We update:
\[ z_t \leftarrow z_t + \delta; \quad y_i \leftarrow y_i + \delta; \]
and push onto the stack all the intervals in I whose dual constraint becomes satisfied as a result. The algorithm iterates until all the dual constraints become satisfied (and thus all the intervals are in the stack), at which time the dual variables constitute a feasible dual solution. We now pop each interval in turn off the stack and add it to the schedule if doing so does not violate the feasibility of the schedule.

Observe that a given point in time \( t \) may serve as the timeline during several successive iterations, and that if \( t \) is a timeline then all the intervals whose right endpoint is less than \( t \) are pushed into the stack prior to the first iteration in which \( t \) is the timeline.

**Lemma 7.1** Upon termination, for every interval \( I \) in the final schedule, \( \sum_{i \in T_I} z_i \leq p(I) \).

**Proof:** Assume by contradiction that \( I \in R \) is an interval in the final schedule for which \( \sum_{i \in T_I} z_i > p(I) \). Consider the iteration in which \( I \) was pushed onto the stack. Let \( t_1 \) be the timeline during this iteration and let \( R \) be the activity to which the interval determining this timeline belongs. Let \( Z \) be the value of \( \sum_{i \in T_I} z_i \) at the beginning of the iteration and let \( Z' \) be the value of this sum at the end of the iteration. Let \( y \) denote the value of \( y_i \) at the beginning of the iteration. The fact that \( I \) is pushed onto the stack in this iteration indicates that it either contains \( t_1 \) or else it belongs to activity \( R \). In the former case \( Z' = Z + \delta \), where \( \delta \leq p(I) - y - Z \leq p(I) - Z \), and in the latter, \( Z' = Z < p(I) - y \leq p(I) \) (where the first inequality is implied by the fact the constraint corresponding to \( I \) is unsatisfied at the beginning of the iteration). Either way, \( Z' \leq p(I) \).

Hence, there exists a timeline \( t_2 \in T_I, t_2 > t_1 \), such that \( z_i > 0 \). Let \( I' \) be the activity that determined the timeline \( t_2 \). Since \( I' \) has a common point with \( I \) it cannot be in the schedule. The reason \( I \) is not in the schedule is that when it was popped off the stack, some other interval intersecting it was already in the schedule. This interval must have been pushed on the stack after \( I' \) (and certainly after \( I \)), and thus it must contain \( t_2 \) in order to intersect \( I' \). Hence, it intersects \( I \) as well, so \( I \) cannot be in the schedule. \( \square \)

**Lemma 7.2** For all \( t \in T \) for which \( z_t > 0 \) upon termination, there exists an interval \( I \) in the final schedule such that \( t \in T_I \).

**Proof:** The fact that \( z_t > 0 \) implies that \( t \) is a timeline and hence a right endpoint of some interval \( I \) which determined it. If none of the intervals containing \( t \) where added to the schedule by the time \( I \) was popped off the stack, then \( I \) was added to the schedule. \( \square \)

**Lemma 7.3** Upon termination, \( \sum_{i \in T} z_i \geq \sum_{i=1}^n y_i \).

**Proof:** During the steps of the algorithm, whenever a variable \( y_i \) is incremented, another variable \( z_i \) is also incremented by the same amount. \( \square \)

**Theorem 7.4** The approximation factor achieved by the primal-dual algorithm is at least \( 1/2 \).

**Proof:** Denote the set of activity intervals in the final schedule by \( S \). Then,
\[
\sum_{I \in S} p(I) \geq \sum_{I \in S} \sum_{i \in T_I} z_i \geq \sum_{i \in T} z_i \geq \frac{1}{2} \left( \sum_{i=1}^n y_i + \sum_{i \in T} z_i \right),
\]
where the inequalities follow in order from Lemmas 7.1, 7.2, and 7.3. \( \square \)
Appendix: A Primal-Dual Interpretation

Recall the integer programming formulation of the problem from Section 2; for the interval scheduling problem $\text{Width}(t)$ is identically 1 and $w(I) = 1$ for all $I$. Let us construct the dual program of the linear relaxation of this program. We define $y_i$ as the dual variable corresponding to the constraint on activity $R_i$ and $z_t$ as the dual variable corresponding to the constraint at time $t$. Let $T$ be the set of all start and end times of all instances belonging to all the activities. Denote the subset of $T$ which is contained in activity instance $I$ by $T_I$.

Minimize $\sum_{i=1}^{n} y_i + \sum_{t \in T} z_t$ subject to:

For each activity instance $I \in R_i$: $y_i + \sum_{t \in T_I} z_t \geq p(I)$.

For all $i$ and $t$: $y_i, z_t \geq 0$.

Let us reformulate the unified algorithm described in Section 3.1 as a primal-dual algorithm. We explain how to update the dual variables. Initially, all dual variables are set to zero. For interval $I \in R_i$, the corresponding dual constraint is called satisfied if $y_i + \sum_{t \in T_I} z_t \geq p(I)$, and unsatisfied otherwise. Thus all the constraints are initially unsatisfied. At each iteration the algorithm selects an interval $I \in R_i$ whose end time is earliest among all intervals whose dual constraints are unsatisfied. Let $t$ be the right endpoint of interval $I$. We say that $t$ is the timeline during the this iteration and that $I$ is the interval determining the timeline. Let $I$ be the set of intervals that either contain $t$ or belong to $R_i$ or correspond to some unsatisfied constraint. Let

$$\delta = \min_{I \in T} \left\{ p(I') - y_i - \sum_{t \in T_I} z_t \right\}.$$
Suppose that the optimum for the ring topology is \( x + y \) where \( x \) is the contribution of \( I' \) and \( y \) is the contribution of \( I'' \). Then the value of the solution we return is at least \( \max \{ \frac{x}{\rho}, y/(1 + \varepsilon) \} \).

It is not difficult to verify that regardless of whether \( x/\rho \geq y/(1 + \varepsilon) \) or \( y/(1 + \varepsilon) > x/\rho \), it is always true that \( \rho + 1 + \varepsilon \geq (x + y)/\max \left\{ \frac{x}{\rho}, \frac{y}{1 + \varepsilon} \right\} \).

7 A Primal-Dual Interpretation

Our approximation algorithms can be reformulated in the primal-dual schema. We demonstrate this on the interval scheduling problem. For lack of space, details are given in the Appendix.

Acknowledgement

We thank Sanjeev Khanna for many fruitful discussions.

References


size at most $6W^*$, where $W^*$ is the maximum over all times $t$ of the sum of the width demands at time $t$. We use Kierstead’s result to achieve an approximation ratio of $1/35$ for the throughput maximization version of DSA. In fact, we solve a more general problem in which a single request for storage may offer several alternative time intervals, only one of which is to be selected. The duration, storage demand, and profit may all vary within different intervals pertaining to the same request. This is just the bandwidth allocation problem with a contiguity requirement thrown in.

Let $W$ be the size of available storage. We partition the instances into wide instances, whose widths are more than $W/2$, and narrow instances, whose widths are at most $W/2$. We solve the bandwidth allocation problem separately for the wide instances and for the narrow instances: we obtain a $1/2$-approximation for the wide instances via interval scheduling, and a $1/3$-approximation for the narrow instances as we have done for the bandwidth allocation problem. Let $S_w$ be the solution for the wide instances and $S_n$ the solution for the narrow instances. Observe that $S_n$ is feasible for DSA since it is an independent set. We proceed to extract from $S_n$ a feasible solution for DSA, which we denote by $S'_n$, and return the solution with higher profit among $S_w$ and $S'_n$.

We derive $S'_n$ from $S_n$ as follows. Given the instances in $S_n$, we run Kierstead’s algorithm [15] and find a DSA solution consisting of a subset of $S_n$ that fits in a storage block of size $6W$. We cover this storage block by 11 overlapping strips, each of width $W$: $[0..W]$, $[W/2..3W/2]$, $[W..2W]$, $[3W/2..5W/2]$, $[5W/2..11W/2]$, $[5W..6W]$. Since the instances are all narrow, each is covered completely by one (or more) of these strips. Each strip defines a feasible solution for our problem—the set of instances covered completely by the strip. We select $S'_n$ as a solution with maximum profit among these solutions.

Since the total profit of these solutions is equal to the profit of $S_n$, which is a $1/3$-approximation for bandwidth allocation, the profit of $S'_n$ is a $1/33$-approximation for bandwidth allocation. It is also a $1/33$-approximation to DSA since the optimum profit for DSA can only be higher than the optimum for bandwidth allocation. By an argument similar to the one employed in the solution of the bandwidth allocation problem, the better solution of $S_n$ and $S'_n$ is a $1/35$-approximation.

### 6 Scheduling Sessions on Line and Ring Topologies

As mentioned in the Introduction, all of our results regarding the time axis interpretation hold for the processors axis interpretation as well. They hold for both throughput maximization and loss minimization. They also hold for the contiguous case. Moreover, since processors are discrete there are no issues of continuous “windows” to contend with (although it is hard to think of any “real” application involving activities with multiple instances).

In this section we show how to adapt any approximation result for the throughput maximization problem on the line into one for the same problem on the ring. If the approximation factor for the line is $1/\rho$, the resulting factor for the ring will be $1/(\rho + 1 + \varepsilon)$ where $\varepsilon$ may be chosen arbitrarily small (the running time of the algorithm depends polynomially on $1/\varepsilon$).

The idea is to cut the ring at an arbitrary point and to partition the set of instances (of all the activities) into those that do not pass through the cut and those that do. We denote the former by $I'$ and the latter by $I''$. The instances in $I'$ define an equivalent problem on the line, whereas the instances in $I''$ amount to a knapsack problem. It is well known that the knapsack problem admits a FPTAS. We obtain a $1/\rho$-approximation for $I'$ and a $1/(1+\varepsilon)$-approximation for $I''$, and return the better solution of the two.
Thus, since \( p(I) \leq \Delta^* \) for all \( I \), and in particular for \( I' \),

\[
\sum_{I \in \mathcal{S}(t)} p(I) = p(I') + \sum_{I \in \mathcal{S}(t) \setminus \{I'\}} p(I) < 2\Delta^*.
\]

Thus, if \( t^* \) is an unstable point, then the \( p_1 \)-cost of the schedule is \( \sum_{I \in \mathcal{S}(t^*)} p(I) < 2\Delta^* \) and we are done. Otherwise, let \( t_L \) be the latest unstable point in time earlier than \( t^* \) and let \( t_R \) be the earliest unstable point in time later than \( t^* \) (it may be that only one of them exists). The schedule’s maximality implies that every interval in \( \mathcal{S}(t^*) \) is the cause of instability of at least one point in time and thus belongs also to \( \mathcal{S}(t_L) \) or \( \mathcal{S}(t_R) \) (or both). Hence, the \( p_1 \)-cost of the schedule satisfies

\[
\sum_{I \in \mathcal{S}(t^*)} p(I) \leq \sum_{I \in \mathcal{S}(t_L)} p(I) + \sum_{I \in \mathcal{S}(t_R)} p(I) < 4\Delta^*.
\]

The time complexity of the algorithm is polynomial in the number of activities and the number of points in time at which \( \text{Width}(t) \) changes value.

### 4.1 Application: General Caching

As was shown in [1], the general caching problem is an instance of the loss minimization problem. Our algorithm thus yields a 4-approximation for this problem. For the sake of completion we show how the general caching problem reduces to the loss minimization problem. This is done as follows. Let \( \text{size}(t) \) be the size of the page requested at time \( t \). The resource width function is \( \text{Width}(t) = W - \text{size}(t) \), where \( W \) is the cache size. Each page request defines an activity as follows. Suppose the request is made at time \( t \) for page \( g \). Let \( t' \) be the last time page \( g \) was requested prior to time \( t \) (or \( t' = 0 \) if this is the first request for the page). Let \( r \) be page \( g \)’s reload cost and let \( z \) be its size. The activity corresponding to this request consists of a single instance \( I: s(I) = t' + 1, e(I) = t + 1, w(I) = z, \) and \( p(I) = r \).

### 5 Contiguous Allocation

Let us describe the generic throughput maximization problem in pictorial terms. We visualize the resource as a strip, perhaps of varying width (in accordance with \( \text{Width}(t) \)), stretching along the \( t \)-axis. Each instance is a rectangle to be embedded in the strip at a certain \( t \) coordinate. The bandwidth constraint says that all the rectangles chosen must fit in the strip together without overlap. It does not, however, prevent us from cutting up a rectangle into strips and embedding them in the resource strip in a non-contiguous manner. In fact, we can slice the rectangle both horizontally and vertically. For example, in the general caching problem we are not obliged to store a page in contiguous memory location (horizontal slicing) nor must we keep it fixed in the same memory locations over time (vertical slicing). The dynamic storage allocation problem (DSA) models situations where contiguity is required both in space and in time. The resource is assumed to consist of linearly ordered units of storage and each object must be stored in contiguous units and occupy the same units for as long as it remains in storage (i.e., it may not be moved around). Pictorially, this means that no cutting of rectangles is allowed.

The traditional goal in DSA has been to store all objects in minimum size memory. This problem is NP-Hard [10]. Kierstead [15] describes an algorithm that uses a block of storage of
total width greater than $W/2$, for otherwise $I$ can be added. Thus, the $p_1$-profit of every maximal solution is at least $\delta \cdot W/2$.

## 4 Loss Minimization

In the loss minimization problem that we consider in this paper, each activity $R_i$ comprises a single instance $I$, and the objective is to find a feasible schedule with minimum cost, where the cost of a schedule is the sum of profits of instances not in the schedule. Strictly speaking, for the problem to fit in the framework of the Local Ratio Theorem, we should define the feasible solutions to be the complements of feasible schedules. However, since the unified algorithm returns feasible schedules, we refer to schedules as solutions but still measure the cost in terms of the complements. We show how to instantiate the unified algorithm so as to obtain a $1/4$-approximation for this problem.

We use the unified algorithm, but with a different decomposition of the profit function. For a given point in time $t$, let $I(t)$ be the set intervals containing $t$. Define the shortage at time $t$ as $\Delta(t) = \sum_{I \in I(t)} w(I) - \text{Width}(t)$. To compute the decomposition $p = p_1 + p_2$, find a point in time $t^*$ maximizing the shortage $\Delta(t^*)$ and let $\Delta^* = \Delta(t^*)$. If $\Delta^* \leq 0$, define $p_1 = p$ (which is equivalent to simply returning $A$ as the solution). Otherwise define

$$p_1(I) = \begin{cases} \min\{\Delta^*, w(I)\} & I \in I(t^*), \\ 0 & \text{otherwise}. \end{cases}$$

Note that if $\Delta^* \leq 0$, the solution returned is the unique optimal solution and it is maximal with respect to containment. Thus, we need only show that when $\Delta^* > 0$ every maximal solution is a $1/4$-approximation for $(A, p_1)$.

Suppose $\Delta^* > 0$ and consider some feasible solution. Let $S(t)$ be the set of intervals that contain the point in time $t$, and let $\overline{S}(t) = I(t) - S(t)$ be the set of intervals that contain $t$ and are not part of the schedule. The intervals that contribute to the schedule's $p_1$-cost are precisely those in $\overline{S}(t^*)$, and the sum of the widths of these intervals is at least $\Delta^*$ (otherwise the constraint at $t^*$ would be violated). If the width of one of these intervals is $\Delta^*$ or more, then its $p_1$-cost is $\Delta^*$. Otherwise, $p_1(I) = w(I)$ for all $I \in \overline{S}(t^*)$. In either case $\sum_{I \in \overline{S}(t^*)} p_1(I) \geq \Delta^*$.

Hence, the $p_1$-cost of any feasible solution is at least $\Delta^*$.

It remains to show that the $p_1$-cost of every maximal solution is at most $4\Delta^*$. Consider a maximal solution and define $S(t)$ and $\overline{S}(t)$ with respect to it. We say that a point in time $t$ is unstable if there is an interval $I \in \overline{S}(t)$ such that adding $I$ to the schedule would violate the constraint at $t$. We say that $t$ is unstable because of $I$. A point in time may be unstable because of more than one interval and a single interval may be the cause of instability of more than one point in time.

**Lemma 4.1** If $t$ is an unstable point in time, then $\sum_{I \in \overline{S}(t)} p_1(I) < 2\Delta^*$.

**Proof:** Let $I' \in \overline{S}(t)$ be of maximum width. Since $t$ is unstable, it is surely unstable because of $I'$. Thus, $w(I') + \sum_{I \in S(t)} w(I) > \text{Width}(t)$. Since $p_1(I) \leq w(I)$ for all $I$, we have

$$\sum_{I \in \overline{S}(t) - \{I'\}} p_1(I) \leq \sum_{I \in \overline{S}(t) - \{I'\}} w(I) = \sum_{I \in \overline{S}(t)} w(I) - \left( w(I') + \sum_{I \in S(t)} w(I) \right) < \sum_{I \in I(t)} w(I) - \text{Width}(t) = \Delta(t) \leq \Delta^*.$$
single machine considered in [5]. The only difference between this problem and interval scheduling is that the number instances may be infinite. Thus, our algorithm can be used to find a \((1/2 - \varepsilon)\)-approximation.

3.2.2 Parallel Machines and Bandwidth Allocation

**Maximum weight throughput on parallel machines.** This problem is the same as the single machine version, but each job may be scheduled on one of \(k\) identical machines. The parameters are thus \(\text{Width}(t) = k\) for all \(t\) and \(w(I) = 1\) for all \(I\). To obtain a 2-approximation we use \(a = k\) and \(\beta(I) = 1\) for all \(I\). Thus, \(p_1(I) = \delta \cdot k\) for all \(I \in \bar{R}\), and \(p_1(I) = \delta\) for all \(I \in \mathcal{I}\). Any feasible solution may contain at most one instance from \(\bar{R}\) and \(k\) instances from \(\mathcal{I}\), and therefore the optimum \(p_1\)-profit is at most \(2k \cdot \delta\). Every maximal solution that does not contain any instance \(I \in \bar{R}\) must contain \(k\) instances from \(\mathcal{I}\) (otherwise \(I\) can be added). Thus the \(p_1\)-profit of every maximal solution is at least \(k \cdot \delta\).

**Maximum weight \(k\)-colorable subgraph in interval graphs.** This is just the maximum weighted throughput problem on parallel machines where the length of the time window associated with a job is exactly the same as its processing time. This problem can be solved in polynomial time via minimum cost flow [2]. Using our approach, a 1/2-approximation factor is therefore immediate via a much simpler and more efficient algorithm.

**Bandwidth allocation.** We consider a scenario in which the bandwidth of a communication channel must be allocated to calls. Here \(\text{Width}(t) = W\) where \(W\) is the channel’s bandwidth and the activities are calls to be routed through the channel. The calls may be specified in either of two ways: (1) Discrete input that specifies for each call a set of intervals in which it can be scheduled together with a width requirement and a profit. In this case the width and profit may vary among intervals corresponding to the same call. (2) Continuous input that uses a time windows (as in the maximum weight throughput problem), and in addition, each call also specifies a bandwidth requirement, which is the width of all instances of the call.

We first observe that if all instances have width greater than \(W/2\), the problem reduces to maximum weight throughput on a single machine. To solve the problem in the general case, we partition the instances into two sets: narrow instances, whose widths are at most \(W/2\), and wide instances, whose widths are greater than \(W/2\). We find a 1/2-approximation solution for the wide instances via maximum weight throughput on a single machine (or interval scheduling if the input is of the discrete type) and a 1/3-approximation solution for the narrow instances as described below. We then return the solution with higher profit. The solution returned is a 1/5-approximation, since either the optimum value for the narrow instances is at least three fifths of the optimum value for the original problem, or the optimum value for the wide jobs is at least two fifths of the optimum value for the original problem. In either case the solution returned is a 1/5-approximation.

It remains to show how to find a 1/3-approximation for the narrow instances. We use \(a = W/2\) and \(\beta(I) = w(I)\) for all \(I\). Consequently, \(p_1(I) = \delta \cdot W/2\) for all \(I \in \bar{R}\) and \(p_1(I) = \delta \cdot w(I)\) for all \(I \in \mathcal{I}\). Any feasible solution may schedule at most one instance from \(\bar{R}\), the \(p_1\)-profit of which is \(\delta \cdot W/2\), and a subset of \(\mathcal{I}\) whose total width is at most \(W\). Since the \(p_1\)-profit of every instance in \(\mathcal{I}\) is equal to its width times \(\delta\), the \(p_1\)-profit of the solution is at most \(3 \delta \cdot W/2\). On the other hand, if a maximal schedule does not contain any instance of \(\bar{R}\), it must contain a subset of \(\mathcal{I}\) with
where the instances are given explicitly. The running time is polynomial in $1/\varepsilon$ and the number of activities. The idea of the discretization is to make sure that in each iteration of the unified algorithm the $p_1$-profit is at least one over a polynomial times $\varepsilon$ of the maximum profit. We are able to show that guaranteeing this property would deteriorate the approximation factor by only a multiplicative factor of $(1 - \varepsilon)$. We omit the details for lack of space.

### 3.2 Applications

The throughput maximization problem generalizes several known problems. In this section we show the profit function decomposition for a selection of such problems.

#### 3.2.1 Single machine scheduling

Here we assume that the resource is a single machine, i.e., $\text{Width}(t) = 1$ for all $t$, and each activity instance requires the machine ($w(I) = 1$ for all $I$). Note that different instances of the same activity may have different profits.

**Maximum weight independent set in interval graphs.** Consider the special case where each set of requirements $R_i$ is a singleton $\{I_i\}$. This problem is exactly the problem of finding a maximum-weight independent set in an interval graph, where each instance $I_i$ corresponds to an interval. This is a well known problem which can be solved precisely and efficiently in polynomial time [13]. We claim that the unified algorithm with scalers $\alpha = 1$ and $\beta(I) = 1$ for all $I$ yields an optimal solution. To prove this we need only show that every maximal solution is optimal with respect to $p_1$ (in the decomposition $p = p_1 + p_2$). Recall that $T$ is an interval whose end time is earliest and note that $\tilde{R} = \{I\}$ since all activities are singletons. Thus, $\mathcal{T} \cup \tilde{R}$ is the set intervals containing time $e(I)$, and they all have the same $p_1$-profit $\delta > 0$. Furthermore, the $p_1$-profit of all other intervals is 0. Thus, the optimum $p_1$-profit is $\delta$. Since any feasible solution not containing an interval in $\mathcal{T} \cup \tilde{R}$ may be extended by adding one of these intervals, every maximal solution is also optimal.

**Interval scheduling.** In this problem, each activity consists of a finite set of instances (see, e.g., [16, 9, 8]. Our algorithm, with $\alpha = 1$ and $\beta(I) = 1$ for all $I$, finds a 2-approximation, because every maximal solution is a 2-approximation with respect to $p_1$. To prove this, note that once again all the instances in $\tilde{R} \cup \mathcal{T}$ have the same $p_1$-profit $\delta > 0$, and they are the only instances with non-zero $p_1$-profit. This time, however, the set $R \cup \mathcal{T}$ is the union of $\tilde{R}$ and the set of instances containing time $e(I)$. Any feasible solution may contain up to two instances from $\tilde{R} \cup \mathcal{T}$, one from $\tilde{R}$ and one from $\mathcal{T}$, and thus the optimum $p_1$-profit is at most $2\delta$. Every maximal solution, on the other hand, must contain at least one instance from $R_1 \cup \mathcal{T}$ since the constraints in which $\tilde{T}$ appears involve only instances from $\tilde{R} \cup \mathcal{T}$.

**Maximum weight throughput.** Here each activity $R_i$ is defined by a time window $[r_i, d_i]$ and a duration $\ell_i$. The set $R_i$ consists of all time intervals $I$ such that $s(I) \geq r_i$ and $e(I) = s(I) + \ell_i \leq d_i$. All the instances of a given activity have the same profit. Notice that the sets $R_i$ are not given explicitly and each may contain uncountably many instances. This problem is exactly the problem of maximizing the weighted throughput of jobs with release times and deadlines on a
respect to containment will be an $r$-approximation for $(A, p_1)$. The following proof by induction shows that our algorithm then returns an $r$-approximation. Let $S'$ be the schedule obtained recursively for $(A', p_2)$ and let $S$ be the schedule returned for $(A, p)$. By the induction hypotheses $S'$ is an $r$-approximation for $(A', p_2)$. Clearly, both $(A', p_2)$ and $(A, p_2)$ have the same optimum profit since all the activity instances in $A - A'$ have a $p_2$-profit of 0. Thus, $S'$ is an $r$-approximation for $(A, p_2)$ and so is $S$ (since it contains $S'$). In addition, $S$ is maximal with respect to containment, so it is also an $r$-approximation for $(A, p_1)$. Thus, by the Local Ratio Theorem, it is an $r$-approximation for $(A, p)$.

We note that the actual implementation does not require full recursion. At each iteration we compute the decomposition and identify the activity instances belonging to $A - A'$. We then push these instances onto a stack, delete them, and reiterate with $p \leftarrow p_2$. When $p$ becomes identically 0, we pop each activity instance in turn off the stack and add it to the schedule if doing so does not violate the feasibility of the schedule.

How do we determine the decomposition of the profit function? This, too, is done by a generic procedure. Let $I$ be an instance with the earliest end time among all activity instances (of all activities) and let $\bar{R}$ be the activity to which it belongs. Let $\bar{I}$ be the set of all instances $I$ such that $s(I) \leq e(\bar{I}) \leq e(I)$ and $I \not\in \bar{R}$, i.e., all instances that intersect $\bar{I}$ but belong to other activities. To determine $p_1$, we use two scalers on the profits of instances. Specifically, we choose a constant $\alpha$ and a function $\beta(I)$, both positive, and define

$$\delta = \min \left\{ \min_{I \in \bar{R}} \frac{p(I)}{\alpha}, \min_{I \in \bar{I}} \frac{p(I)}{\beta(I)} \right\}.$$ 

The decomposition is $p = p_1 + p_2$ where

$$p_1(I) = \begin{cases} \delta \cdot \alpha & I \in \bar{R}, \\ \delta \cdot \beta(I) & I \in \bar{I}, \\ 0 & \text{otherwise}. \end{cases}$$

The choice of the scalers $\alpha$ and $\beta(I)$ and the proof that this decomposition has the desired property stated above will be given for each problem separately.

In the loss minimization problem (Section 4) we use a somewhat different decomposition.

**Time complexity.** The time complexity of the algorithm depends on the way the sets of activity instances are given and on the value of the scalers. The easier case is when the instances are given explicitly. A straightforward implementation of the algorithm in this case is quadratic in the number of instances, independent of the scalers, since at each iteration the profit of at least one additional instance drops to 0 and it is deleted. In case all the scalers $\beta(I)$ are the same, as is the case, for example, in the interval scheduling, and the bandwidth allocation on a line applications, a more sophisticated implementation exists which runs in $O(n \log n)$ time. The key idea is to use a sweep-line algorithm with a heap and to increase the profits of the intervals in $A'$ rather than decrease those of the intervals in $A - A'$. We omit the details for lack of space.

In some of the applications the instances are not given explicitly. Moreover, in some cases, the number of instances is even uncountable. For example, in real-time scheduling the set of instances is given as a “window” of time in which the instance may be scheduled at any time in the window. Using a novel discretization idea, we are able to implement this case in polynomial time losing only a multiplicative factor of $(1 - \varepsilon)$ relative to the approximation ratio of the corresponding problem.
Let the value of the solution be \( p \cdot x \). A feasible solution is optimal if its value is maximal (or minimal) among all feasible solutions. A feasible solution \( x \) is an \( r \)-approximation if \( p \cdot x \geq (or \leq) r \cdot p \cdot x^* \) where \( x^* \) is an optimal solution.

**Theorem 2.1 (Local Ratio)** Let \( F \) be a set of constraints and let \( p, p_1, \) and \( p_2 \) be profit (or penalty) vectors such that \( p = p_1 + p_2 \). Then, if \( x \) is an \( r \)-approximation both with respect to \((F, p_1)\) and with respect to \((F, p_2)\), then \( x \) is an \( r \)-approximation with respect to \((F, p)\).

**Proof:** Let \( x^*, x^*_1, x^*_2 \) be be optimal solutions for \((F, p), (F, p_1), \) and \((F, p_2)\) respectively. Then \( p \cdot x = p_1 \cdot x + p_2 \cdot x \geq (or \leq) r \cdot p_1 \cdot x_1 + r \cdot x_2 \geq (or \leq) r \cdot (p_1 \cdot x^* + p_2 \cdot x^*) = r \cdot p \cdot x^* \). \( \square \)

The local ratio formulation contains all linear and integer programming problems, but is not restricted to them, since arbitrary feasibility constraints are allowed. While the Local Ratio Theorem is valid for all problems in this formulation, it is not always clear how to use it successfully. In fact, all successful applications to date are to problems in which the constraints are linear or linear integer. The problems treated in this paper also fall into this category.

### 3 Throughput Maximization

The throughput maximization problem is the problem of finding a feasible schedule of the activities that maximizes the total profit accrued. More formally, the goal is to find an optimal solution to the following integer programming problem.

Maximize \( \sum_{i=1}^{n} \sum_{l \in R_i} p(I) \cdot x(I) \) subject to:

At each time \( t \):

\[
\sum_{i=1}^{n} \sum_{l \in R_i, s(l) \leq t \leq e(l)} w(I) \cdot x(I) \leq \text{Width}(t),
\]

For each activity \( i, 1 \leq i \leq n \):

\[
\sum_{l \in R_i} x(I) \leq 1.
\]

For all \( 1 \leq i \leq n \) and \( I \in R_i \):

\[
x(I) \in \{0, 1\}.
\]

#### 3.1 The Unified Algorithm

We use a generic scheme based on the Local Ratio Theorem to approximate the throughput maximization problem. We denote an instance of the problem by \((A, p)\) where \( p \) is the profit function and \( A \) denotes the specification of the activities excluding the profits. The idea is to decompose \( p \) by \( p = p_1 + p_2 \), and to solve recursively \((A', p_2)\) where \( A' \) is obtained from \( A \) by deleting activity instances whose \( p_2 \)-profits are 0. The schedule obtained recursively for \((A', p_2)\) is then extended by scanning the activity instances in \( A - A' \) in an arbitrary order and adding an instance to the schedule if its addition does not violate the feasibility of the schedule. The recursion stops when \( A = \emptyset \), at which time the empty schedule is returned to the next higher level. The schedule thus generated has the all-important property that it is maximal with respect to containment.

Why is this an application of the Local Ratio Theorem, and how do we guarantee that the solution obtained is of good quality? Consider the decomposition \( p = p_1 + p_2 \). We choose this decomposition such that for an appropriately chosen \( r \), any feasible schedule that is maximal with
as the time axis and the y-axis as representing the bandwidth. A dual problem considers a line of processors on the x-axis, where sessions, which are connections between pairs of processors, are permanent in time. The y-axis is still the bandwidth axis (see [4]). Clearly, all of our results regarding the time axis interpretation hold for the processors axis interpretation as well. They hold for both throughput maximization and loss minimization. They also hold for the contiguous case.

An interesting application of our general framework concerns the ring topology. The ring topology is considered a viable network topology in the optical network setting and it is well studied in the context of bandwidth allocation. In the ring topology a session may choose between two routes: going clockwise and going counter clockwise. In our framework, this is equivalent to two instances per each activity. Traditionally, in optical networks, the identical width case was considered in which each session requires one wavelength. Our solutions address the general case of more than one wavelength per session. We show how to adapt any approximation result for the throughput maximization problem on the line into one for the same problem on the ring. If the approximation factor for the line is $1/\rho$, the resulting factor for the ring will be $1/(\rho + 1 + \varepsilon)$ where $\varepsilon$ may be chosen arbitrarily small (the running time of the algorithm depends polynomially on $1/\varepsilon$).

2 Preliminaries

The input consists of a resource and activities. The amount of the resource available is given as a function of “time”. (We use the term time to simplify the exposition. However, the same also applies when the resource is given as a function of other variables.) Our terminology is inspired by bandwidth allocations problems. Hence, we denote the amount of resource available at time $t$ by $\text{Width}(t)$ and we speak of the width of an activity meaning the amount of resource it requires. The activities are given as a collection of sets $R_1, \ldots, R_n$. Each set represents a single activity; it consists of all possible instances (time intervals) in which the activity can be scheduled. Each instance $I$ is defined by its start time $s(I)$, its end time $e(I)$, its width $w(I)$, and the profit $p(I)$ gained by scheduling the activity at this instance. Different instances of the same activity may have different parameters of duration, width, or profit. A feasible schedule is a schedule in which at most one instance of every activity is scheduled and the total width of the instances scheduled at any time $t$ does not exceed $\text{Width}(t)$. In the throughput maximization problem we are asked to find a feasible schedule that maximizes the total profit accrued by instances in the schedule. In the loss minimization problem we are asked to find a feasible schedule of minimum cost, where the cost of a schedule is defined as the total profit of instances not in the schedule. For a given profit function $p$, we use the term $p$-profit to refer to the profit (of a single instance or a set of instances) with respect to $p$. We use $p$-cost similarly.

2.1 The Local Ratio Technique

Our algorithms use the Local Ratio Theorem (and technique) first developed by Bar-Yehuda and Even [7], later extended by Bafna, Berman, and Fujito [3], and recently extended by Bar-Yehuda [6], all of whom treated minimization covering problems. In this paper we further extend the theorem to cover all problems of the form: given a profit (or penalty) vector $p \in \mathbb{R}^n$, find a solution $x$ that maximizes (or minimizes) the inner product $p \cdot x$ subject to some set $\mathcal{F}$ of feasibility constraints on $x$.

A vector $x$ is a feasible solution to a given problem $(\mathcal{F}, p)$ if it satisfies all of the constraints in $\mathcal{F}$.
allocation problem is the problem of finding the most profitable set of sessions that could utilize the available bandwidth. Our framework includes this problem. Moreover, we capture the case in which each session has either a window of time in which it can be scheduled, or a set of discrete intervals in which it can be scheduled. Our algorithm achieves a factor of $1/5$ for this case. Prior to our paper, no constant factor approximation algorithms for this problem were known. We note that using the techniques of Albers et al. [1], it seems that a constant factor approximation (where the constant is at least $22$) can be obtained for the version where each session can be scheduled in precisely one time interval [14]. Finally, our model also captures the case in which sessions do not specify needed bandwidth and are characterized only by their volume and their deadline. It is the freedom of the algorithm to schedule them at any time with enough bandwidth so that they finish before their deadline.

**General caching.** The general caching problem models situations in which a cache is to be used for pages of varying sizes and the fault penalty may vary as well. Specifically, the input consists of a cache size and a sequence of requests for pages. Each request is defined by its time and the page it refers to. At each moment in time exactly one page is requested, and without loss of generality we may assume that time is discrete. In addition, each page has a size and a reload cost that is incurred whenever the page is reloaded into the cache. A replacement schedule is a specification of the contents of the cache at all times. A replacement schedule is feasible if the total size of pages in the cache never exceeds the cache size and if for all times $t$, the page requested at time $t$ is in the cache at time $t$. Pages may be moved around in the cache at no cost so memory fragmentation is never a problem. The initial contents of the cache may be chosen by the scheduler at no cost. A request for page $g$ at time $t$ is said to incur the reload cost of $g$ if $g$ is not in the cache at time $t-1$. The cost of a replacement schedule is the sum of the reload costs incurred by all requests. The goal is to find a replacement schedule of minimum cost. We consider the off-line version of this problem, i.e., the case where the input is given ahead of time. As was shown in [1], the general caching problem can be modeled in our framework where the objective is to minimize the loss accrued by the activities that are not scheduled. Our algorithm yields a $4$-approximation for this problem. This improves the result of Albers et al. [1]. They were able to achieve an $O(1)$-approximation factor using LP rounding only by increasing the size of the cache by $O(1)$ times the largest page size. If the cache size is not increased, then [1] obtained only a logarithmic factor approximation.

**Contiguous allocation problems.** Suppose that the allocated resources must be contiguous in contrast to bandwidth allocation and machine scheduling applications. For example, in the dynamic storage allocation problem, the memory allocated for objects must reside in contiguous memory units. The traditional goal in contiguous allocation has been to store all given activities in minimum size memory [15, 12]. For contiguous allocation problems, to the best of our knowledge, we are the first to consider the throughput version. We are able to obtain a $1/35$ factor approximation. Moreover, we solve the dynamic storage allocation problem in the general case where objects can be scheduled in more than one time interval. Informally, we show how to transfer a solution for the non-contiguous version of the problem into a solution for the contiguous version and losing only a constant factor. Another example of contiguous allocation is strip packing [17] where the goal is to pack rectangles into a strip. Our constant-factor bound apply to this problem as well.

**The line and ring topology networks.** In the bandwidth allocation problem we assumed a connection between two end-points and sessions that are temporary in time. We viewed the x-axis
1 Introduction

We study the allocation and scheduling of resources over time to a set of activities such that a profit function is maximized. Suppose that a fixed size resource is available. A typical request for resources by an activity specifies a set of time intervals in which it can be scheduled, together with the amount of resources required and the profit obtained if the activity is scheduled. We allow both the resource requirement and the profit to vary over the different time intervals. We also consider the case where a loss function is associated with the activities and the goal is to minimize the total loss accrued by activities that were not scheduled.

This scenario models a wide range of applications. Two prominent problems fall immediately into this framework: bandwidth allocation for sessions in communication networks and machine scheduling of jobs. General caching also fits in this framework (we will explain how later). Finally, the case where the allocation is required to be “contiguous”, e.g., the dynamic storage allocation problem, is also related to our framework.

The simplest problem which can be cast in our framework is the maximum weight independent set in interval graphs \([13]\) which can be solved efficiently in polynomial time. This problem corresponds to the case of a single machine where each activity can be scheduled in precisely one time interval. However, the problem becomes NP-hard when either the resource requirement or the number of feasible time intervals (per activity) are relaxed: (i) If activities may require different amount of resources, then the problem is NP-hard. (The knapsack problem \([10]\) is a special case). (ii) If each activity may be scheduled in several time intervals, a problem known as interval scheduling, then the problem is NP-hard \([10]\).

1.1 Our contribution

We provide constant-factor approximation algorithms for all the problems that are formulated in our model using a novel technique for combining time and resource constraints. For some of the problems, this is the first constant factor approximation algorithm. Our algorithms are extremely simple and efficient and they are based on the local ratio technique \([7]\). Alternatively, we show how to interpret our algorithms within the primal-dual schema. We note that obtaining a primal-dual algorithm for a (natural) maximization problem was posed as an open problem \([18]\).

Machine scheduling. The resource consists of \(k\) machines and the activities are jobs, where for each job several time intervals are available. This problem was considered by \([5]\) who gave an approximation factor of 1/2 based on LP rounding. We provide an efficient combinatorial algorithm that achieves the same approximation factor. Bar-Noy et al.\([5]\) also considered the version of the problem where the input is given in a continuous form, i.e., for each activity we are given a release time, a deadline, and a processing time on each of the machines. (The processing time is typically much smaller than the available window.) For this version, \([5]\) gave approximation factors of 1/3 (for single machine) and 1/4 (for parallel machines). Our algorithm achieves a factor of \(1/2 - \varepsilon\) for this version.

Bandwidth allocation of sessions in communication networks. In modern communication networks (e.g., ATM networks), there exists some available bandwidth over time between two endpoints which is either constant over time (CBR) or variable over time (VBR). The bandwidth
A Unified Approach to Approximating Resource Allocation and Scheduling

Amotz Bar-Noy*  Reuven Bar-Yehuda†  Ari Freund‡  Joseph (Seffi) Naor§
Baruch Schieber¶

Abstract

This paper addresses a general framework for resource allocation and scheduling problems. Given a resource of fixed size, we present algorithms that approximate the maximum throughput or the minimum loss by a constant factor. Our approximation factors apply to many prominent problems among others: (i) Real-time scheduling of jobs on parallel machines. (ii) Bandwidth allocation for sessions between two endpoints. (iii) General caching. (iv) Dynamic storage allocation. (v) Bandwidth allocation on optical line and ring topologies. For many of the problems, this is the first constant factor approximation algorithm. Our algorithms are extremely simple and efficient. They use the local-ratio technique but they can also be equivalently interpreted within the primal-dual schema.