Minimizing Total Busy Time in Parallel Scheduling with Application to Optical Networks ⋆

Michele Flammini [a] Gianpiero Monaco [a] Luca Moscardelli [b]
Hadas Shachnai [c] Mordechai Shalom [d] Tami Tamir [e]
Shmuel Zaks [c]

[a] *Department of Computer Science, University of L'Aquila, Italy.*

[b] *Department of Science, University of Chieti-Pescara, Pescara, Italy.*

[c] *Department of Computer Science Department, Technion, Haifa, Israel.*

[d] *Tel-Hai Academic College, 12210 Upper Gallilee, Israel.*

[e] *School of Computer Science, The Interdisciplinary Center, Herzliya, Israel.*

Abstract

We consider a scheduling problem in which a bounded number of jobs can be processed simultaneously by a single machine. The input is a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$. Each job, J_j, is associated with an interval $[s_j, c_j]$ along which it should be processed. Also given is the parallelism parameter $g \geq 1$, which is the maximal number of jobs that can be processed simultaneously by a single machine. Each machine operates along a contiguous time interval, called its *busy interval*, which contains all the intervals corresponding to the jobs it processes. The goal is to assign the jobs to machines so that the total busy time is minimized.

The problem is known to be NP-hard already for $g = 2$. We present a 4-approximation algorithm for general instances, and approximation algorithms with improved ratios for instances with bounded lengths, for instances where any two intervals intersect, and for instances where no interval is properly contained in another. Our study has application in optimizing the switching costs of optical networks.

⋆ A preliminary version of this work has been published in the Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009).

Email addresses: flammini@di.univaq.it (Michele Flammini), gianpiero.monaco@di.univaq.it (Gianpiero Monaco), moscardelli@sci.unich.it (Luca

1 Introduction

1.1 Problem Statement

Job scheduling on parallel machines has been widely studied (see, e.g., the surveys in [3,5]). In particular, much attention was given to *interval scheduling* [15], where jobs are given as intervals on the real line, each representing the time interval in which a job should be processed; each job has to be processed on some machine, and it is commonly assumed that a machine can process a *single* job at any time.

In this paper we consider interval scheduling with *bounded parallelism*. Formally, the input is a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$. Each job, J_j, is associated with an interval $[s_j, c_j]$ in which it should be processed. Also, given is the parallelism parameter $g \geq 1$, which is the maximal number of jobs that can be processed simultaneously by a single machine. At any given point t in time a machine M_i is said to be busy if there is at least job J_j scheduled to it such that $t \in [s_j, c_j]$, otherwise the machine is said to be idle at time t. We call the time period in which a machine M_i is busy, its busy period and denote its length by $busy_i$. The goal is to assign the jobs to machines such that the total busy time of the machines, given by $\sum_i busy_i$, is minimized. W.l.o.g the busy period of a machine M_i is contiguous, because otherwise we can divide the busy period to contiguous intervals and assign the jobs of each contiguous interval to a different machine. Obviously this will not change to total busy time. Therefore we say that a machine M_i has a *busy interval*, which starts at the minimum start time of any job scheduled on M_i and ends at the maximum completion time of any of these jobs. Note that the number of machines to be used is part of the output of the algorithm (and can take any integral value $m \geq 1$).

Our study is motivated from a central problem in optical network design, namely, assigning wavelengths to a given set of lightpaths so as to minimize the total switching cost of the network (we elaborate on that in Section 4). In fact, our scheduling problem was first considered in [16] as a problem of *fiber minimization*. The paper shows that the problem is NP-hard already for $g = 2$.

The problem of interval scheduling with bounded parallelism naturally arises in systems where service costs depend on the busy times (or, utilization) of the machines/servers. We

Moscardelli), hadas@cs.technion.ac.il (Hadas Shachnai), cmshalom@telhai.ac.il (Mordechai Shalom), tami@idc.ac.il (Tami Tamir), zaks@cs.technion.ac.il (Shmuel Zaks).

note that when the objective is to minimize the number of machines needed to complete all jobs our problem is polynomially solvable. Indeed, given an instance of the problem, we can initially solve the minimum coloring problem on interval graphs; then, we scan the jobs by color classes and schedule the next g colors classes on a single machine. Thus, a k-coloring induces a schedule on $\lceil k/g \rceil$ machines.

Throughout the paper, we refer to our scheduling problem also as the following graph theoretic problem. An *interval graph* is the intersection graph of a set of intervals on the real line. It has one vertex for each interval in the set, and an edge between every pair of vertices corresponding to intersecting intervals. In our settings, each vertex corresponds to a job, and there is an edge between two jobs whose processing times overlap. Let $H = (\mathcal{J}_H, \mathcal{E}_H)$ be a subgraph of an interval graph, induced by the set of vertices $\mathcal{J}_H$, then *the cost of* H is given by $\max_{j \in \mathcal{J}_H} c_j - \min_{j \in \mathcal{J}_H} s_j$. Our problem can be stated as the following *graph partitioning* problem. Given an interval graph, partition its vertices into subsets (i.e., assign the jobs to machines) such that: (i) the maximal size of a clique in each set is at most g, for some $g \geq 1$, and (ii) the total cost of the subgraphs is minimized.

1.2 Related Work

Some of the earlier work on interval scheduling considers the problem of scheduling a feasible subset of jobs whose total weight is maximized, i.e., a *maximum weight independent set* (see, e.g., [1] and the comprehensive survey in [14]). There has been earlier work also on the problem of scheduling *all* the jobs on a set of machines so as to minimize the total cost (see, e.g., [2]), but in these works the cost of scheduling each job is *fixed*. In our problem, the cost of scheduling each of the jobs depends on the number of other jobs scheduled on the same machine in the corresponding time interval; thus, it may change over time and among different machines.

Our study relates also to *batch scheduling* of conflicting jobs, where the conflicts are given as an interval graph. In batch scheduling [1] a set of jobs can be processed jointly. All the jobs in the batch start simultaneously, and the completion time of a batch is the last completion time of any job in the batch. (For known results on batch scheduling, see e.g., [3].) Our scheduling problem differs from batch scheduling in several aspects. While each machine can process g jobs simultaneously, for some $g \geq 1$, the jobs need not be partitioned to batches, i.e., each job can start at different time. Also, while in known batch scheduling problems the set of

[1] We refer here to the p-batch scheduling model (see e.g. Chapter 8 in [3]).

machines is given, we assume that *any* number of machines can be used for the solution. Finally, while common measures in batch scheduling refer to the maximum completion time of a batch, or a function of the completion times of the jobs, we consider the total busy times of the machines.

Our approximation algorithm for proper interval graphs (see in Section 3) solves as a subroutine a *clique partitioning* problem in interval graphs. There has been earlier work, more generally, on the problem of partitioning a graph to cliques, however, the objective functions are different than ours (see, e.g., [11]).

Finally, as mentioned above, the complexity of our scheduling problem was studied in [16]. We are not aware of earlier studies that present approximation algorithms for this problem.

Our results have immediate consequences on the regenerator minimization problem in optical networks when the topology under consideration is a path (see Section 4). In [7] we generalize the results implied by this work to any network topology.

1.3 Our contribution

In Section 2 we give a 4-approximation algorithm for general inputs. In Section 3 we present algorithms with improved ratios for some special cases. Specifically, we give 2-approximation algorithm for instances where no interval is properly contained in another (i.e., the input forms a *proper* interval graph), and a $(2+\varepsilon)$-approximation for bounded length instances. In the Appendix we give a 2-approximation algorithm for instances where any two intervals intersect. While an algorithm with the same approximation ratio was given in [6], the present algorithm as well as the analysis are different. We believe it may find uses in solving our problem for other subclasses of instances. In Section 4 we discuss the application of our results to optimizing the switching costs in a linesystem optical network.

1.4 Preliminaries

Unless specified otherwise, we use lowercase letters for indices and upper case letters for jobs, time intervals and machines; also, calligraphic characters are used for sets of jobs, sets of intervals and sets of machines.

Definition 1.1 *Given a time interval $I = [s, c]$, the length of I is $len(I) = c - s$. This*

extends to a set $\mathcal{I}$ of intervals; namely, the length of $\mathcal{I}$ is $len(\mathcal{I}) = \sum_{I \in \mathcal{I}} len(I)$.

Definition 1.2 *For a set $\mathcal{I}$ of intervals we define the span of $\mathcal{I}$ as $span(\mathcal{I}) = len(\cup \mathcal{I})$.*

Note that $span(\mathcal{I}) \leq len(\mathcal{I})$ and equality holds if and only if $\mathcal{I}$ is a set of pairwise disjoint intervals. Because jobs are given by time intervals, we use the above definitions also for jobs and sets of jobs, respectively.

Definition 1.3 *For any instance $\mathcal{J}$ and parallelism parameter $g \geq 1$, $OPT(\mathcal{J})$ denotes the cost of an optimal solution, that is, a solution in which the total busy time of the machines is minimized.*

The next observation gives two immediate lower bounds for the cost of any solution.

Observation 1.1 *For any instance $\mathcal{J}$ and parallelism parameter $g \geq 1$, the following bounds hold:*

- *The parallelism bound: $OPT(\mathcal{J}) \geq \frac{len(\mathcal{J})}{g}$.*
- *The span bound: $OPT(\mathcal{J}) \geq span(\mathcal{J})$.*

The parallelism bound holds since g is the maximum parallelism that can be achieved in any solution. The span bound holds since at any time $t \in \cup \mathcal{J}$ at least one machine is working.

W.l.o.g., we assume that the interval graph induced by the jobs is connected; otherwise, the problem can be solved by considering each connected component separately. Clearly, in any optimal solution, no machine is busy during intervals with no active jobs. As mentioned in Subsection 1.1 w.l.o.g. we assume that each machine is busy along a contiguous interval.

Given any solution we denote by $\mathcal{J}_i$ the set of jobs assigned to machine M_i by the solution. As explained in Section 1.4, the cost of M_i is the length of its busy interval, i.e. $busy_i = span(\mathcal{J}_i)$.

2 Approximation algorithm for the general case

In this section we present an algorithm for general instances and show that its approximation ratio is between 3 and 4.

2.1 Algorithm FirstFit

Algorithm FirstFit schedules the jobs greedily by considering them one after the other, from longest to shortest. Each job is scheduled to the first machine it can fit.

1. Sort the jobs in non-increasing order of length, i.e., $len(J_1) \geq len(J_2) \geq \ldots \geq len(J_n)$.
2. Consider the jobs by the above order: assign the next job, J_j, to the first machine that can process it, i.e., find the minimum value of $i \geq 1$ such that, at any time $t \in J_j$, M_i is processing at most $g - 1$ jobs. If no such machine exists, then open a new machine for J_j.

2.2 Analysis of FirstFit

2.2.1 Upper Bound

We show that algorithm FirstFit is a 4-approximation algorithm. Formally,

Theorem 2.1 *For any instance $\mathcal{J}$,*

$$FirstFit(\mathcal{J}) \leq 4 \cdot OPT(\mathcal{J}).$$

The proof is based on the following observation depicted in Figure 1.

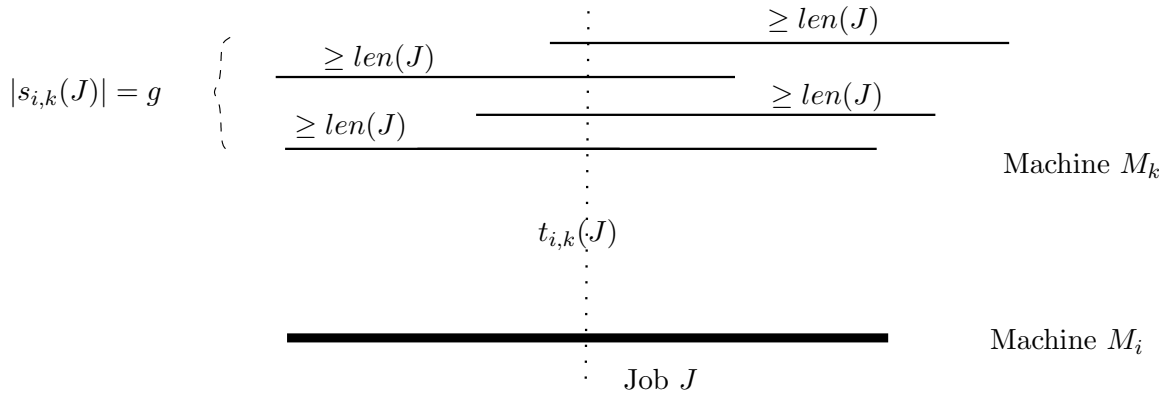

Fig. 1. Basic observation

Observation 2.2 *Let J be a job assigned to machine M_i by FirstFit, for some $i \geq 2$. For any machine $M_k, (k < i)$, there is at least one time $t_{i,k}(J) \in J$ and a set $s_{i,k}(J)$ of g jobs assigned to M_k such that, for every $J' \in s_{i,k}(J)$, it holds that (a) $t_{i,k}(J) \in J'$, and (b) $len(J') \geq len(J)$.*

Proof: In order to prove property (a), assume by contradiction that for any $t \in J$ the machine M_k is processing at most $g-1$ jobs. Since the algorithm assigns jobs to machines incrementally (i.e. it never un-assigns jobs), this property was true also when FirstFit scheduled job J. Thus, J should have been assigned to M_k by FirstFit, if it was not already assigned to a machine with smaller index. A contradiction to the fact that J is assigned to M_i. Property (b) follows from property (a) and the fact that the jobs are considered by the algorithm in a non-increasing order of their lengths. ∎

Now we present an overview of the upper bound analysis. The proof combines the span bound and the parallelism bound given in Observation 1.1 with the following analysis. Using Observation 2.2 we relate the cost incurred by FirstFit for the jobs in $\mathcal{J}_{i+1}$ to $len(\mathcal{J}_i)$. This relates the overall cost incurred by FirstFit for the jobs in $\mathcal{J} \setminus \mathcal{J}_1$ to OPT, using the parallelism bound. Then we relate the cost incurred by FirstFit for the jobs in $\mathcal{J}_1$ to OPT, using the span bound. We now proceed to the details of the proof.

For two distinct machines M_i and M_k, where $k < i$, by choosing a value among the possible values for $t_{i,k}(J)$, we obtain a pair of functions $t_{i,k} : \mathcal{J}_i \mapsto \cup\mathcal{J}_k$, and $s_{i,k} : \mathcal{J}_i \mapsto 2^{\mathcal{J}_k}$. The inverse of $t_{i,k}$ is defined naturally, as follows. For any time $t \in \cup\mathcal{J}_k$, $t_{i,k}^{-1}(t)$ is the set of all jobs $J \in \mathcal{J}_i$ such that $t_{i,k}(J) = t$. For any job $J \in \mathcal{J}_k$ we define $t_{i,k}^{-1}(J) = \bigcup_{t \in J} t_{i,k}^{-1}(t)$. We use t_i (resp. s_i) as a shorthand for $t_{i,i-1}$ (resp. $s_{i,i-1}$); moreover, we omit the index i whenever it is clear from the context and simply write t (resp. s). In particular, if $J \in \mathcal{J}_i$ the index i is not needed in $t_i(J)$, and we use for short $t(J)$.

The following key lemma is needed in the proof of Theorem 2.1:

Lemma 2.3 *For any $i \geq 1$,*

$$len(\mathcal{J}_i) \geq \frac{g}{3} span(\mathcal{J}_{i+1}).$$

Proof: Consider the maximal subset $\mathcal{J}'_{i+1}$ of proper intervals of $\mathcal{J}_{i+1}$, obtained by removing any job that is completely contained in another job. By this construction we have $\forall J, J' \in \mathcal{J}'_{i+1}, J \not\subseteq J'$ and $\cup\mathcal{J}'_{i+1} = \cup\mathcal{J}_{i+1}$, thus $span(\mathcal{J}'_{i+1}) = span(\mathcal{J}_{i+1})$. It follows that $\forall J, J' \in \mathcal{J}'_{i+1}$ J starts before J' if and only if J ends before J'. Denote by $\prec$ the total order on $\mathcal{J}'_{i+1}$ implied by the start (or completion) times of its jobs.

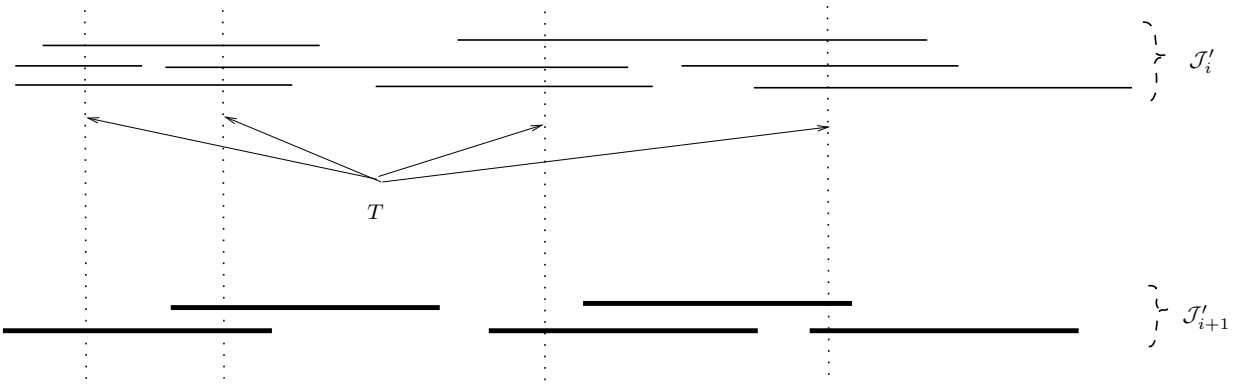

Fig. 2. Lemma 2.3 setting

Let $T = \{t(J)|J \in \mathcal{J}'_{i+1}\}$. Let also $\mathcal{J}'_i = \bigcup_{J \in \mathcal{J}'_{i+1}} s(J)$. Note that $\mathcal{J}'_i \subseteq \mathcal{J}_i$. By Observation 2.2, each $t \in T$ is contained in at least g jobs of $\mathcal{J}'_i$; on the other hand, by the algorithm, t is contained in at most g jobs of $\mathcal{J}_i$. Therefore, each $t \in T$ is contained in exactly g jobs of $\mathcal{J}'_i$ (see Figure 2).

It is easy to show that $\mathcal{J}'_i$ can be partitioned into exactly g sets $\mathcal{J}'_{i,1}, \mathcal{J}'_{i,2}, ..., \mathcal{J}'_{i,g}$ such that each set $\mathcal{J}'_{i,l}$ is an independent set with respect to intersection, or in other words an independent set of the interval graph. This can be done by considering the g threads of execution of machine M_i, and by scheduling the jobs of $\mathcal{J}'$ to these threads.

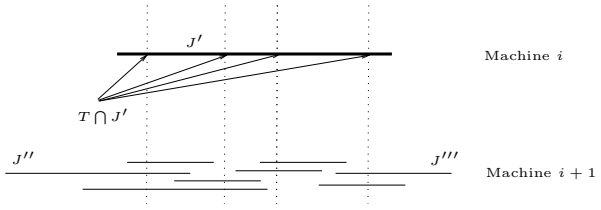

Fig. 3. Analysis of a job J' assigned to machine M_i

Now, let us consider a job $J' \in \mathcal{J}'_{i,l}$ for some $1 \leq l \leq g$ (see Figure 3). By the construction, $J' \cap T \neq \emptyset$, therefore $t^{-1}(J') \neq \emptyset$. Let J'' (resp. J''') be the smallest (resp. biggest) job in $t^{-1}(J')$ with respect to the relation $\prec$. It is easy to conclude that

$$span(t^{-1}(J')) \leq len(J') + len(J'') + len(J''').$$

Moreover, $len(J') \geq max(len(J''), len(J'''))$ because $J' \in s(J'') \cap s(J''')$. Therefore:

$$span(t^{-1}(J')) \leq 3len(J').$$

Now, we use the union bound and sum up the above for all the jobs $J' \in \mathcal{J}'_{i,l}$ to get

$$span(\bigcup_{J' \in \mathcal{J}'_{i,l}} t^{-1}(J')) \leq \sum_{J' \in \mathcal{J}'_{i,l}} span(t^{-1}(J'))$$
$$\leq 3 \sum_{J' \in \mathcal{J}'_{i,l}} len(J') = 3 \cdot len(\mathcal{J}'_{i,l}). \quad (1)$$

We note that any set $\mathcal{J}'_{i,l}$ contains all the points of T. Thus,

$$\bigcup_{J' \in \mathcal{J}'_{i,l}} t^{-1}(J') = \mathcal{J}'_{i+1} \quad (2)$$

Combining (1) and (2) we conclude that $span(\mathcal{J}'_{i+1}) \leq 3 \cdot len(\mathcal{J}'_{i,l})$. Summing for all g possible values of l, we have that

$$g \cdot span(\mathcal{J}_{i+1}) = g \cdot span(\mathcal{J}'_{i+1})$$
$$\leq 3 \sum_{l=1}^{g} len(\mathcal{J}'_{i,l}) = 3 \cdot len(\mathcal{J}'_i) \leq 3 \cdot len(\mathcal{J}_i)$$

■

Combining the span bound and Lemma 2.3 we can now complete the analysis of the algorithm.

Proof of Theorem 2.1: By definition, all the jobs in $\mathcal{J}_{i+1}$ are assigned to one machine, i.e. M_{i+1}. For such a set the cost of the assignment is exactly its span. Thus, $FirstFit(\mathcal{J}_{i+1}) = busy_{i+1} = span(\mathcal{J}_{i+1}) \leq \frac{3}{g} len(\mathcal{J}_i)$. Let $m \geq 1$ be the number of machines used by FirstFit. Then

$$\sum_{i=2}^{m} FirstFit(\mathcal{J}_i) = \sum_{i=1}^{m-1} FirstFit(\mathcal{J}_{i+1})$$
$$\leq \frac{3}{g} \sum_{i=1}^{m-1} len(\mathcal{J}_i)$$
$$< \frac{3}{g} \sum_{i=1}^{m} len(\mathcal{J}_i) = \frac{3}{g} len(\mathcal{J}) \leq 3 \cdot OPT(\mathcal{J})$$

where the last inequality follows from the parallelism bound.

Now, using the span bound, we have that $FirstFit(\mathcal{J}_1) = busy_1 = span(\mathcal{J}_1) \leq span(\mathcal{J}) \leq OPT(\mathcal{J})$. Therefore, $FirstFit(\mathcal{J}) \leq 4 \cdot OPT(\mathcal{J})$. ■

2.2.2 Lower Bound

Theorem 2.4 *For any $\varepsilon > 0$, there are infinitely many instances $\mathcal{J}$ having infinitely many input sizes, such that FirstFit$(\mathcal{J}) > (3-\varepsilon) \cdot OPT(\mathcal{J})$.*

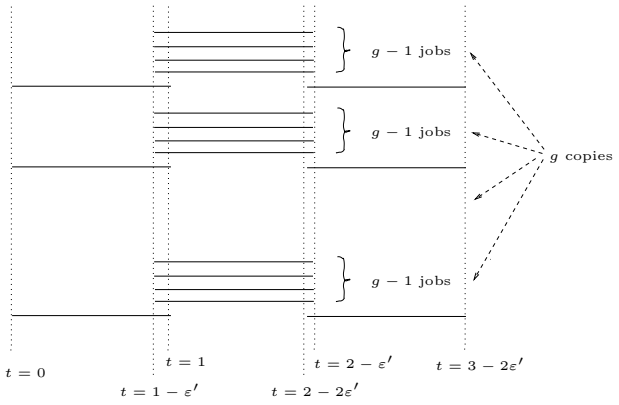

Fig. 4. Instance for the proof of the lower bound

Proof: Consider the instance $\mathcal{J}$ depicted in Figure 4. For this instance OPT uses one machine in the interval $[0,1]$, one machine in the interval $[2-2\varepsilon', 3-2\varepsilon']$, and $g-1$ machines in the interval $[1-\varepsilon', 2-\varepsilon']$, for a total cost of $OPT(\mathcal{J}) = g+1$. In contrast, FirstFit may use g machines in the interval $[0, 3-2\varepsilon']$ for a total cost of FirstFit$(\mathcal{J}) = (3-2\varepsilon')\frac{g}{g+1}$. Choosing g and ε' appropriately (for example, $\varepsilon' = \varepsilon/4$ and $g \geq 6/\varepsilon - 1$) we get that FirstFit$(\mathcal{J}) > (3-\varepsilon)OPT(\mathcal{J})$. ∎

Combining Theorems 2.1 and 2.4, we get:

Theorem 2.5 *The approximation ratio of FirstFit is between 3 and 4.*

3 Improved Approximations for Special Cases

3.1 Proper Interval Graphs

In this section we consider instances in which no job interval is properly contained in another. The intersection graphs for such instances are known as *proper interval graphs*. The simple greedy algorithm consists of two steps. In the first step, the jobs are sorted by their starting times (note that in a proper interval graph this is also the order of the jobs by completion times). In the second step the jobs are assigned to machines greedily in a NextFit manner;

that is, each job is added to the currently filled machine, unless its addition is invalid, in which case a new machine is opened.

Greedy Algorithm for Proper Interval Graphs

1. Sort the jobs in non-decreasing order by their starting points, i.e., $s_1 \leq s_2 \leq \ldots \leq s_n$.
2. Scan the list of jobs in the above order. If possible, assign the next job to the currently filled machine; else (adding the job forms a $(g+1)$-clique in the currently filled machine), open a new machine for the currently scheduled job.

Theorem 3.1 *The Greedy algorithm yields a 2-approximation for proper interval graphs.*

Proof: Let N_t denote the number of jobs active at time t. Also, let M_t^O denote the number of machines active at time t in an optimal schedule, and let M_t^A denote the number of machines active at time t in the schedule output by the algorithm.

Claim 1 *For any t, $N_t \geq (M_t^A - 2)g + 2$.*

Proof: For a given $t > 0$, let $m = M_t^A$. At the time the m-th machine is opened, there are $m-1$ additional active machines. The first one processes at least one job, and the following $m-2$ machines process exactly g jobs each. Suppose that the first machine processes at time t job J. Any job J'assigned to another machine starts after J and ends after J, since the graph is proper; thus,J' is active at time t. Since there are m active machines at time t, the same count of $(m-2)g+2$ jobs is still valid (additional jobs may have started). Therefore $N_t \geq (m-2)g + 2$. ∎

Claim 2 *For any $t, M_t^O \geq M_t^A - 1$*

Proof: Clearly, for any $t \geq 0$, $M_t^O \geq \lceil N_t/g \rceil$. Using Claim 1, we get that

$$M_t^O \geq \lceil N_t/g \rceil \geq \left\lceil ((M_t^A - 2)g + 1)/g \right\rceil \geq M_t^A - 1.$$

∎

The cost of the entire schedule of an instance $\mathcal{J}$ is obtained by taking the integral of M_t^A over all values of t in $[0, span(\mathcal{J})]$; thus, we have that $ALG(\mathcal{J}) \leq OPT(\mathcal{J}) + span(\mathcal{J})$. In particular, since $OPT(\mathcal{J}) \geq span(\mathcal{J})$, this gives the statement of the theorem. ∎
We note that by a slight ranked-shift of the jobs in the middle column of the instance

described in Figure 4, we get a proper interval graph, for which algorithm FirstFit provides an approximation ratio arbitrarily close to 3.

3.2 Bounded Length Instances

Given an instance $\mathcal{J}$ of our problem, recall that $len(J_j) = c_j - s_j$ is the *length* of J_j. We now consider instances in which for all $1 \leq j \leq n$, $len(J_j) \in [1, d]$, where $d \geq 1$ is some fixed constant. We further assume that the start times of all jobs are integral. [2]

Bounded_Length Algorithm

1. Partition the input to segments as follows. All intervals I_j for which $s_j \in [d \cdot (r-1), d \cdot r)$ for some integer $r \geq 1$ are assigned to segment r. Let $S(\mathcal{J})$ be the resulting set of segments and $R = |S(\mathcal{J})|$.
2. For each segment $1 \leq r \leq R$ do
   (a) Guess $k_r \geq 1$, the number of machines in some optimal schedule for segment r, and OPT_r, the minimum total busy time of these machines.
   (b) Guess the vector of busy intervals of the machines allotted to segment r, where the busy interval of machine i is given by $(s(M_i), busy_i)$: $s(M_i) \in [d \cdot (r-1), d \cdot r)$ is the start time of the busy interval, and $busy_i = (1+\varepsilon)^m$, for some $0 \leq m \leq \lceil \log_{1+\varepsilon}(2d) \rceil$, is the length of the busy interval.
   (c) Guess a partition of the intervals in segment r to *independent sets (IS)*, i.e., the vector containing the number of ISs of each type assigned to the machines. Let $N_r \leq k_r \cdot g$ be the total number of ISs in the solution.
   (d) Construct a bipartite graph $B = (U, V, E)$, where $U = (M_1, \ldots, M_{k_r})$ and $V = (IS_1, \ldots, IS_{N_r})$. There is an edge (M_i, IS_h) if IS_h can be scheduled on M_i. For any vertex $M_i \in U$ let $b(M_i) = g$, and for any $IS_h \in V$ let $b(IS_h) = 1$.
   (e) Solve for B the maximum b-matching problem, and assign the jobs to the machines as determined by the matching.

Algorithm Bounded_Length accepts as parameter some $\varepsilon > 0$. In Step 2. of the algorithm we use *guessing*. By this we refer to enumeration over the set of possible values, using a polynomial number of steps. Also, in Step 2(e) we solve the *b-matching* problem, defined as

[2] Such instances show up, e.g., in optical networks, where the jobs represent lightpaths between pairs of nodes (see Section 4).

follows. Given a graph $G = (V, E)$ with degree constraints $b : V \to \mathbb{N}$ for the vertices, a b-matching is a subset of edges $M \subseteq E$ such that for all $v \in V$ the number of edges in M incident to v is at most $b(v)$. The b-matching problem is to find a b-matching M of maximum size. The problem is known to be solvable in polynomial time (see, e.g., [9]).

Theorem 3.2 *The Bounded_Length Algorithm is a polynomial time $(2+\varepsilon)$-approximation algorithm for the scheduling problem.*

We use in the proof the next lemma.

Lemma 3.3 *Let $OPT(\mathcal{J})$ be the value of an optimal solution for $\mathcal{J}$, and let $OPT(S(\mathcal{J}))$ be the value of an optimal solution for $\mathcal{J}$ in which jobs from different segments in $S(\mathcal{J})$ are not assigned to the same machine. Then $OPT(S(\mathcal{J})) \leq 2 \cdot OPT(\mathcal{J})$.*

Proof: The proof is by construction. Given an optimal solution for $\mathcal{J}$, construct a solution in which jobs from different segments are not assigned to the same machine, such that the total busy time of the resulting schedule is at most twice the total busy time of the given schedule. For each machine in the optimal solution that spans over jobs from k adjacent segments in $S(\mathcal{J})$, simply replace the machine by k machines, each processing only the jobs of a single segment.

Claim 3 *Any single machine M_i of OPT with busy time $busy_i$ is replaced by a set of machines having total busy time at most $2busy_i$.*

Proof: Let $S_1, S_2, \ldots$ be the segments formed in Step 1., ordered from left to right. Consider a machine M_i of OPT, and assume that it covers jobs from $S_{i_1}, \ldots, S_{i_1+k-1}$. It is therefore replaced by k machines, each processing the jobs of a different segment. Denote these machines by $M_{i_1}, \ldots, M_{i_1+k-1}$. Note that the busy intervals of two adjacent machines may intersect. However, for any $r \in \{i_1, \ldots, i_1 + k - 3\}$, the busy intervals of M_r, M_{r+2} do not intersect. This follows from the fact that the length of any interval is at most d. In other words, the completion time of any job in $\mathcal{J}_r$ is in $[d \cdot (r-1), d \cdot (r+1))$.

By the above discussion, the total busy time of the machine M_i is at least the total busy time of the even-indexed replacing machines, as well as at least the busy time of the odd-indexed replacing machines. Formally, assume w.l.o.g that k is even, then $busy_i \geq busy_{i_1} + busy_{i_1+2} + \ldots + busy_{i_1+k-2}$, and also $busy_i \geq busy_{i_1+1} + busy_{i_1+3} + \ldots + busy_{i_1+k-1}$. Sum up the two inequalities to get $2busy_i \geq \sum_{\ell=i_1}^{i_1+k-1} busy_\ell$ as required. ∎

The statement of the lemma follows by summing up the busy times of all the machines

replacing the machines of OPT. ∎

Proof of Theorem 3.2: We first show that, given any $\varepsilon > 0$, Step 2. of the Bounded_Length algorithm yields in polynomial time a $(1+\varepsilon)$-approximation for the minimum busy time of segment r, for any $1 \leq r \leq R$. Formally, let $busy_i^o$ be the busy time of M_i in some optimal schedule for segment r, and suppose that $(1+\varepsilon)^h \leq busy_i^o \leq (1+\varepsilon)^{h+1}$, then by taking $busy_i = (1+\varepsilon)^{h+1}$ we increase the busy time of M_i at most by factor $1+\varepsilon$. Also, given a correct guess of the vector of busy intervals of the machines in segment r and the partition of the jobs in segment r to independent sets, the algorithm finds in B a b-matching in which all the IS vertices are matched. Thus, using Lemma 3.3, and taking in the algorithm $\varepsilon' = \varepsilon/2$, we get a $(2+\varepsilon)$-approximation ratio.

For the running time of the algorithm, we note that the optimal total busy time for any segment $1 \leq r \leq R$ satisfies $1 \leq OPT_r \leq n \cdot d$ and can be guessed within factor $1+\varepsilon$ in $O(\ln n/\varepsilon)$ steps. Similarly, the total number of machines assigned to segment r is at most n, and can be guessed in $O(\log n)$ steps. Indeed, we can obtain this way the number of machines used in some optimal solution, by guessing a value $OPT_r \leq k \leq 2OPT_r$ and setting the busy intervals of machines $OPT_r + 1, \ldots, k$ to be of length zero. Also, since for any $1 \leq i \leq k$, $s(M_i)$ can get at most d values, and $busy_i$ can get $\log_{1+\varepsilon}(2d) = O(\ln d/\varepsilon)$ possible values (since all job lengths are in $[1, d]$), the busy intervals vector can be guessed in polynomial time.

Finally, the vector of independent sets assigned to the machines in segment r can be guessed in polynomial time. Indeed, any independent set consists of at most d intervals. The number of possible starting points for these intervals is d. Also, for a given independent set , IS_ℓ, let $s(IS_\ell)$ be the earliest start time of any interval in IS_ℓ, and let $c(IS_\ell)$ be the latest completion time of any interval in IS_ℓ, then the *stretch* of IS_ℓ is $St(IS_\ell) = c(IS_\ell) - s(IS_\ell)$. In guessing the stretch values for the independent sets, we round up the stretch of any IS to the nearest integral power of $1+\varepsilon$. Thus, the number of possible stretch values is $O(\ln d/\varepsilon)$. For each independent set IS_ℓ, we keep the start times of all intervals, as well as the stretch of IS_ℓ, i.e., the vector $(s_{i_1}, \ldots s_{i_t}, St(IS_\ell))$, where $1 \leq t \leq d$. The number of starting points for the intervals in IS_ℓ is given by

$$\sum_{t=1}^{d} \binom{d+t-1}{t-1} \leq d\binom{2d-1}{d-1} \leq \left(\frac{(2d-1)e}{d-1}\right)^{d-1} \leq d(2e)^d,$$

where e is the base of the natural logarithm. Since the stretch of any independent set can take $O(\ln d/\varepsilon)$ values, we have that the number of possible IS vectors is $O(d \cdot (2e)^d \ln d/\varepsilon)$, which is a constant. Hence,we can guess in polynomial time the set of ISs assigned to the machines

of segment r. Clearly, all other steps of the algorithm can be implemented in polynomial time. ∎

4 Application to Optical Networks

4.1 Background

All-optical networks have been widely studied in recent years, due to the promise of data transmission rates several orders of magnitudes higher than current networks [12,13]. The focus of optimality criteria lies in the hardware costs. This is modeled by considering the utilization of the electronic switching units (Add-Drop Multiplexers, or ADMs) and the signal *regenerators*. The communication is done by *lightpaths*, which are simple paths in the network. A lightpath connecting nodes u and v is using one ADM in each of its endpoints u and v, and one regenerator in each intermediate node. An ADM at a node can serve two lightpaths of the same wavelength, provided they do not have any edge in common.

Often, the traffic supported by the network requires transmission rates which are lower than the full wavelength capacity; thus, the network operator has to be able to put together (or, groom) low-capacity demands into the high capacity fibers. Taking g to be the *grooming* factor, for some $g \geq 1$, this can be viewed as assigning colors to the lightpaths so that at most g of them can share one edge.

In terms of ADMs, this means that at most g lightpaths of the same wavelength can enter through the same edge to any node and use the same ADM (thus saving $g-1$ ADMs). Similarly, in terms of regenerators, if g lightpaths of the same wavelength need such a component at the same node, they can share a single regenerator (thus saving $g-1$ regenerators). The goal is to minimize the cost measured by $\alpha \cdot |REGENERATORs| + (1-\alpha) \cdot |ADMs|$, for any value $0 \leq \alpha \leq 1$. This *grooming prblem* has become central in optimizing switching costs for optical networks.

The notion of traffic grooming was introduced in [10] for ring topologies. The case where $\alpha = 0$, namely, minimizing the number of ADMs, is the topic of many previous studies. Our results for scheduling with bounded parallelism apply to the case where $\alpha = 1$, where the goal is to minimize the number of regenerators. The ADMs minimization problem was shown to be NP-complete in [4] for rings and for arbitrary values of $g \geq 1$. An algorithm

with approximation ratio of $2 \ln g$, for any fixed $g \geq 1$ on a ring topology, was given in [8]. The problem of minimizing the number of regenerators was shown to be NP-complete in [6] (see also in [16]). The paper [6] also presents an approximation algorithm for the problem of minimizing every linear combination of the total number of ADMs and regenerators. The algorithm is polynomial in the input size (and in g) and yields an approximation ratio of $O(\sqrt{g} \log n)$.

4.2 Results

Given a network with a path topology, a grooming factor $g \geq 1$ and a set of lightpaths $\{p_j = (a_j, b_j) | j = 1, ..., n\}$, we need to color the lightpaths such that the number of regenerators is minimized. This yields the following instance of our scheduling problem. The set of jobs is $\{J_j = [a_j + 1/2, b_j - 1/2] | j = 1, ..., n\}$, and the parallelism parameter is g. Grooming up to g lightpaths in the optical network implies that the corresponding jobs are scheduled on the same machine, and vice versa. In other words, different colors in the optical network instance correspond to different machines in the scheduling problem instance, and a regenerator at node i corresponds to the interval $[i-1/2, i+1/2]$. A lightpath uses a given regenerator if and only if the corresponding job contains the interval corresponding to the given regenerator. Therefore, the cost of a coloring of the lightpaths is equal to the cost of the schedule of the corresponding set of jobs.

Thus, we have the following results for the grooming problem. (*i*) A 4-approximation algorithm for an arbitrary set of lightpaths (following Section 2); (*ii*) a 2-approximation algorithm for instances where any two lightpaths intersect (this follows from the algorithm for cliques in the Appendix, and also from [6]; see discussion in Section 1.3), (*iii*) a 2-approximation algorithm for instances where no lightpath is properly contained in another (following Section 3.1), and (*iv*) a $(2+\varepsilon)$- approximation algorithms for instances in which the ratio between the lengths of any two lightpaths is bounded by d, where $d \geq 1$ is some fixed constant (following Section 3.2).

References

[1] R. Bar-Yehuda A. Bar-Noy, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, pages 1–23, 2000.

[2] S. Bhatia, J. Chuzhoy, A. Freund, and J. Naor. Algorithmic aspects of bandwidth trading. *ACM Transactions on Algorithms*, 3(1), 2007.

[3] P. Brucker. *Scheduling Algorithms, 5th ed.* Springer, 2007.

[4] A. L. Chiu and E. H. Modiano. Traffic grooming algorithms for reducing electronic multiplexing costs in wdm ring networks. *Journal of Lightwave Technology*, 18(1):2–12, January 2000.

[5] J. Y-T. Leung (ed.). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRS Press, 2004.

[6] M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem with respect to adms and oadms. In *14th International European Conference on Parallel and Distributed Computing (EuroPar), Las Palmas de Gran Canaria, Spain, August 26-29*, 2008.

[7] M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Optimizing regenerator cost in traffic grooming. *Submitted for publication*, 2010.

[8] M. Flammini, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem. In *ISAAC*, pages 915–924, 2005.

[9] H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC*, pages 448–456, 1983.

[10] O. Gerstel, R. Ramaswami, and G. Sasaki. Cost effective traffic grooming in wdm rings. In *INFOCOM'98, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 1998.

[11] D. Gijswijt, V. Jost, and M. Queyranne. Clique partitioning of interval graphs with submodular costs on the cliques. *RAIRO-Operations Research-Recherche Opérationelle*, 41(3):275–287, 2007.

[12] P. E. Green. *Fiber-Optic Communication Networks.* Prentice Hall, 1992.

[13] R. Klasing. Methods and problems of wavelength-routing in all-optical networks. In *Proceeding of the MFCS'98 Workshop on Communication, August 24-25, Brno, Czech Republic*, pages 1–9, 1998.

[14] M. Y. Kovalyov, C. T. Ng, and T. C. E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178(2):331–342, 2007.

[15] E. Lawler, J.K. Lenstra, A.H.G.R. Kan, and D. Shmoys. Sequencing and scheduling: Algorithms and complexity. *S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin (eds.), Handbooks in Operations Research and Management Science*, 4, 1993.

[16] P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. In *SODA*, pages 830–831, 2003.

Appendix: Approximation Algorithm for Cliques

If all jobs of an instance intersect with each other, then the resulting interval graph is a clique. In this section we present a 2-approximation algorithm for this class. For a given clique C, select an arbitrary point t which belongs to all the intervals. Such a point must exist by the definition of a clique in an interval graph. For a fixed $t \geq 0$, for each job j, let $\delta_j = \max(t - s_j, c_j - t)$ be the maximal distance of an endpoint of j from the point t. Also, let $\Delta = \max_{j \in C} \delta_j$ (see Figure 5).

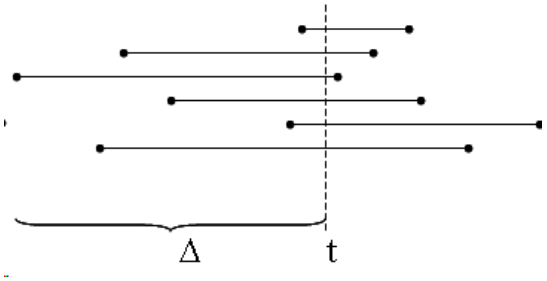

Fig. 5. The left-right partition of a clique for an arbitrary intersection point t.

Scheduling Algorithm for a Clique
1. Sort the jobs in the clique C by their distance from t such that $\delta_1 \geq \ldots \geq \delta_{|C|}$.
2. While not all jobs are assigned to machines, open a new machine and assign to it the next g jobs (the last machine may be assigned less than g jobs).

The algorithm schedules the jobs on $\left\lceil \frac{|C|}{g} \right\rceil$ machines: the g jobs with the maximal δ_j values are assigned to M_1, and so on.

Theorem 4.1 *Let $OPT(C)$ be the total busy time in an optimal solution for C, then the total busy time of the machines used by the algorithm is at most $2 \cdot OPT(C)$.*

Proof: Let $m = \left\lceil \frac{|C|}{g} \right\rceil$ be the number of machines used by the algorithm. Clearly, the number of machines used in any optimal solution is $m' \geq m$.

For any machine M_i in some optimal solution, let $\delta_O^i = \max_{\{j \in \mathcal{J}_i\}} \delta_j$, where $\mathcal{J}_i$ is the set of jobs assigned to M_i. Sort the machines of the optimal solution such that $\delta_O^1 \geq \delta_O^2 \geq \ldots \geq \delta_O^{m'}$. Since M_i processes at least one job of length at least δ_O^i, it holds that $busy_i \geq \delta_O^i$. Thus, $OPT(C) \geq \sum_{i=1}^{m'} \delta_O^i$.

Similarly, for any machine M_i used by the algorithm, let $\delta_A^i = \max_{\{j \in \mathcal{J}_i\}} \delta_j$. Sort the machines used by the algorithm such that $\delta_A^1 \geq \delta_A^2 \geq \ldots \geq \delta_A^m$. Note that this is also the order in which the machines were filled. The busy interval of M_i is therefore contained in $[t - \delta_A^i, t + \delta_A^i]$, implying that $busy_i \leq 2\delta_A^i$, and thus, $ALG(C) \leq 2\sum_{i=1}^{m} \delta_A^i$. The following claim completes the proof.

Claim 4 $\sum_{i=1}^{m} \delta_A^i \leq \sum_{i=1}^{m'} \delta_O^i$.

Proof: It suffices to show that for any $1 \leq i \leq m$, $\delta_A^i \leq \delta_O^i$. For $i = 1$, both δ_A^1 and δ_O^1 are equal to Δ. For any $i > 1$, note that the $i-1$ first machines filled by the algorithm are assigned the $g(i-1)$ jobs with the maximal δ_j values. Therefore, the remaining $n - g(i-1)$ jobs have the minimal maximal distance from point t among any set of $n - g(i-1)$ jobs. This distance is exactly δ_A^i. In particular, the set of $n - g(i-1)$ jobs that are assigned on the remaining machines in the optimal solution has at least the same maximal distance from t. Thus, $\forall i,\ \delta_A^i \leq \delta_O^i$. ∎