DIMACS Technical Report 99-52

Improvements in Throughput Maximization for Real-Time Scheduling

by

Piotr Berman[1] Bhaskar DasGupta[2]

[1]Computer Science & Engg. Department, Penn State, University Park, PA 16802, email: berman@cse.psu.edu. Supported in part by NSF grant CCR-9700053 and National Library of Medicine grant LM05110.
[2]Computer Science Department, Rutgers University, Camden, NJ 08102, email: bhaskar@crab.rutgers.edu. Supported in part by NSF grant CCR-9800086.

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, Telcordia Technologies (formerly Bellcore) and NEC Research Institute.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We consider the problem of off-line throughput maximization for job scheduling on one or more machines, where each job has a release time, a deadline and a profit. Most of the versions of the problem discussed here were already treated by Bar-Noy *et al.* [2]. Our main contribution is to provide algorithms that do not use linear programming, are simple and much faster than the corresponding ones proposed in [2], while either having the same quality of approximation or improving it. More precisely, compared to the results of in Bar-Noy *et al.* [2], our pseudo-polynomial algorithm for multiple unrelated machines and all of our strongly-polynomial algorithms have better performance ratios, all of our algorithms run much faster, are combinatorial in nature and avoid linear programming. Finally, we show that algorithms with better performance ratios than 2 are possible if the stretch factors of the jobs are bounded.

1 Introduction

We consider the problem of scheduling jobs with profits and time constraints which we define as (for example) in Bar-Noy *et al.* [2]. We have jobs $J_1, J_2, \ldots, J_n$ that can be performed on machines $M_1, M_2, \ldots, M_k$. The job J_i has *profit* $w_i \geq 0$, a *release time* r_i, a *deadline* d_i and a *length* (execution time) $l_{i,j}$ for each machine M_j. A schedule is a set of triples (j, m, s) where such a triple schedules job J_j to be executed on machine M_m starting at time s, and thus ending at time $s + l_{j,m}$. A schedule is valid if each job is scheduled at most once, time intervals $[s, s + l_{j,m})$ defined by the triples with the same m are pairwise disjoint (each machine M_j can execute only one job at the time) and for each such interval $[s, s + l_{j,m}) \subseteq [r_j, d_j)$ (a job can be executed only between its release time and its deadline). The *throughput* of a schedule is the sum of profits of the scheduled jobs, and the goal is to maximize the throughput. For the case of one machine, a schedule can be viewed as a set of pairs (j, s) and we refer to the respective problem with the acronym TMP, for the *throughput maximization problem*. The machines are called *identical* if $l_{i,j}$ is simply equal to l_j, i.e. a job has the same execution time on every machine; we use TMP$_k$-id to denote the throughput maximization problem for k identical machines. In the remaining case, we say that the machines are *unrelated* and use the acronym TMP$_k$-un. Notice that preemption of jobs is not allowed. Following the nomenclature of Bar-Noy *et al.* [2], we distinguish between the case when the job parameters are positive integers (the corresponding algorithms are pseudo-polynomial time) as opposed to when these parameters are arbitrary real numbers (the corresponding algorithms are strongly-polynomial time).

An characterization of the job scheduling problem, relevant to adaptive rate-controlled scheduling for multimedia and other applications [10, 8, 12], is one in which each job $J_i = (w_i, r_i, d_i, l_{i,j})$ is instead characterized as $J_i = (w_i, r_i, \alpha_{i,j}, l_{i,j})$, where $\alpha_{i,j} = (d_i - r_i)/l_{i,j}$ is the rate or stretch factor for J_i on machine M_j. An efficient approximation algorithm for this case should take into consideration the values of various $\alpha_{i,j}$'s in that the performance ratio of such algorithms should depend on the $\alpha_{i,j}$'s.

Here is a brief history of the throughput maximization problem; the reader is referred to the paper [2] for more detailed discussions. TMP, the problem for a single machine, is NP-hard even when all the jobs are released at the same time [11]; however this special case has a fully polynomial-time approximation scheme. The preemptive version of TMP was studied by Lawler [6], who has found a pseudo-polynomial time algorithm, as well as polynomial time algorithms for two important special cases. Kise, Ibaraki and Mine [4] presented solutions for the special case when the release times and deadlines are similarly ordered. On-line versions of the problem for preemptive and nonpreemptive cases were considered, among others, in [1, 5, 7].

The following notations and nomenclatures are used for the rest of this paper. An interval is of the form $[a, b)$ for two non-negative numbers $a, b \in \mathbb{R}$, denoting the set of real numbers $\{x \in \mathbb{R} : \ a \leq x < l\}$; a and b are called the beginning and the ending of the interval, respectively. Note that $[a, b)$ is empty if $b \leq a$. When our discussion concerns a set of positive integers, then for two positive integers k and l, the notation $[k, l]$ (resp., $[k, l)$) denotes the *integer interval* $\{m \in \mathbb{N} : \ k \leq m \leq l\}$ (resp., $\{m \in \mathbb{N} : \ k \leq m < l\}$). Let t

	Pseudo-polynomial algorithm			
	Bar-Noy *et al.* [2]		This paper	
	ratio	time—Ω	ratio	time—O
TMP	2	$\mathrm{LP}(tn, t+n) + t^2n^2$	2	$tn \log\log t$
TMP$_k$-un	3	$\mathrm{LP}(tnk, tk+n) + t^2n^2k$	2	$tnk \log\log(tk)$
TMP$_k$-id	$\frac{(k+1)^k}{(k+1)^k-k^k}$	$\mathrm{LP}(tnk, tk+n) + t^2n^2k$	$\frac{(k+1)^k}{(k+1)^k-k^k}$	$tnk \log\log t$

	Strongly polynomial algorithm			
	Bar-Noy *et al.* [2]		This paper	
	ratio	time—Ω	ratio	time—O
TMP	3	$\mathrm{LP}(n^4, n^3)+n^8$	$\frac{2}{1-\varepsilon}$	$\frac{n^2}{\varepsilon}$
TMP$_k$-un	4	$\mathrm{LP}(n^4k, n^3k) + n^8k$	$\frac{2}{1-\varepsilon}$	$\frac{n^2}{\varepsilon}$
TMP$_k$-id	$\frac{(2k+1)^k}{(2k+1)^k-(2k)^k}$	$\mathrm{LP}(n^4k, n^3k) + n^8k$	$\frac{(k+1)^k}{(k+1)^k-(k+\varepsilon)^k}$	$\frac{kn^2}{\varepsilon}$

Table 1: *Comparison of our algorithms with those of Bar-Noy et al.* [2]. *$LP(a, b)$ denotes the time required to solve a linear-programming problem with a variables and b inequalities, t denotes the latest deadline of any job in the pseudo-polynomial case, n denotes the number of jobs, $k > 1$ denotes the number of machines and $0 < \varepsilon < 1$ is an arbitrary value.*

denote the latest deadline of all jobs.

To solve our scheduling problems, we will first discuss a more abstract problem, termed as the *interval selection problem* or ISP, which can be formulated as follows. For each integer $i \in [1, n]$ we are given a family of integer intervals S_i and a number $w_i > 0$, so that selecting any integer interval $[d, e)$ from S_i yields a profit w_i. Our task is to select at most one interval from each set, so that the selected intervals are disjoint and the sum of profits is maximum. This problem was studied in the context of scheduling by Bar-Noy *et al.* [2] who described an algorithm with ratio 2. However, their algorithm utilizes linear programming, and therefore is much less efficient than the two-phase algorithm (2PA) proposed by Berman, Miller and Zhang in [3] (in that paper, a more general version of ISP is discussed).

In this paper we show that 2PA can be used to provide pseudo-polynomial time algorithms for all versions of scheduling problems discussed in Bar-Noy *et al.* [2] with better running times and better or same approximation ratios, and that it can be modified to provide strongly polynomial algorithms for these problems with both better running times and better

```
(* definitions *)
    interval is a quadruple of the following kind:
        (family, value, beginning, ending);
    L is sequence that contains an interval (i, w_i, d, e)
        for every integer i ∈ [1, n] and every [d, e) ∈ S_i,
        L is sorted so the value of ending is non-decreasing;
    S is an initially empty stack that stores intervals;
    TOTAL(c) returns the sum of values of those intervals on S
        that have ending > c;
    total(i, c) returns the sum of values of those intervals on S
        that have ending ≤ c and family = i;
    for ( each integer i ∈ [1, n] ) done[i] ← false;
    occupied ← t;
(* evaluation phase *)
    for ( each (i, w, d, e) from L )
    {   v ← w − total(i, d) − TOTAL(d);
        if ( v > 0 )
            push((i, v, d, e),S);
    }
(* selection phase *)
    while ( S is not empty )
    {   (i, v, d, e) ← pop(S);
        if ( not done[i] and e ≤ occupied )
            insert (i, [d, e)) to the solution,
            done[i] ← true, occupied ← d;
    }
```

Figure 1: TPA, the two-phase algorithm for ISP

approximation ratios. We also present a $2/(1+1/(2^{\lfloor\alpha\rfloor+1}-2-\lfloor\alpha\rfloor))$-approximation algorithm for the special case of the TMP problem when the stretch factor α_i of each job J_i is at most α, which is better than the 2-approximation algorithm previously known. Table 1 above summarizes our main results and compares them with the corresponding ones in [2].

2 Two-phase algorithm for ISP

Figure 2 shows 2PA for ISP, where we assume that the input consists of families of interval sets, $S_1, \ldots, S_n$ that are contained in $[1, t)$ and the notation $(i, [d, e))$ denotes the interval $[d, e) \in S_i$. Before characterizing the approximation properties of this algorithm, we will analyze its correctness and the running time. Let N denote the total number of intervals in the input. Then we can prove the following lemma.

Lemma 1 *2PA returns a correct solution to ISP problem in $O(N(1+\min\{\log N, \log\log t\}))$ time.*

Proof. To show that the algorithm is correct we need to prove that it selects at most one interval from each family and that the selected intervals are disjoint. Both properties are ensured by the selection phase. Once a interval from family S_i is selected, done$[i]$ is reset from **false** to **true**, thus preventing the future selections from S_i. Moreover, in each iteration of the selection phase [occupied,t) contains all the selected intervals, and the algorithm never selects a set that overlaps [occupied,t), thus the selected intervals are disjoint.

It is easy to see that for each of N intervals in the input the algorithm performs only constant number of operations, which are elementary except for the computation of total(i, d) and TOTAL(d) in the evaluation phase. We need to show how these functions can be computed in $O(\min\{\log N, \log\log t\})$ time.

First, we show how to compute total(i, d) efficiently. Looking at 2PA it follows that we need to maintain, for each $i \in [1, n]$, a data structure $\mathcal{D}_i$ for the endings of all the intervals belonging to S_i in the stack such that the following two operations can be performed:

Insert(i, v, d, e)**:** Insert the ending e of the interval $[d, e)$ belonging to S_i with value v in $\mathcal{D}_i$.

Query(i, v, d, e)**:** Given the interval $[d, e)$ belonging to S_i, find the sum of the values of all endings b in $\mathcal{D}_i$ with $b \leq d$.

We will maintain, for each ending e already inserted in $\mathcal{D}_i$, the quantity $e_{\mathrm{S}um}$ which is sum of values of all the endings not to its right in $\mathcal{D}_i$. Notice that when we insert an ending e in $\mathcal{D}_i$, no endings in $\mathcal{D}_i$ is to the right of e (since we scan the endings in 2PA in left to right order).

If $\log N \leq \log\log t$, then both these operations can be easily implemented in $O(\log N)$ time. $\mathcal{D}_i$ consists of a sorted list of all the endings of intervals belonging to S_i already inserted. The Insert operation simply appends the new ending e with value v to the end of $\mathcal{D}_i$ (unless it is already there) and updates the value of $e_{\mathrm{S}um}$ to be $v + e'_{\mathrm{S}um}$ where e' was the most recent ending inserted in $\mathcal{D}_i$ (if e was already there, we simply update $e_{\mathrm{S}um}$ to be its old value plus v). For the Query operation, we need to do a binary search on $\mathcal{D}_i$ in $O(\log N)$ time and retrieve the value of $e'_{\mathrm{S}um}$ for the appropriate ending e' found by the search.

If $\log N > \log\log t$, these operations can be implemented by using the vanEmdeBoas tree as described in [9]. Remember that our universe consists of integers in $[1, t]$. While inserting e with value v, we first check if e is already in $\mathcal{D}_i$. If so, we update $e_{\mathrm{S}um}$ to be the sum of its old value plus v. Otherwise, we find the leftmost neighbour of e, say e', and update $e_{\mathrm{S}um}$ to be the sum of $e'_{\mathrm{S}um}$ and v. While doing a Query, we find the leftmost neighbour w of d in $\mathcal{D}_i$ (which could be d or less), and simply retrieve the value of $w_{\mathrm{S}um}$. As a result, we achieve a time of $O(\log\log t)$ per Insert and Query operation.

The implementation of TOTAL(d) is very similar. Now, we need to maintain a single data structure $\mathcal{D}$ for all the endings currently in the stack such that the following operations can be implemented efficiently.

Insert(v, d, e)**:** Insert the ending e of the interval $[d, e)$ with value v in $\mathcal{D}$.

Query(d, e)**:** Given the interval $[d, e)$, find the sum of the values of all endings b in $\mathcal{D}$ with $b > d$.

If we now additionally store the sum of all values currently in $\mathcal{D}$ in a variable β, then the answer to Query(d, e) is β minus sum of the values of all endings b in $\mathcal{D}$ with $b \leq d$. Hence, the same techniques described previously can be used to implement theses operations in $O(\min\{\log N, \log\log t\})$ time. ❑

Now, we prove the performance ratio of 2PA. The next lemma will characterize the status of $\mathbf{S}$ at the end of the evaluation phase.

Lemma 2 *Consider a feasible solution A to the input ISP instance and the set of intervals $\mathbf{S}$ in the stack at the end of the evaluation phase. Let*

- $\mathbf{S}_i$ *be the set of entries of* $\mathbf{S}$ *with* family $= i$*;*
- $\mathbf{S}_A$ *be the union of* $\mathbf{S}_i$*'s for i's that are present in A;*
- $\mathbf{S}_{d,e}$ *be the set of entries of* $\mathbf{S}$ *with* $d <$ ending $\leq e$*;*
- $V(X)$ *be the sum of* values *of* intervals *in the set X;*
- $P(A) = \sum_{(i,[d,e))\in A} w_i$.

Then $V(\mathbf{S}) + V(\mathbf{S}_A) \geq P(A)$.

Proof. Because the intervals in A are disjoint,

$$\sum_{(i,[d,e))\in A} V(\mathbf{S}_{d,e}) \;\leq\; V(\mathbf{S}).$$

Therefore if suffices to show that for every $(i, [d, e)) \in A$ we have

$$V(\mathbf{S}_{d,e}) + V(\mathbf{S}_i) \;\geq\; w_i.$$

Let $\mathbf{S}'$ be the content of the stack at the time when the evaluation phase starts the processing of (i, w_i, d, e). At that time we compute $v = w_i - \text{total}(i, d) - \text{TOTAL}(d)$. Note that $\text{total}(i, d) \leq V(\mathbf{S}_i)$, and since all entries of $\mathbf{S}'$ have *ending* $\leq e$, we also have $\text{TOTAL}(d) = \mathbf{S}'_{d,e}$. Consequently, if $v \leq 0$, then

$$w_i \leq \text{total}(i, d) + \text{TOTAL}(d) \leq V(\mathbf{S}_i) + V(\mathbf{S}'_{d,e}) \leq V(\mathbf{S}_i) + V(\mathbf{S}_{d,e}).$$

On the other hand, if $v > 0$, then we push (i, v, d, e) onto $\mathbf{S}$ and

$$V(\mathbf{S}_{d,e}) \;\geq\; V(\mathbf{S}'_{d,e}) + v \;\geq\; V(\mathbf{S}'_{d,e}) + w_i - V(\mathbf{S}'_{d,e}) - V(\mathbf{S}_i) \;=\; w_i - V(\mathbf{S}_i).$$

❑

The next lemma shows how the profit of the solution that is found in the selection phase can be determined from the status of the stack at the end of the evaluation phase.

Lemma 3 *The sum of profits of the intervals selected during the selection phase is at least* $V(\mathbf{S})$.

Proof. With each interval inserted to the solution during the selection phase we can associate a set of stack entries. It suffices to show that (a) the sum of *values* of entries associated with $(i, [d, e))$ is at least w_i, and (b) each stack entry is associated with at least one element of the solution.

To describe the association rule, observe that if $(i, [d, e))$ is selected, then for some $v > 0$ the quadruple (i, v, d, e) was pushed onto $\mathbf{S}$ during the evaluation phase; in turn, v was computed by subtracting from w_i the sum of values of all the stack entries that have $family = i$ and $ending \leq d$ (represented as total(i, d)) or have $d < ending \leq e$ (represented as TOTAL(d)). We associate with $(i, [d, e))$ all the stack entries that were involved in the computation of v and (i, v, d, e) itself; by the very definition, w_i is the sum of their *values*.

Observe that we associate a stack entry (i', v', d', e') not in the solution with an element of the solution, say $(i, [d, e))$, if and only if one of the following holds true: (i) $i' = i$, so we set done$(i') =$ **true**, or (ii) $e' > d$, so occupied becomes $d < e'$. Thus if in the selection phase we have top of the stack (i', v', d', e') that has not been associated with a solution element, we include $(i', [d', e'))$ in the solution and associate (i', v', d', e') with $(i', [d', e'))$. Therefore each entry of $\mathbf{S}$ has to associated with an element of the solution. ❑

Now we can prove our first theorem.

Theorem 4 *2PA solves ISP problem with approximation ratio at most 2 and in time* $O(N \min\{\log N, \log\log t\})$*, where there are* N *intervals in the input contained in* $[1, t)$*.*

Proof. By Lemma 1, 2PA a valid solution in time $O(N(1 + \min\{\log N, \log\log t\}))$. Let V be the total profit of this solution and A be an optimal solution. By Lemma 3, $V \geq V(\mathbf{S})$, and by Lemma 2, $2V \geq V(\mathbf{S}) + V(\mathbf{S}_A) \geq P(A)$. ❑

3 ISP and Throughput Maximization

In this section, we show how to use the ISP to provide efficient solutions for the Throughput Maximization problems for both the pseudo-polynomial and the strongly-polynomial case.

3.1 Pseudo-polynomial Algorithms

We use the same notations as defined in Section 1.

Theorem 5 *There is an pseudo-polynomial algorithm for TMP with an approximation ratio of* 2 *that runs in* $O(tn \log\log t)$ *time.*

Proof. Given a TMP instance I, we can define an ISP instance $T_1(I)$by having the same profit coefficients w_i, and the families of integer intervals indicating the time intervals during which a job may be executed:

$$S_i = \{[s_i, s_i + l_i) : \; r_i \leq s_i \text{ and } s_i + l_i \leq d_i\}$$

Notice that there are at most $N = tn$ intervals in our collection and that $\log\log t \leq \log\log N$. In turn, a solution A of $T_1(I)$ yields the schedule $\{(i, s) : \; (i, [s, s+l_i)) \in A\}$. This translation, in conjunction with 2PA from Figure 2, proves the theorem. ❑

Notice that the algorithm in Theorem 5 is pseudo-polynomial since N, the total number of intervals in $T(I)$, is not a polynomial function of n, but rather of the largest coefficient t of the input. Still, this algorithm achieves the same approximation ratio as the pseudo-polynomial algorithm of Bar-Noy *et al.* [2], which starts from solving a linear program with at least N variables followed up by a costly preprocessing, and thus takes at least $\Omega(N^2)$ time.

In the special case when the profits of all the jobs $J_1, J_2, \ldots, J_n$ are identical, 2PA as applied above in Theorem 5 reduces to the algorithm 1-GREEDY of [2], and hence the tightness of the performance ratio of in Theorem 5 follows from that of 1-GREEDY as described in [2]. One may be tempted to try to improve the performance ratio by running it twice, once in left-to-right order of endings and another time in right-to-left order of endings, and take the better of the two solutions. The following example illustrates that the performance ratio will still be 2 in the worst case. There are $2n-1$ jobs $J_1, J_2, \ldots, J_n$ and $J'_1, J'_2, \ldots, J'_{n-1}$, all with profit 1. Let the notation (p, q, r) denote a job with release time p, deadline q and length r and let $0 < \varepsilon < 1$ be any arbitrary constant. Then, $J_i = (2i-1, 2i, 1)$ and $J'_i = (2i-1-\varepsilon, 2i+2+\varepsilon, 1)$. The optimal schedule schedules all the $2n-1$ jobs whereas both runs of 2PA will only schedule $n-1$ jobs $J'_1, J'_2, \ldots, J'_{n-1}$. Note that maximum stretch factor in the above example is slightly more than 3.

Theorem 6 *There is a pseudo-polynomial algorithm for TMP$_k$-un with an approximation ratio of* 2 *which runs in* $O(tnk \log\log(tk))$ *time.*

Proof. Consider an instance I of TMP$_k$-un problem. We can define the corresponding instance $T_2(I)$ of ISP by setting

$$S_i = \{[s_i, s_i + l_{i,m}) + (m-1)t : \ m \in [1, k],\ r_i \le s_i \text{ and } r_i + l_{i,m} \le d_i\}$$

where $[d, e) + c$ denotes $[d + c, e + c)$. Notice that we have at most $N = tnk$ intervals in our collection, each contained in $[1, tk)$. Now, a solution A of $T_2(I)$ yields the schedule $\{(i, m, s) : \ s \le t \text{ and } (i, [s, s + l_{i,m}) + (m-1)t) \in A\}$. ❑

Notice that the algorithm in Theorem 6 has a better performance ratio then the ratio 3 algorithm of Bar-Noy *et al.* [2]. Next, we consider the case when all the $k > 1$ machines are identical.

Theorem 7 *There is a pseudo-polynomial algorithm for TMP$_k$-id with an approximation ratio of* $\frac{(k+1)^k}{(k+1)^k - k^k}$ *which runs in* $O(tnk \log\log t)$ *time.*

Proof. Let $T_1(I)$ be as described in Theorem 5. We solve an instance I of TMP$_k$-id by running k iterations of 2PA:

$J \leftarrow T_1(I)$
for *(each integer* $m \in [1, k]$ *)*
{ *create solution* A *by running 2PA on* J*;*
  for *(each* $(i, [d, e)) \in A$ *)*
    insert (i, m, d) *to the solution,*
    remove S_i *and* w_i *from* J*;*
}

Correctness of the algorithm is obvious, as in each iteration 2PA assures that the jobs assigned to a particular machine are executed during disjoint time intervals. By removing the interval families of the jobs already scheduled, we also assure that each job is scheduled at most once. The time taken is also clearly $O(tnk \log\log t)$, since the selected jobs and their profits can be removed in $O(tn)$ time after each iteration of 2PA .

Let us rescale the profits of the jobs so that the profit of the optimum solution is 1. Let P_m be the profit of the jobs scheduled in the fist m iterations of 2PA. Our goal is to show that

$$\frac{1}{P_k} \leq \frac{(k+1)^k}{(k+1)^k - k^k} \quad \equiv \quad P_k \geq 1 - \left(\frac{k}{k+1}\right)^k \quad \equiv \quad 1 - P_k \leq \left(\frac{k}{k+1}\right)^k$$

Obviously, $P_0 = 0$, so $1 - P_0 = 1$. Thus it suffices to show that for $j \in [1, k]$

$$1 - P_j \leq \frac{k}{k+1}(1 - P_{j-1}) \quad \equiv \quad P_j - P_{j-1} \geq \frac{1}{k+1}(1 - P_{j-1})$$

The left-hand side of the last inequality is the sum of profits of the jobs scheduled in the j^{th} iteration. Expression $(1 - P_{j-1})$ on the right-hand side is a lower bound on the optimum profit that can be interpreted as follows: the optimum profit for the initial input J is 1, if we delete from this schedule all jobs that were scheduled in the previous $j - 1$ iterations (and thus were deleted from J), the remaining profit is at least $1 - P_{j-1}$. Therefore it suffices to show that in an iteration we produce a schedule for one machine that has profit at most $k + 1$ times smaller than the optimum schedule for k machines.

Consider an instance I of TMP$_k$-id and an optimum schedule B with profit $V(B)$. Then we can form k solutions for $T_1(I)$ that correspond to schedules for each machine: $A(m) = \{(i, [s, s + l_i]) : \ (i, m, s) \in B\}$. Recall Lemma 2 and its notation. Because the sets of jobs scheduled on various machines are disjoint, we have $\sum_{m=0}^{k-1} V(\mathbf{S}_{A(m)}) \leq V(\mathbf{S})$. The claim of Lemma 2 states that $V(\mathbf{S}) + V(\mathbf{S}_{A(m)}) \geq V(A(m))$. By adding all such inequalities we get

$$(k+1)V(\mathbf{S}) \geq kV(\mathbf{S}) + \sum_{m=0}^{k-1} V(\mathbf{S}_{A(m)}) \geq \sum_{m=0}^{k-1} V(A(m)) = V(B).$$

By Lemma 3, the profit obtained by this run of 2PA is at least $V(\mathbf{S})$, which in turn is at least $V(B)/(k + 1)$. ❑

Notice that the algorithm in Theorem 7 has the same performance ratio as in Bar-Noy *et al.* [2], but the algorithm in Bar-Noy *et al.* [2] takes at least $\Omega(t^2 n^2 k)$ time in the worst case.

3.2 Strongly polynomial algorithms

The algorithms for TMP, TMP$_k$-id and TMP$_k$-un presented in the previous section are pseudo-polynomial, because all of them start from forming an instance of ISP with N intervals, where N is pseudo-polynomial. However, the families of intervals in an ISP instance that is formed have a very regular definition, and therefore 2PA can be accelerated with approximation ratios that are arbitrarily close to the ratios of the pseudo-polynomial algorithms of the previous section. In particular, we have the following theorem where $0 < \varepsilon < 1$

is any value. Notice all the ratios and running times are better than the corresponding ones in Bar-Noy *et al.* [2].

Theorem 8 *There is an approximation algorithm for the TMP (respectively, TMP_k-id, TMP_k-un) problem with approximation ratio $2/(1-\varepsilon)$ (respectively, $\frac{(k+1)^k}{(k+1)^k-(k+\varepsilon)^k}$, $2/(1-\varepsilon)$) which runs in $O(n^2/\varepsilon)$ (respectively, $O(kn^2/\varepsilon)$, $O(n^2/\varepsilon)$) time.*

Proof. We use the notations in Section 2. As the first step, we will modify 2PA so that the number of stack operations will be proportional to n, the number of families of intervals (and the number of jobs in the original scheduling problems). In particular, we will change the condition for pushing an interval from

$$\textbf{if } (\ v > 0\ )\ \text{push}((i, v, d, e), \mathbf{S});$$

to

$$\textbf{if } (\ v > \varepsilon w_i\ )\ \text{push}((i, v, d, e), \mathbf{S});$$

We will call the modified algorithm ε-2PA.

Observe that the size of stack $\mathbf{S}$ at the end of the evaluation phase of ε-2PA is at most $\lfloor \varepsilon^{-1} \rfloor n$: each entry of $\mathbf{S}$ has *value* $> \varepsilon w_{family}$, thus when we evaluate some $(i, w_i, [d, e])$ and $\mathbf{S}$ contains j entries with *family* $= i$, then the computed v is smaller than $(1 - j\varepsilon)w_i$, and if $j = \lfloor \varepsilon^{-1} \rfloor n$ then

$$v < (1 - j\varepsilon)w_i < (1 - (\varepsilon^{-1} - 1)\varepsilon)w_i = \varepsilon w_i$$

and thus v is too small to push a new entry onto $\mathbf{S}$. We can conclude that the running time of the selection phase of ε-2PA is $O(n/\varepsilon)$. Now, we show that if I is an instance of TMP with n jobs, we can implement the evaluation phase of ε-2PA to run in time $O(n^2/\varepsilon)$ on input $T_1(I)$.

Notice that we need to design a data structure such that every time we need to find a new entry to push to the stack $\mathbf{S}$, we will find it in $O(n)$ time. To design our data structure, we need to inspect how the value of v is computed for $(i, w_i, e - l_i, e)$. We can compute v by starting with the expression $w_i - V(\mathbf{S})$, and then adding the value of all *intervals* on $\mathbf{S}$ that have *family* $\neq i$ and *ending* $\leq e - l_i$. When we add a new entry to $\mathbf{S}$, we need to update $V(\mathbf{S})$.

Let $q = \lfloor \varepsilon^{-1} \rfloor n$ and assume that the entries of $\mathbf{S}$ occupy array positions $S[1]$ to $S[q]$, with top($\mathbf{S}$)$\leq q$ denoting the top of $\mathbf{S}$ at any moment. Let V denote the sum of all values in $\mathbf{S}$ at any moment. We also maintain, for each J_i, an array left$[i, 1:q]$ of size q such that left$[i, j]$ stores the sum of values of all intervals that have *family* $\neq i$ and *ending* $\leq S[j].ending$, and a pointer $r[i]$ to the most recent entry visited in left$[i, 1:q]$ (left$[i, r[i]]$$= \infty$ will indicate that no further interval of J_i is available for further consideration). The array element end$[i]$ stores, for each job J_i, the earliest ending that gives a positive value (end$[i]$$= \infty$ indicates that no further instance of J_i can be under consideration). Assume for convenience that left$[i, 0]$ is 0 for all i and $S[0].ending$ is 1. We initialize $V \leftarrow 0$, top($\mathbf{S}$)$\leftarrow 0$ and $r[i] \leftarrow 0$ for all i. Now, we find an entry with a positive value of v and earliest ending in the following way:

for (each integer $i \in [1, n]$)
{ end$[i] \leftarrow \infty$; $p \leftarrow r[i]$

while ($p \leq$ top($\mathbf{S}$) **and** left$[i,p] < V-(1-\varepsilon)w_i$) $p \leftarrow p+1$;
if (S[p].ending$> d_i$ **or** $p >$top($\mathbf{S}$)) left$[i,p] \leftarrow \infty$
else { val$[i] \leftarrow$ left$[i,p]-(V-w_i)$; end[i]$\leftarrow$ s[p].ending$+l_i$; $r[i] \leftarrow p$ }
}

let end$[j]$ be $\min_{1\leq i\leq n}\{$end$[i]\}$

if (end$[j]=\infty$) algorithm terminates
else the entry $(j, \text{val}[j], S[r[j]].\text{ending}, S[r[j]].\text{ending}+l_j)$ should be pushed to $\mathbf{S}$

and while pushing the entry $(j, \text{val}[j], S[r[j]].\text{ending}, S[r[j]].\text{ending}+l_j)$ to $\mathbf{S}$ we perform the following updates:

push $(j, \text{val}[j], S[r[j]].\text{ending}, S[r[j]].\text{ending}+l_j)$ to $\mathbf{S}$
$V \leftarrow V+\text{val}[j]$
for (each integer $i \in [1,n]$)
if ($i \neq j$) left[i,top($\mathbf{S}$)]$\leftarrow$ left[i,top($\mathbf{S}$)-1]+val[j]
else left[i,top($\mathbf{S}$)]$\leftarrow$ left[i,top($\mathbf{S}$)-1]

Each of the $O(n/\varepsilon)$ push takes $O(n)$ time. Each of the $O(n/\varepsilon)$ minimum computation also takes $O(n)$ time. Moreover, since the values of $r[i]$ are increasing and at most $O(n/\varepsilon)$, it is easy to see that the total time for all the remaining operations is also $O(n^2/\varepsilon)$.

Now it remains to find the approximation ratios that result from applying ε-2PA. The analysis that we have performed for 2PA can be largely repeated. In particular, Lemma 3 still applies, so it remains to characterize $V(\mathbf{S})$ at the end of the evaluation phase. By inspecting the proof of Lemma 2 one can see that the lower estimate of $V(\mathbf{S}_{d,e})+V(S_i)$ must be decreased by v in the case when the evaluation phase computes v to be positive, and yet it does not push (i,v,d,e) onto $\mathbf{S}$; because in this case we have $v \leq \varepsilon w_i$, we decrease the estimate of Lemma 2 to

$$V(\mathbf{S})+V(\mathbf{S}_A) \geq (1-\varepsilon)V(A)$$

As a result, if we have an instance I of TMP and apply ε-2PA to $T_1(I)$, then we obtain approximation ratio $2/(1-\varepsilon)$ and the running time of n^2/ε. Similarly, if I was an instance of TMP$_k$-id, and x is the profit of an optimum schedule on k machines, then a run of ε-2PA on $T_1(I)$ returns a schedule for one machine with profit at least $x(1-\varepsilon)/(k+1)$, and by iterating ε-2PA for k times we get a ratio of

$$\frac{(k+1)^k}{(k+1)^k-(k+\varepsilon)^k}.$$

in time $O(kn^2/\varepsilon)$.

A similar modification can be applied to the algorithm for TMP$_k$-un to run in $O(n^2/\varepsilon + kn) = O(n^2/\varepsilon)$ time with a performance ratio of $2/(1-\varepsilon)$. ❑

4 Scheduling with Bounded Stretch Factor

In this section, we consider TMP restricted to cases when $d_i - r_i \le \alpha l_i$ for some fixed upper bound of the stretch factor $\alpha \ge 1$. For $\alpha \le 2$, the problem is solvable in polynomial time [2]. These cases are interesting because of their applications to adaptive rate-controlled scheduling [10, 8, 12] as discussed in the introduction and, in fact, we can obtain approximation ratios below 2 using a variation of algorithm 2PA. In particular, we can prove the following theorem.

Theorem 9 *Assume that the stretch factor of each job is at most α and let $a = \lfloor \alpha \rfloor$. Then, there is a pseudo-polynomial time algorithm for the TMP problem which runs in $O(atn \log \log t)$ time with performance ratio*

$$\frac{2}{1 + \frac{1}{2^{a+1} - 2 - a}}$$

Proof. We will use a modified version of Lemma 2. In particular, consider a feasible solution A to an ISP instance, and the state of stack $\mathbf{S}$ at the end of the evaluation phase of 2PA, and the notation of Lemma 2. Then for $(i, [d, e)) \in A$ we redefine $\mathbf{S}_i$ to be the set of entries of $\mathbf{S}$ with *family* $= i$ and *ending* $\le d$ (rather than all entries of $\mathbf{S}$ with *family* $= i$). After this change, Lemma 2 still holds, and the proof remains virtually unchanged. We will refer to this new statement as Lemma 2'.

We also need the following notation. If (i, s) belongs to an optimum schedule A, then we say that job J_i is scheduled optimally with parameter $\tau_i = (s - r_i)/l_i$. Moreover, we will view a stack entry $(i, [d, e))$ as an attempt to schedule job J_i with parameter $\beta = (d - r_i)/l_i$. Clearly, $\beta \le \alpha - 1$. One can see that in Lemma 2' the set $\mathbf{S}_i$ consist of attempts to schedule job J_i with parameter $\beta \le \tau_i - 1$.

Before we proceed further, let us observe that if $\lfloor \alpha \rfloor = 1$, then $\mathbf{S}_A$ must be empty, and thus algorithm 2PA guarantees the optimum profit, or the fraction of optimum profit equal to $(1 + (2^{1+1} - 2 - 1)^{-1})/2$.

Our idea is simple. By Lemma 2', in the worst case we obtain the total profit that is equal to the optimum profit, minus the values of the attempts to schedule jobs with parameters that are smaller by at least 1 than the optimal ones. Suppose that we obtained exactly half of the optimum profit. This would mean that all our attempts to schedule a job had parameters that were by at least 1 too low. We could try to run our algorithm again, but if in the current run we had an attempt to schedule job J_i with parameter γ, in the next run we will prohibit scheduling J_i with parameter lower than $\gamma + 1$.

To make this idea precise, we need to consider the fact that algorithm 2PA allows to make partial attempts, i.e. attempts with values lower than the profit of the respective job. Thus a partial attempt must be followed with a partial prohibition. More concretely, when in a run of 2PA we calculate a value of a possible attempt to schedule J_i with parameter γ, we subtract the value of attempts (i.e. stack entries) with overlapping time intervals (TOTAL(d) in Figure 2), the values of attempts to schedule J_i with parameter $\beta \le \gamma - 1$ (total(i, d) in Figure 2), and, to express our partial prohibition, the values of attempts to schedule J_i that were made in the previous runs with parameter $\beta > \gamma - 1$.

With the above proviso, we run 2PA a times, and then we make separate a runs after reversing the direction of the time. The latter means that instead of considering all possible scheduling attempts in the order of increasing terminations, we consider them in the order of decreasing starts, and the notions of TOTAL, total and prohibition are changed symmetrically. After we are done with all $2a$ runs, we choose the best of their solutions.

To make the proof of the improved performance easier, we rescale all the profits so that $V(A) = 1$. In our analysis, an attempt to schedule J_i with parameter β is *correct* if $\beta = \tau_i$, and *wrong*, if $\beta \leq \tau_i - 1$.

To analyze this new algorithm, we first show that the sum of $V(\mathbf{S}_A)$ (or, equivalently, the sum of values of all wrong scheduling attempts) over all $2a$ runs is at most $(a-1)$.

Consider a job J_i that is scheduled optimally with parameter τ_i. It suffices to show that the sum of $V(\mathbf{S}_i)$'s over the first a runs of 2PA is at most $\lfloor \tau_i \rfloor w_i$, and that the similar sum over the a runs with reversed time is at most $\lfloor \alpha - \tau_i - 1 \rfloor w_i$. Intuitively, the worst we can do in the first a runs is to make attempts with parameters $0, 1, \ldots, \lfloor \tau_i - 1 \rfloor$, each with value w_i; similarly, in the runs in the reverse direction, at worst we have attempts to schedule J_i with parameters $\alpha - 1, \alpha - 2, \ldots, \alpha - \lfloor \alpha - \tau_i - 1 \rfloor$.

Let us assume that the sum of $V(\mathbf{S}_A)$ over the first a runs is at most $(a-1)/2$ (if not, then it is true for the second set of a runs). We define $\delta = (2^{a+1} - 2 - a)^{-1}$. We will prove that in one of these a runs the sum of values of the scheduling attempts (and hence, the obtained profit) is at least $(1+\delta)/2$. To do it, we will obtain a contradiction with the assumption that in each of the a runs this sum is below $(1+\delta)/2$. We start by showing that under the latter assumption the following holds for $b = 0, \ldots, a$:

> after b runs we have prohibited correct attempts with joint value at most $(2^b - 1)\delta$ and, if $b > 0$, in the b^{th} run we have $V(\mathbf{S}_A) > (1 - (2^b - 1)\delta)/2$.

It is clear for $b = 0$. If we assume this claim for $b - 1$, then we start the run number b with non-prohibited correct attempts with joint value at least $1 - (2^{b-1} - 1)\delta$. Because in this run the joint value of all attempts is smaller than $(1+\delta)/2$, we have

$$V(\mathbf{S}_A) > 1 - (2^{b-1} - 1)\delta - \frac{1+\delta}{2} = \frac{1 - (2^b - 1)\delta}{2}$$

and thus the value of the correct attempts (that will be prohibited later) is smaller than

$$\frac{1+\delta}{2} - \frac{1 - (2^b - 1)\delta}{2} = 2^{b-1}\delta$$

Because before the b^{th} run we prohibited correct attempts of value at most $(2^{b-1} - 1)\delta$, the new total is below $(2^b - 1)\delta$.

After proving our inductive claim, we can derive a contradiction: the sum of $V(\mathbf{S}_A)$ over the first a runs is larger than

$$\sum_{b=1}^{a} \frac{1 - (2^b - 1)\delta}{2} = \frac{a}{2} - \frac{\delta}{2} \sum_{b=1}^{a} (2^b - 1) = \frac{a - \delta(2^{a+1} - 2 - a)}{2} = \frac{a-1}{2}$$

while we assumed that this sum is at most $(a-1)/2$. ❑

Using ε-2PA instead of 2PA in a manner similar to that in the proof of Theorem 8, we can also devise a strongly polynomial algorithm for this case whose performance ratio is arbitrarily close to that in Theorem 9.

5 Conclusion and open problems

We have shown simple combinatorial algorithms that in some cases match, and in other cases exceed the performance of LP based algorithms of Bar-Noy *et al.* [2]. Our algorithms can be viewed as a proper extension of the greedy algorithms that can be used when every job has the same profit.

A major open problem is to bring the approximation ratio for TMP below 2. One can also point out several problems of more modest scope. Can we improve the running time of our algorithms, mainly 2PA and ε-2PA? Is the performance ratio proven for the algorithms for bounded stretch factor optimal?

6 Acknowledgments

The authors would like to thank Sefi Naor and Baruch Schieber for their useful discussion and explanations, Amos Fiat and Gerhard Woeginger for organizing Dagstuhl workshop on online algorithms where some of these discussions took place, Michael A. Palis for pointing out applications of stretch factors to adaptive rate-controlled scheduling, as well as NSF and NLM for providing the financial support for this research.

References

[1] Barauh S., G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha and F. Wang, *On the competitiveness of on-line real-time scheduling*, Real-Time Systems **4**, 125-144, 1992.

[2] Bar-Noy, A., S. Guha, J. (S.) Naor and B. Schieber, *Approximating the throughput of multiple machines in real-time scheduling*, Proc. 31st ACM STOC, 622-631, 1999. Full version available at Prof. Amotz Bar-Noy's web-site `http://www.eng.tau.ac.il/~amotz/`.

[3] Berman, P., Z. Zhang, J. Bouck and W. Miller, *Large aligning two fragmented sequences*, submitted for RECOMB'00.

[4] Kise H., T. Ibarakiand H. Mine, *A solvable case of one machine scheduling problems with ready and due dates*, Operations Research **26**, 121-126, 1978.

[5] Koren G. and D. Shasha, *An optimal on-line scheduling algorithm for overloaded real-time systems*, SIAM J. on Computing **24**, 318-339, 1995.

[6] Lawler, E. L., *A dynamic programming approach for preemptive scheduling of a single machine to minimize the number of late jobs*, Annals of Operations Research **26**, 125-133, 1990.

[7] Lipton R. J. and A. Tomkins, *Online interval scheduling*, Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms, 302-311, 1994.

[8] Liu H. and M. E. Zarki, *Adaptive source rate control for real-time wireless video transmission*, Mobile Networks and Applications **3**, 49-60, 1998.

[9] Overmars, M. H., *Computational geometry on a grid: an overview*, Theoretical Foundations of Computer Graphics and CAD, NATO ASI Series **F40**, Edited by R. A. Earnshaw, Springer-Verlag Berlin Heidelberg, 167-184, 1988.

[10] Rajugopal, G. R. and R. H. M. Hafez, *Adaptive rate controlled, robust video communication over packet wireless networks*, Mobile Networks and Applications **3**, 33-47, 1998.

[11] Sahni, S, *Algorithms for scheduling independent tasks*, JACM **23**, 116-127, 1976.

[12] Yau, D. K. Y. and S. S. Lam, *Adaptive rate-controlled scheduling for multimedia applications*, Proc. IS&T/SPIE Multimedia Computing and Networking Conf., San Jose, CA, January 1996.