same rate until some constraint of type (4) becomes tight, that is, we increase each of the βs in question by $\delta_k = \min\{w(T_i) - \sum_{t\in E\cap I(T_i)} \beta_i^t \mid T_i \in \mathcal{T}_k\}$. All the machine types that become tight are considered allocated from now on. For each currently unscheduled job j whose interval is contained in the interval of one of these newly allocated machine types, we allocate a separate machine of the appropriate type (say T_i), schedule j on it, and set $\alpha_j = \sum_{t\in E\cap I(j)} \beta_i^t$.

We claim that the dual solution thus constructed is feasible. Clearly, the algorithm satisfies constraints (4) and (6) at all times. To see that the solution satisfies all constraints of type (5) as well, consider any such constraint $\alpha_j - \sum_{t\in E\cap I(j)} \beta_i^t$. Suppose job j was scheduled in the kth iteration. Following the kth iteration, α_j remains unchanged and the sum of βs can only increase, so it suffices to show that the constraint is satisfied at the end of the kth iteration. Let $T_{i'}$ be the type of machine on which job j was scheduled. If $i = i'$, the constraint is satisfied by equality. Otherwise, machine type T_i could not have been allocated prior to the kth iteration (for otherwise, job j would have already been scheduled by the time the kth iteration commenced), and thus for all $t \in E \cap I(j)$, the values of β_i^t and $\beta_{i'}^t$ must have increased identically during the first k iterations. Hence, $\sum_{t\in E\cap I(j)} \beta_i^t = \sum_{t\in E\cap I(j)} \beta_{i'}^t$ at the end of the kth iteration, and the claim follows.

Phase 2: *Reverse Reschedule and Delete*. Let $\mathcal{M}$ be the set of machines allocated in the first phase. Later on, we describe a reverse-reschedule and delete procedure that returns a feasible schedule using a subset $\mathcal{M}' \subseteq \mathcal{M}$ of machines which has the property that for all k, the number of machines in $\mathcal{M}'$ of types from $\mathcal{T}_k$ is, at most, $3n_k$. We note that in standard primal-dual algorithms, the second phase is a simple *reverse-delete* phase, whose purpose is to yield a minimal solution. The approximation guarantee then follows from an upper bound on minimal solutions. In our case, we do not know how to find a minimal solution. In fact, even determining whether all jobs can be scheduled on a given set of machines is NP-hard. We therefore do not attempt to find a minimal solution, but instead, gradually discard some of the machines in a very special manner designed to achieve the aforementioned property.

Analysis. Let (α, β) be the dual solution constructed in Phase 1 and let (x, y) be the primal solution corresponding to the schedule generated in Phase 2. To show that this schedule is 3-approximate, it suffices to prove that $\sum_{i=1}^{m} w(T_i)x_i \leq 3\sum_{j\in J}\alpha_j$. This inequality follows from the next two claims.

CLAIM 1. $$\sum_{i\in T} w(T_i)x_i \leq 3\sum_k n_k\delta_k.$$

PROOF. For each allocated machine type T_i, $w(T_i)$ equals the sum of δ_k taken over all k such that machine type T_i was unallocated at the beginning of the kth iteration and $t_k \in I(T_i)$. Thus, $\sum_{i\in T} w(T_i)x_i = \sum_k \delta_k m_k$, where m_k is the number of machines of types in $\mathcal{T}_k$ used by the final schedule. The claim then follows, since $m_k \leq 3n_k$. □

CLAIM 2. $$\sum_{j\in J}\alpha_j = \sum_k n_k\delta_k.$$

PROOF. For each job j, α_j is the sum of all δ_k such that job j was still unscheduled at the beginning of the kth iteration and $t_k \in I(j)$. For each iteration k, the number of jobs that were unscheduled at the beginning of the kth iteration and contain t_k is exactly n_k. □

The reverse-reschedule-and-delete procedure. Let $\mathcal{M}$ be the set of machines used in the schedule constructed in the first phase, and let $\mathcal{M}_k \subseteq \mathcal{M}$ be the subset of machines of types in $\mathcal{T}_k$. The purpose of the reverse-reschedule-and-delete procedure is to prune each machine set $\mathcal{M}_k$, leaving only $3n_k$ (or less) of its members allocated, yet manage to feasibly schedule all of the jobs on the surviving machines. To achieve this, we consider the sets $\mathcal{M}_k$ in reverse order (decreasing k), and prune each in turn.

The pruning procedure for $\mathcal{M}_k$ is the following. If $|\mathcal{M}_k| \leq 3n_k$, we do nothing. Otherwise, consider the jobs currently assigned to machines in $\mathcal{M}_k$. They are of three possible types: *left jobs*, which lie entirely to the left of t_k; *right jobs*, which lie entirely to the right of t_k; and *middle jobs*, which cross t_k.

The middle jobs are easiest. The number of middle jobs is exactly n_k (by definition), so they are currently scheduled on n_k different machines. We retain these machines, denoting them $\mathcal{M}_{\text{mid}}$, as well as the scheduling of *all* jobs currently assigned to them (these may include some right jobs or left jobs in addition to all middle jobs).

The remaining left jobs are scheduled in the following manner (see Figure 1). First, note that $|\mathcal{M}_k \setminus \mathcal{M}_{\text{mid}}| \geq 2n_k$, since $|\mathcal{M}_k| > 3n_k$. Denote by $\mathcal{M}_{\text{left}}$ the set of n_k machines in $\mathcal{M}_k \setminus \mathcal{M}_{\text{mid}}$ with leftmost left endpoints. Observe that the intervals of these machines all contain t_k by definition (as do those of all machines in $\mathcal{M}_k$). Let t be the rightmost endpoint among the left endpoints of machines in $\mathcal{M}_{\text{left}}$. All left

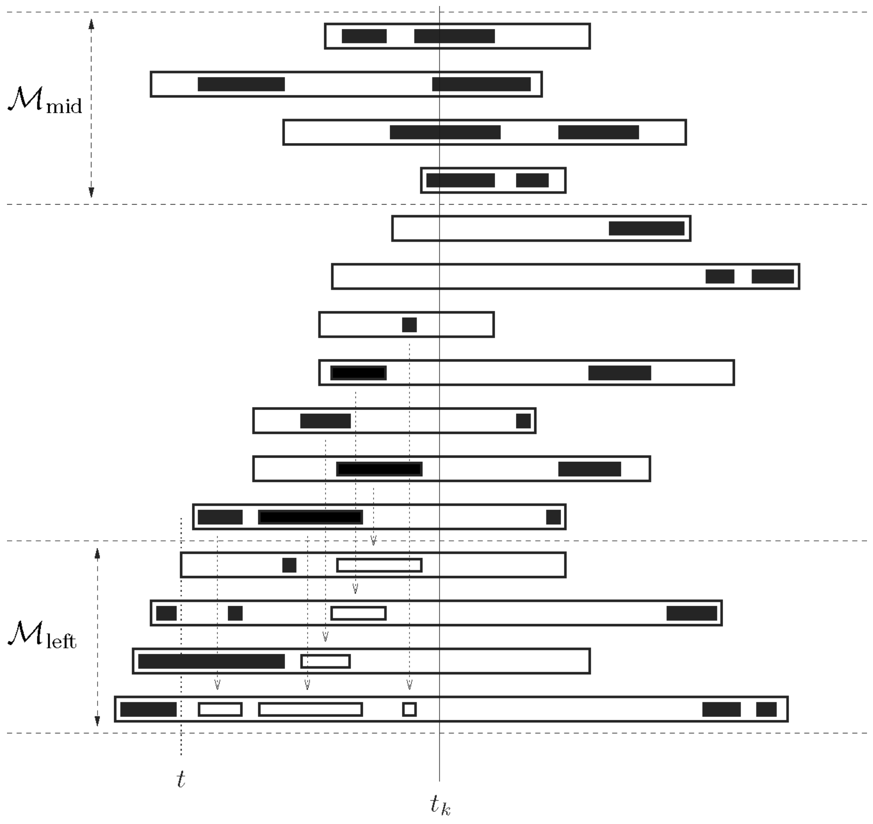


FIG. 1. Rescheduling left jobs. Note that the jobs originally scheduled on $\mathcal{M}_{\text{mid}}$ stay in place.

jobs whose left endpoints are strictly to the left of t must be currently scheduled on machines in $\mathcal{M}_{\text{left}}$, so we leave them intact. We proceed to reschedule all remaining left jobs greedily in order of increasing left endpoint. Specifically, for each job j, we select any machine in $\mathcal{M}_{\text{left}}$ on which we have not already rescheduled a job that conflicts with j, and schedule j on this. To see that this is always possible, observe that all n_k machines are available between t and t_k, and thus if a job cannot be scheduled, then its left endpoint must be contained in n_k other jobs which were scheduled on machines in $\mathcal{M}_k$. These $n_k + 1$ jobs were therefore unscheduled at the beginning of the kth iteration (since we are pruning the sets $\mathcal{M}_k$ in reverse order), but this contradicts the definition of n_k, as these jobs all intersect at one time point.

The remaining right jobs are scheduled in a symmetric manner on the n_k machines in $\mathcal{M} \setminus \mathcal{M}_{\text{mid}}$ with rightmost right endpoints. Some (or all) of these machines may belong to $\mathcal{M}_{\text{left}}$, and therefore may already have left jobs scheduled on them, but this is not a problem because the intervals of left and right jobs do not intersect.

2.1.1. *The Prize Collecting Version of the Problem.* In the prize collecting version of the problem, we are not obligated to schedule all jobs, but must pay a penalty $p(j)$ for each job j that we reject. To cope with this added complication, we need to modify the linear program. For each job $j \in J$, we introduce a new variable z_j that indicates whether j is rejected. The modified linear program is:

$$\text{Min} \quad \sum_{i=1}^{m} w(T_i)x_i + \sum_{j \in J} p(j)z_j \qquad \text{such that}$$

$$\sum_{i=1}^{m} y_{ij} + z_j = 1, \qquad \forall j \in J; \tag{7}$$

$$x_i - \sum_{j \in J(t)} y_{ij} \geq 0, \qquad \forall 1 \leq i \leq m,\ \forall t \in E \cap I(T_i); \tag{8}$$

$$x, y, z \geq 0; \tag{9}$$

and the dual program is:

$$\text{Max} \quad \sum_{j \in J} \alpha_j \qquad \text{such that}$$

$$\sum_{t \in E \cap I(T_i)} \beta_i^t \leq w(T_i), \qquad \forall 1 \leq i \leq m; \tag{10}$$

$$\alpha_j - \sum_{t \in E \cap I(j)} \beta_i^t \leq 0, \qquad \forall 1 \leq i \leq m,\ \forall j \text{ such that } I(j) \subseteq I(T_i); \tag{11}$$

$$\alpha_j \leq p(j), \qquad \forall j \in J; \tag{12}$$

$$\beta \geq 0. \tag{13}$$

The algorithm needs to be modified too. We begin with Phase 1 (dual ascent). As before, machines and machine types may be in allocated or unallocated states, and jobs may be in scheduled or unscheduled states, but may now also be in a third state, namely, *rejected* (of course, the states are mutually exclusive). Initially, all machines and machine types are unallocated, and all jobs unscheduled.

The kth iteration proceeds as follows. As before, let $t_k \in E$ be such that a maximum number of unscheduled jobs contain t_k, and let n_k denote the number of these jobs. Let $\mathcal{T}_k$ be the set of all unallocated machine types whose intervals contain t_k. We increase uniformly at the same rate the dual variables $\beta_i^{t_k}$ for all i such that $T_i \in \mathcal{T}_k$, and α_j for all unscheduled jobs j such that $t_k \in I(j)$ (to clarify: We increase the αs and βs together at the same rate). We increase these variables until some constraint of type (10) or (12) becomes tight, that is, we increase each by $\delta_k = \min\{\min\{w(T_i) - \sum_{t \in E \cap I(T_i)} \beta_i^t \mid T_i \in \mathcal{T}_k\}, \min\{p(j) - \alpha_j \mid j \in I(j) \text{ and is unscheduled}\}\}$. We then reject every unscheduled job whose corresponding constraint (12) becomes tight, and allocate every unallocated machine type whose corresponding constraint (10) becomes tight. For each remaining unscheduled job j whose interval is contained in the interval of one of the newly allocated machine types, we allocate a separate machine of the appropriate type and schedule j on it. In anticipation of Phase 2 and the analysis, we let r_k denote the number of jobs rejected in the kth iteration.

It is not difficult to see that the dual solution remains feasible at all times. Constraints (10), (12), and (13) are clearly always satisfied. Every constraint of type (11), corresponding to machine type T_i and job j, is also initially satisfied and subsequently, whenever α_j increases, the sum of βs appearing in the constraint increases by the same amount (since exactly one of the βs in the sum—namely, $\beta_i^{t_k}$—increases). Since the βs never decrease, the constraint remains satisfied throughout the phase.

Phase 2 proceeds exactly as before, except that the number of middle jobs assigned to machines in $\mathcal{M}_k$ is now $n_k - r_k$ (rather than n_k), and so we do the pruning if $|\mathcal{M}_k| > 3n_k - r_k$, and end up with, at most, $3n_k - r_k$ machines from $\mathcal{M}_k$ in the final schedule.

Let (α, β) be the dual solution obtained at the end of Phase 1, and let (x, y, z) be the primal solution corresponding to the final schedule. To show that the schedule is 3-approximate, it suffices to prove that $\sum_{i=1}^{m} w(T_i)x_i + \sum_{j\in J} p(j)z_j \le 3\sum_{j\in J} \alpha_j$. This inequality follows from the next three claims.

CLAIM 3. $$\sum_{j\in J} p(j)z_j = \sum_k \delta_k r_k.$$

PROOF. First observe that for all jobs j, α_j can increase from its initial value of 0 only once, since it increases during Phase 1 only in iterations k such that $t_k \in I(j)$, and then the job becomes either rejected or scheduled. Now consider the kth iteration of Phase 1. Suppose job j was rejected in this iteration. Then, α_j must have increased during this iteration from 0 to $p(j)$, implying that $p(j) = \delta_k$. This is true for all jobs rejected in the kth iteration (i.e., they must all have penalty equal to δ_k). The claim follows immediately. □

CLAIM 4. $$\sum_{i\in T} w(T_i)x_i \le \sum_k \delta_k(3n_k - r_k).$$

CLAIM 5. $$\sum_{j\in J} \alpha_j = \sum_k \delta_k n_k.$$

The proofs of Claims 4 and 5 follow the same lines as the proofs of Claims 1 and 2 in the previous section and are very similar.

2.2. THE UNWEIGHTED CASE.

We present a two-approximation algorithm for this case. As before, let E denote the set of left endpoints of the job intervals. For each point of time $t \in E$, let n_t be the number of jobs whose intervals contain t. The algorithm consists of two stages. In the first stage, it solves the optimization problem of allocating a minimum number of machines such that for all $t \in E$, at least n_t of the allocated machines are available at time t. In the second stage it schedules the jobs using (at most) twice the number of machines allocated in the first stage.

Stage 1. Scan the points of time in E in left-to-right order. For each point $t \in E$, let n'_t be the number of machines that are available at time t and have already been allocated. If $n'_t < n_t$, allocate another $n_t - n'_t$ machines of type T_t, where T_t is the machine type with the rightmost right endpoint among all machines available at time t.

PROPOSITION 6. *The solution found in Stage* 1 *is optimal.*

PROOF. We say that a time point $t \in E$ is *covered* by a set of machines $\mathcal{M}$ if $\mathcal{M}$ contains at least n_t machines that are available at time t. Let t_i be the time point considered in the ith iteration of Stage 1, and let $\mathcal{M}_i$ be the set of machines allocated in the first i iterations of Stage 1. We prove by induction that for all i, there exists an optimal solution which contains $\mathcal{M}_i$. For $i = 0$, $\mathcal{M}_i = \emptyset$ and the claim holds trivially. Consider $i > 0$. By the induction hypothesis, there exists an optimal solution $\mathcal{M}^*$ such that $\mathcal{M}_{i-1} \subseteq \mathcal{M}^*$. If no new machines are allocated in the ith iteration, then $\mathcal{M}_i = \mathcal{M}_{i-1} \subseteq \mathcal{M}^*$. Otherwise, there are at least $n_{t_i} - n'_{t_i}$ machines in $\mathcal{M}^* \setminus \mathcal{M}_{i-1}$ that are available at time t_i. We remove any $n_{t_i} - n'_{t_i}$ of these from $\mathcal{M}^*$ and replace them by the newly allocated machines $\mathcal{M}_i \setminus \mathcal{M}_{i-1}$. This cannot affect the feasibility because all time points $t_j < t_i$ remain covered by $\mathcal{M}_{i-1}$, and the choice of $\mathcal{M}_i \setminus \mathcal{M}_{i-1}$ as machines with rightmost right endpoints that are available at time t_i guarantees that they are all available at all times $t_j \geq t_i$ at which any of the machines they replace are available. □

Remark 7. A different approach to the solution of the optimization problem of Stage 1 is through the natural integer linear program for this problem. It is easy to see that the constraint matrix defining the linear program is totally unimodular (TUM) and thus, the optimal solution to the linear program is always integral.

Stage 2. Let $\mathcal{M}$ be the set of machines allocated in Stage 1. Order the jobs by their left endpoint (from left-to-right) and schedule them in this order on machines in $\mathcal{M}$. Select for each job any machine on which no previously scheduled jobs intersect with the present job. The machine selected must also satisfy the condition that its time interval contains the job's left endpoint (though not necessarily the job's entire interval). The resultant schedule might, of course, be infeasible due to jobs extending beyond the right endpoints of the machines on which they are scheduled, but at most, one job per machine may do so. Fix the schedule by allocating new machines, one for each of these jobs. At most, $|\mathcal{M}|$ new machines are added to the schedule.

THEOREM 8. *Stage* 2 *returns a 2-approximate solution.*

PROOF. The initial (infeasible) schedule constructed in Stage 2 contains all jobs, for if Stage 2 cannot schedule some job, then there are at least k other jobs containing its left endpoint $t_i \in E$, where k is the number of machines in $\mathcal{M}$ available at time t_i. This implies $n_{t_i} > k$, contradicting the fact that at each point of time $t \in E$, at least n_t machines are allocated in Stage 1. Thus, the final schedule constructed in Stage 2 is feasible and uses at most $2|\mathcal{M}|$ machines. Since $|\mathcal{M}|$ is clearly a lower bound on the optimum, the solution is 2-approximate. □

3. *Cost Minimization With Machine Strengths*

In this section we present a two-approximation algorithm for the special case of a cost minimization problem where there is a linear order of strength on the machine types $T_1 \prec T_2 \prec \cdots \prec T_m$, such that a job which may be processed by a machine of a given type may also be processed on every stronger machine. In other words, $S(j)$ has the form $\{T_{i_j}, T_{i_j+1}, \ldots, T_m\}$ for all $j \in J$. We also assume that the stronger a machine, the higher its cost, that is, $w(T_i) < w(T_{i+1})$ (otherwise there is no point in ever using weaker machines). We say that job j *exists* at time t if $t \in I(j)$.

For $1 \leq i \leq m$, let n_i be the maximum cardinality of a set of jobs that all exist simultaneously at some time point and require machines of type T_i or stronger. Clearly, every feasible schedule requires at least n_i machines of type T_i or stronger, for all i. Thus, the cost of an optimal schedule is at least as high as the minimum cost of a set of machines with the property that for all i, the set contains at least n_i machines of type T_i or stronger. Define $n_{m+1} = 0$. Consider a set of machines $\mathcal{M}$ consisting of $n_i - n_{i+1}$ machines of type T_i, for all $1 \leq i \leq m$ (note that $n_i \geq n_{i+1}$

for all i, and the number of machines allocated in $\mathcal{M}$ of type T_i or stronger is n_i). Then, $\mathcal{M}$ has the aforementioned property, and because stronger machines cost more than weaker ones, $\mathcal{M}$ is a minimum cost set with this property. Thus, the cost of $\mathcal{M}$ is a lower bound on the cost of optimal solution. We show how to schedule all jobs on a set of machines containing, at most, two copies of each machine in $\mathcal{M}$. This schedule is therefore 2-approximate.

Let $M_1, \ldots, M_k$ (where $k = n_1$) be the machines in $\mathcal{M}$ ordered from weakest to strongest. Construct an *initial* infeasible schedule as follows. Consider the machines in order from M_1 to M_k. For each M_i, construct a schedule containing a subset of the jobs as follows. First, schedule on M_i all of the currently unscheduled jobs that can be processed on it. Ignore job overlap when constructing this schedule. Then, iterate: As long as there is a job j scheduled on M_i that is fully contained in the union of other jobs scheduled on M_i, unschedule job j. Although the schedule thus constructed for M_i may contain overlapping jobs, it has the redeeming property that the interval graph it induces is two-colorable, as it is an easy fact that if three intervals intersect, at least one must be contained in the union of the other two. Having constructed the initial schedule (on all machines), color the induced interval graph on each machine with two colors and create a feasible schedule by using two copies of $\mathcal{M}$, one for each color class.

It remains to show that the initial schedule contains all jobs. Restricting our attention to this schedule, we say that a time point t is covered on machine M_i if there is a job containing t scheduled on M_i. By construction, the set of points covered on M_i is precisely the union of all jobs that could be processed on M_i and were still unscheduled when the algorithm reached M_i. It follows that if a time point t is not covered on M_i, then every job that contains it either cannot be processed on M_i or is scheduled on some machine $M_{i'}$, $i' < i$. Thus, suppose the algorithm fails to schedule some job j. Let t be any time point in j. Then t must be covered on the strongest machine, namely, M_k, since it is contained in an unscheduled job (namely, j) that can be processed on it. Let i be minimal such that t is covered on machines $M_i, M_{i+1}, \ldots, M_k$. Let J' be the set of jobs scheduled on these machines that contain t. Assuming $i > 1$, point t is not covered on M_{i-1}, by definition. Thus, by our previous observation, all of the jobs in $J' \cup \{j\}$ cannot be processed on M_{i-1}, and one (or two) of them are scheduled on M_i, so M_i is strictly stronger than M_{i-1}. Let T_{l-1} be the type of machine M_{i-1}. Then, all the jobs from $J' \cup \{j\}$ require machines of type at least as strong as T_l. Thus, $|J' \cup \{j\}| \leq n_l$. On the other hand, the number of available machines of types T_l or stronger is less than $|J' \cup \{j\}|$, which is at most n_l, contradicting the fact that $\mathcal{M}$ contains n_l such machines. In the case $i = 1$, we get a contradiction directly: $n_1 \geq |J' \cup \{j\}| \geq k + 1 > n_1$.

THEOREM 9. *Our algorithm returns a 2-approximate solution.*

4. *Revenue Maximization*

In the revenue maximization problem, we are given a set of jobs J and a set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$ (presumably already paid for). Since the set of machines is fixed here, we identify machines with machine types. For each job $j \in J$, there is a time interval $I(j)$ during which it should be processed and a non-negative *profit* (or *weight*) $w(j)$ with which it is associated. Every job j specifies an arbitrary set $S(j) \subseteq \mathcal{M}$ of machines on which it can be scheduled. The goal is to find a feasible schedule of a subset of the jobs on the machines that maximizes the total profit of the jobs scheduled. We present a $(1 - 1/e)$-approximation algorithm for this problem. Our approach is to cast the problem as an integer problem and solve its linear programming relaxation. We then obtain an integral solution by randomly rounding the optimal fractional solution found for the LP relaxation.

The linear program: For each job j and each machine M_i, there is a variable x_{ij} that indicates whether job j is scheduled on machine M_i (if job j cannot be scheduled on machine M_i, we add a constraint $x_{ij} = 0$. These constraints are not shown in the following). Also, recall that E is the set of left endpoints of the job intervals, and for $t \in E$, $J(t)$ is the set of jobs containing t.

$$\text{Max} \qquad \sum_{j \in J} \sum_{M_i \in S(j)} w(j) x_{ij} \qquad \text{such that}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \qquad \forall j \in J; \tag{14}$$

$$\sum_{j \in J(t)} x_{ij} \leq 1, \qquad \forall i \in \{1, \dots, m\},\ \forall t \in E; \tag{15}$$

$$x \geq 0. \tag{16}$$

Constraints (14) guarantee that each job is scheduled (at most) once. Constraints (15) guarantee that each machine executes at most one job at each time point.

Randomized rounding: Let x be an optimal fractional solution. Choose N to be the smallest integer such that $N \cdot x_{ij}$ is integral for all i, j. The rounding is done on each machine separately. For each machine M_i, perform the following steps:

(1) Construct an interval graph $\mathcal{I}$ as follows. For each job j, add $N \cdot x_{ij}$ copies of the time interval $I(j)$ to $\mathcal{I}$. Note that at each time point, the sum of fractions of the jobs executed on machine M_i is, at most, 1. Thus the size of the maximum clique in the interval graph $\mathcal{I}$ is, at most, N.
(2) Color $\mathcal{I}$ with N colors. Each color class induces a feasible schedule on machine M_i.
(3) Choose one of the color classes uniformly at random. Schedule on M_i all the jobs that have time intervals belonging to this color class.

If a job is scheduled on more than one machine, arbitrarily unassign it from all but one machine.

We now estimate the expected revenue of the schedule thus generated. For each job j, let $x_j = \Sigma_{i=1}^{m} x_{ij}$. For a job j, the probability of its being scheduled on a particular machine M_i is exactly x_{ij}. Therefore, the probability that it is not assigned to M_i is $1 - x_{ij}$. Thus, the probability that it is not assigned to any machine is

$$\prod_{i=1}^{m} (1 - x_{ij}) \leq \prod_{i=1}^{m} \left(1 - \frac{x_j}{m}\right) = \left(1 - \frac{x_j}{m}\right)^m < e^{-x_j}.$$

The probability that job j appears in the final schedule is therefore not less than $1 - e^{-x_j} \geq (1 - 1/e) x_j$, where the inequality follows from the fact that the real function $1 - e^{-x} - (1 - 1/e)x$ is non-negative in the range $0 \leq x \leq 1$ (as can easily be seen by differentiation). Thus, the expected revenue is at least $(1 - 1/e) \sum_j w(j) x_j$. The next theorem follows from the preceding discussion.

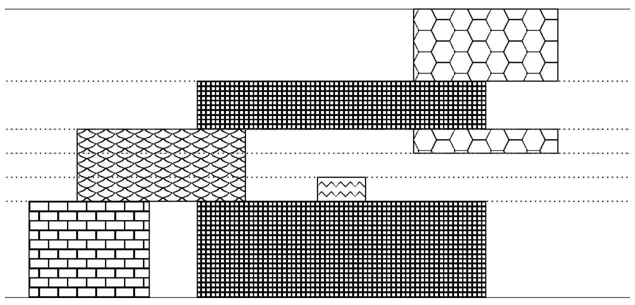


FIG. 2. A packing of five jobs.

THEOREM 10. *The algorithm yields a* $(1 - 1/e)$*-approximate solution.*

We remark that there is no need to build the interval graph $\mathcal{I}$ explicitly, that is, to replicate intervals. We can base the rounding for machine M_i on the following assignment to color classes (which is computable in strongly polynomial time). To understand the color class construction, it is convenient to consider the following *strip packing* problem. Each job j is identified with an axis parallel rectangle of size $|I(j)| \times x_{ij}$ (where $I(j)$ is the horizontal dimension). The problem is to pack (without overlap) all of the rectangles into an infinite horizontal unit-width strip (representing time), subject to the following rules. The horizontal location of rectangle j is fixed—it is the interval $I(j)$. The vertical location, however, may be chosen freely. Furthermore, it is permitted to cut the rectangle into a number of horizontal slices and to position the slices (vertically) independently. Figure 2 shows an example of a possible packing of five jobs.

We construct a packing as follows. We sort the jobs by increasing left endpoint and pack them in this order. Whenever we pack a rectangle slice, we conceptually pass infinite lines through its two horizontal sides, thus partitioning the strip into horizontal substrips, indicated by the dotted lines in Figure 2. Suppose we have already packed several jobs, and are now processing job j. We cut rectangle j into slices, packing each into one of the substrips (choosing, of course, substrips that are not obstructed by previously packed rectangles). The rectangle is sliced up in such a manner that the width (i.e., the vertical dimension) of each slice equals the width of the substrip into which it is packed, except, possibly for the width of the last slice, which might be smaller. If it is smaller, we pack this slice flush against the bottom (or top) of its substrip. We then conceptually extend the sides of the new slices to the left and right. Doing so can increase the number of substrips by, at most 1 (the number will increase only if the last slice's width is smaller than that of its substrip, in which case the substrip will be partitioned into two). Thus, packing the mth jobs entails cutting it into at most m slices, and so the whole process can be carried out in $O(n^2)$ time. The result is a packing of the rectangles and a partitioning of the strip into (at most) $n + 1$ substrips such that no two rectangle slices overlap horizontally anywhere inside any of the substrips. We can therefore think of each substrip as a color class.

Given the packing and partitioning of the strip into substrips, we select one of the substrips at random, and assign the jobs represented in it to machine M_i. The random choice is not made uniformly, though. Rather, the probability of selecting a substrip equals its width. This ensures that the probability that job j is scheduled on machine M_i is exactly x_{ij}, and thus the analysis remains unchanged.

We also remark that the algorithm can be derandomized, without decreasing the approximation factor, using the method of conditional expectations (see, e.g., Alon and Spencer [2000]). Since this method is standard by now, we only sketch how to apply it to our algorithm. The idea is to perform a search in the sample space associated with the random variables so as to find a good schedule. In our case, each random variable corresponds to the choice of a color class for one machine. The random variables are considered one-by-one. At the kth step of the derandomization, the values of the first $k-1$ variables have already been set (i.e., the color classes have already been chosen for $k-1$ machines), and one variable is currently being considered. We compute the conditional expected revenue for each outcome (i.e., choice of color class) of the random variable, where the expectation is on that outcomes of all remaining random variables, conditioned on the choices already made for the first $k-1$ variables and the outcome under consideration of the kth variable. We set the kth variable to that outcome maximizing this conditional expectation. The conditional expectations can be computed, given the color classes for each machine, in a manner easily derived from our previous analysis of the algorithm.

Remark 1. A similar algorithm can be used to obtain a $(1-1/e)$-approximation for the more general problem where each job j has a release date r_j, a deadline d_j, and a processing time p_j, and $d_j - r_j < 2p_j$ for all j.

ACKNOWLEDGMENT. We thank Frits Spieksma for pointing out reference Kolen and Kroon [1991].

REFERENCES

ARKIN, E. M., AND SILVERBERG, E. B. 1987. Scheduling jobs with fixed start and end times. *Discrete Appl. Math. 18*, 1–8.

ALON, N., AND SPENCER, J. H. 2000. *The Probabilistic Method*, 2nd ed. John Wiley, New York.

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

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

BERMAN, P., AND DASGUPTA, B. 2000. Multi-Phase algorithms for throughput maximization for real-time scheduling. *J. Comb. Optim. 4*, 307–323.

CHELIOTIS, G. 2000. Bandwidth trading in the real world: Findings and implications for commodities brokerage. In *3rd Berlin Internet Economics Workshop*. 26–27.

CHIU, S., AND CRAMETZ, J. P. 2000. Surprising pricing relationships. In *Bandwidth Special Report*, 12–14. http://www.riskwaters.com/bandwidth/images/pricing.pdf.

CHUZHOY, J., OSTROVSKY, R., AND RABANI, Y. 2001. Approximation algorithms for the job interval selection problem and related scheduling problems. In *Proceedings of the 42nd Annual Symposium on the Foundations of Computer Science*. 348–356.

GAREY, M. R., JOHNSON, D. S., MILLER, G. L., AND PAPADIMITRIOU, C. H. 1980. The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Discrete Methods 1*, 216–227.

JAIN, K., AND VAZIRANI, V. V. 2001. Approximation algorithms for the metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM 48*, 274–296.

JANSEN, K. 1994. An approximation algorithm for the license and shift class design problem. *Eur. J. Oper. Res. 73*, 127–131.

KENYON, C., AND CHELIOTIS, V. V. 2001. Stochastic models for telecom commoditiy prices. *Comput. Netw. 36*, 533–555.

KOLEN, A. W. J., AND KROON, L. G. 1991. On the computational complexity of (maximum) class scheduling. *Eur. J. Oper. Res. 54*, 23–38.

KOLEN, A. W. J., AND KROON, L. G. 1993. On the computational complexity of (maximum) shift scheduling. *Eur. J. Oper. Res. 64*, 138–151.

KOLEN, A. W. J., AND KROON, L. G. 1994. An analysis of shift class design problems. *Eur. J. Oper. Res. 79*, 417–430.

LENSTRA, J. K., AND KOLEN, A. W. J. 1995. Combinatorics in operations research. In *Handbook of Combinatorics*, R. L. Graham et al. Eds. North-Holland, Amster.

SPIEKSMA, F. C. R. 1999. On the approximability of an interval scheduling problem. *J. Scheduling 2*, 215–227.

VAZIRANI, V. V. 2001. *Approximation Algorithms*. Springer-Verlag.

WINKLER, P., AND ZHANG, L. 2003. Wavelength assignment and generalized interval graph coloring. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 830–831.

RECEIVED SEPTEMBER 2004; REVISED JANUARY 2006; ACCEPTED JANUARY 2006