IIE Transactions (2002) **34**, 501–507

Allocation of bandwidth and storage

BO CHEN[1], REFAEL HASSIN[2] and MICHAL TZUR[3]

[1]*Warwick Business School, University of Warwick, Coventry CV4 7AL, UK*
E-mail: b.chen@warwick.ac.uk
[2]*Department of Statistics and Operations Research and*
[3]*Department of Industrial Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel*
E-mail: hassin@math.tau.ac.il or tzur@eng.tau.ac.il

Received May 1998 and accepted August 2001

We consider two allocation problems in this paper, namely, allocation of bandwidth and storage. In these problems, we face a number of independent requests, respectively, for reservation of bandwidth of a communication channel of fixed capacity and for storage of items into a space of fixed size. In both problems, a request is characterized by: (i) its required period of allocation; (ii) its required bandwidth (item width, respectively); and (iii) the profit of accepting the request. The problem is to decide which requests to accept so as to maximize the total profit. These problems in general are NP-hard. In this paper we provide polynomial-time algorithms for solving various special cases, and develop polynomial-time approximation algorithms for very general NP-hard cases with good performance guarantees.

1. Introduction

We are given a set $N = \{1, \ldots, n\}$ of independent requests for reservation of bandwidth of a communication channel of capacity K. Request i $(i \in N)$ is characterized by a quadruple (s_i, t_i, d_i, w_i), where s_i, $t_i \in [0, T]$ are the start and finish time, respectively, of the required reservation period, d_i $(1 \leq d_i \leq K)$ the requested bandwidth for reservation, or its *size*, and $w_i \geq 0$ the profit for accepting the request or its *weight*. We assume without loss of generality that all the aforementioned numbers are integer.

The *Bandwidth Allocation Problem* (BAP) is to decide which requests to *accept* (or to *satisfy*) so as to maximize the total weight of accepted (satisfied) requests subject to the condition that the total size of accepted requests at any time must not exceed capacity K. An important special case of the BAP is that of *proportional weights*, in which the weight of request i is $(t_i - s_i)d_i$ for all $i \in N$.

The mathematical model for the *Storage Allocation Problem* (SAP) is the same as that for the BAP except that an accepted request i should be allocated a spatial interval $[y_i + 1, y_i + d_i]$, where y_i is integer and $0 \leq y_i < K$, and the two spatial intervals allocated to any two accepted requests should be disjoint, if the reservation periods of the two requests intersect in their interior. A geometric interpretation of SAP is to choose from a given set of rectangles, each of which has a fixed orientation, a given weight and size, and is only allowed to move vertically along its two parallel edges, and then pack without overlap the chosen rectangles into a rectangular frame of given size, so as to maximize the total weight of the chosen rectangles.

Note that the two allocation problems are fundamentally different from each other. Figure 1 illustrates a feasible bandwidth allocation of seven requests with sizes of one or two, where the horizontal and vertical axes represent respectively time and space. It is evident that no feasible storage allocation exists for the corresponding instance of packing seven rectangles, where the seventh rectangle is the vertical concatenation of the two smaller squares.

Bandwidth allocation is a fundamental problem in the design of networks where bandwidth has to be reserved for connections in advance. When the requested total bandwidth exceeds the capacity, the problem is intensified, about which extensive research has been conducted, see, for example, the paper of Harms and Wong (1995) and Bar-Noy *et al.* (1999). A special case of the BAP, where all requests have the same size, has been studied in the context of scheduling jobs with fixed start and end times by Arkin and Silverberg (1987). The special case with proportional weights is considered by Bar-Noy *et al.* (1999) in an on-line setting, where the requests arrive over time and one has to make a decision as to whether the arriving request should be accepted and if accepted, which requests that are currently being processed should be abandoned to make room for the new request. They provide two heuristic algorithms with a *performance guarantee* of $(1 - 2r)/2$ and $(1 - r)/4$, respectively, where $r = \max_j d_j/K$, i.e., they guarantee that the value of a

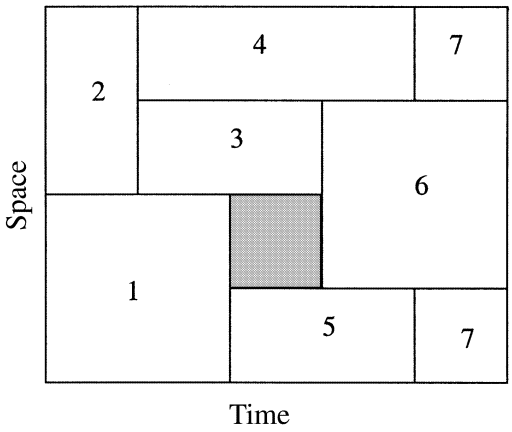

Fig. 1. A feasible bandwidth allocation of seven requests, for which there is no feasible storage allocation of the corresponding seven items.

heuristic solution will be at least $(1-2r)/2$ and $(1-r)/4$, respectively, times that of a corresponding off-line optimal solution.

Storage allocation can arise from a similar situation (Bar-Noy *et al.*, 1999). A sequence of stations are connected on a line by communication links of unit capacity. There are a number of calls, each of which, of its own value, is a request i for connection between two stations s_i and t_i on the line for a time interval of specified length d_i. Due to the capacity constraint, one cannot accept two requests that share a common link on the line. The objective is to accept those requests so that the total value is maximized.

Closely related to SAP is the following *Dynamic Storage Allocation* (DSA) problem, which first arose in computer science. Given a positive storage size K and a collection of blocks (or memory requests) $\{(s_i, t_i, d_i)\}$ for $i \in N$, where s_i, t_i and d_i describe respectively the arrival and departure time and the size of the ith block. The question arises as to whether there exists a feasible allocation in the memory of size K for the blocks. In terms of the SAP, the question becomes whether accepting all these requests is feasible. The optimization version of the DSA problem is to find the minimum memory capacity K sufficient to accommodate all memory requests. This problem has been studied extensively in the literature. See, for example, the papers of Robson (1974), Coffman (1983), Ślusarek (1987a), Gergov (1999), Bar-Noy *et al.*, (2000) and Phillips *et al.* (2000). Surprisingly little research can be found on the SAP.

Both of our allocation problems can also be viewed as *rental* problems. In such a problem, a firm owns K identical units of a given item, and faces a set of requests from customers who wish to rent the units. Each request is specified by the start and finish time of the rental period, the number of units requested and the profit obtained from satisfying the rental request. The rental problem is to decide which of these rental requests have to be rejected due to the limited number of units, so that the total profit is maximized. In this context, the SAP arises when adjacency of the units is required.

In Bar-Noy *et al.* (2000) and Phillips *et al.* (2000) approximation algorithms with performance guarantees of $1/3$ and $1/35$ are developed for the BAP and the SAP, respectively.

The main results in this paper are as follows: (i) an $O(n^{K+1})$ time dynamic programming algorithm for the BAP and an $O(n(Kn)^K)$ time dynamic programming algorithm for the SAP; (ii) a polynomial-time approximation algorithm for the BAP with proportional weights that has a performance guarantee of $\max\{1/3-\epsilon, 1/2-r, (1-r)/3\}$ for any $\epsilon > 0$, where $r = \max_j d_j/K$ as defined above; and (iii) a polynomial-time approximation algorithm for the SAP with bounded request sizes that has a performance guarantee of $0.632(1-\epsilon)$ for any $0 < \epsilon < 1$. We shall derive these results in the following three sections respectively.

2. Preliminaries and solvable cases

Let us first make a straightforward observation.

Observation 1

In the BAP, if a set of requests can be satisfied, then it can be satisfied in such a way that each satisfied request is allocated with the same set of bandwidth units throughout its whole reservation period. In the SAP, if the spatial interval allocated to each request (item) is not necessarily required to be the same during its entire occupation period, then the problem is equivalent to the BAP.

The BAP is a special case of the multi-dimensional binary knapsack problem: maximizing $\{cx \mid Ax \le b, x \in \{0,1\}^n\}$, where $A \in \mathbb{R}_+^{m\times n}$, $c \in \mathbb{R}_+^n$ and $b \in \mathbb{R}_+^m$. In fact, a BAP is readily obtained by setting $b_i = K$ for all i and, for all j, by setting $a_{ij} = d_j$ for i: $s_j \le i \le t_j$ and $a_{ij} = 0$ for other i. For the m-dimensional knapsack problem, if m is a fixed constant, then there are known polynomial-time approximation schemes (Chandra *et al.*, 1976; Frieze and Clarke, 1984), which implies that, if the time horizon is fixed, our BAP can be well-approximated even when each request consists of several non-consecutive time intervals. However, in our consideration here in this paper, we allow the length of the time horizon be part of the input.

On the other hand, we also observe that the one-dimensional binary knapsack problem is a special case of both allocation problems. In fact, any instance of the one-dimensional binary knapsack problem, maximizing $\sum_{i=1}^n c_i x_i$ subject to $\sum_{i=1}^n a_i x_i \le b$ and $x_i \in \{0,1\}$ for all i, can be regarded as a simple BAP as well as a simple SAP with capacity $K = b$, weights $w_i = c_i$, sizes of requests $d_i = a_i$, time horizon [0, 1] where $s_i = 0$ and $t_i = 1$ for all i. Since the Subset Sum problem, a special one-dimensional binary knapsack problem with $c_i = a_i$ for all i, is NP-hard, we conclude that:

Observation 2

Both the BAP and SAP are NP-hard even for a very restricted special case, where all s_i are equal to zero, all t_i are equal to one and $w_i = d_i$.

Note that the observation implies that the BAP with proportional weights is NP-hard. On the other hand, the SAP is even strongly NP-hard, even with bounded request sizes, since its decision version the DSA problem is strongly NP-complete (Garey and Johnson, 1979), and it remains so with only sizes of one and two (Ślusarek, 1987b). In what follows we first identify some polynomially solvable cases of the problems, and then we consider an approximation for the BAP with proportional weights and also the SAP with bounded request sizes, both of which are still NP-hard as we have mentioned above.

Let us start with the observation that, without loss of generality, we can assume that the values $s_1, \ldots, s_n, t_1, \ldots, t_n$ are distinct. Otherwise, we can perturb the data to achieve this property without affecting the feasibility or infeasibility of any subset of requests. On the other hand, it is easy to see that, while the actual values of s_i and t_i are immaterial, it is their relative orders on the time horizon that determine the feasibility of accepting a subset of requests. Therefore, unless otherwise stated (e.g., in the case of proportional weights), we will assume that $\{s_1, \ldots, s_n, t_1, \ldots, t_n\} = \{0, 1, \ldots, 2n-1\}$. In particular, $T = 2n - 1$.

The *uniform* case where $d_i = 1$ for all $i \in N$ is a special case of both allocation problems. It can be formulated as a problem of coloring interval graphs or as that of finding a minimum cost flow (Arkin and Silverberg, 1987). Therefore, it is polynomially solvable (Papadimitriou and Steiglitz, 1982).

The more restricted uniform case where $K = 1$ reduces to the problem of max-weight interval packing. It can be solved in linear time once the endpoints of the intervals are ordered.

Actually, for each of the two problems in their general forms, we can provide a dynamic programming algorithm, which shows that they are polynomially solvable once the capacity K is a constant. More generally, in Sections 2.1 and 2.2, we shall show the following:

Theorem 1. *For the generalized BAP and SAP where, in addition to the capacity constraint, there is a cardinality one, i.e., the number of accepted requests should be no more than* L $(0 < L \leq +\infty)$ *at any time, there are algorithms that run in* $O(n^{c+1})$ *and* $O(n(Kn)^c)$ *time, respectively, where* $c = min\{K, L\}$. *In particular, if in the original BAP and SAP the capacity is not part of the input, i.e., if* K *is a constant, then both problems are polynomially solvable.*

2.1. *The DP algorithm for the generalized BAP*

Renumber the requests so that $s_1 < s_2 < \cdots < s_n$. For notational simplicity, let $s_{n+1} = T + 1 = 2n$. For $i \in N$ define $S_i = \{j \in N : s_i \in (s_j, t_j)\}$ and $\mathscr{S}_i = \{S \subseteq S_i : \sum_{j \in S} d_j \leq K \text{ and } |S| \leq L\}$. For any $S \in \mathscr{S}_i$, let $f_i(S)$ be the weight of an optimal solution if the available time period for allocation is restricted to the subinterval $[s_i, T]$ given that the requests $j \in S$ are accepted but their weights are not added. Hence $f_{n+1}(S) = 0$ for any $S \subseteq N$ by definition. Given any $S \in \mathscr{S}_i$, if $\sum_{j \in S} d_j > K - d_i$ then $f_i(S) = f_{i+1}(S')$ where S' is obtained from S by removing those requests whose termination occurs before s_{i+1}. If $\sum_{j \in S} d_j \leq K - d_i$ and $t_i < s_{i+1}$ then $f_i(S) = w_i + f_{i+1}(S')$. Otherwise,

$$f_i(S) = \max\{f_{i+1}(S'), w_i + f_{i+1}(S' \cup \{i\})\}.$$

Since there are $O(n^c)$ ways to choose a set of c requests from N, we have $|\mathscr{S}_i| = O(n^c)$, where $c = \min\{K, L\}$. The time complexity of the algorithm is thus $O(n^{c+1})$.

2.2. *The DP algorithm for the generalized SAP*

In the case of storage allocation, our dynamic program uses a more detailed description of the states. A state describes now not only the set of accepted requests at the relevant time point but also their physical positions. It indicates whether and which request occupies each position (s_i, y) for all $i = 1, \ldots, n$ and all $y = 1, \ldots, K$. The feasibility rule for accepting the ith request is also different, as it not only compares the unused capacity with d_i, but also verifies that there is an unused (vertical) spatial interval of length d_i.

Assume $s_1 < \cdots < s_n$ and let s_{n+1} and S_i be defined as in Section 2.1. Define $P_i = \{(j, y) : j \in S_i \text{ and } 1 \leq y \leq K - d_j + 1\}$, where a pair $(j, y) \in P_i$ indicates that the lower side of item j is placed in position y. Define $\mathscr{P}_i$ to be the set of subsets $P \subseteq P_i$ such that pairs in P indicate a feasible allocation (with respect to capacity, cardinality and spatial constraints). For $P \in \mathscr{P}_i$, let $Y_i(P)$ be the set of co-ordinates $y \in \{1, \ldots, K - d_i + 1\}$ such that no item involved in placements P occupies any of the positions $(s_i, y), \ldots, (s_i, y + d_i - 1)$. Thus, (s_i, y) for any $y \in Y_i(P)$ is a candidate position for placing the lower side of item i if accepted.

For any $P \in \mathscr{P}_i$, define $g_i(P)$ to be the weight of an optimal solution if the available time period for allocation is restricted to the subinterval $[s_i, T]$ given that allocations indicated by P are accepted but the weights of allocated items are not added. Hence $g_{n+1}(P) = 0$ for any P by definition. Given any $P \in \mathscr{P}_i$, if $Y_i(P) = \emptyset$ then $g_i(P) = g_{i+1}(P')$ where P' is obtained from P by removing those requests whose termination occurs before s_{i+1}. If $Y_i(P) \neq \emptyset$ and $t_i < s_{i+1}$ then $g_i(P) = w_i + g_{i+1}(P')$. Otherwise,

$$g_i(P) = \max\left\{g_{i+1}(P'), w_i + \max_{y \in Y_i(P)} \{g_{i+1}(P' \cup \{(i, y)\})\}\right\}.$$

Since there are $O((Kn)^c)$ ways to choose c pairs from at most nK, we have $|\mathscr{P}_i| = O((Kn)^c)$, where $c = \min\{K, L\}$.

The complexity of solving the dynamic program is thus $O(n(Kn)^c)$.

3. The BAP approximation

We move to considering polynomial approximations in solving problems of bandwidth and storage allocation in their general forms. We first suggest an approximation framework for the BAP. Denote $r = \max_{j \in N} d_j/K$. Suppose that we are given an approximation algorithm that has a performance guarantee of $a - br$ for some constants a and b. Such algorithms have been developed in Bar-Noy *et al.* (1999) for an on-line setting of BAP, and can be used, of course, also in the off-line setting. However, we will use the additional information available in our off-line environment to improve such bounds and obtain bounds independent of r.

Approximation framework

For a fixed $r_0 > 0$, a request i is called *big* if $d_i \geq r_0 K$. Otherwise it is called *small*. Since any feasible solution can accommodate at most $1/r_0$ big requests at a time, we apply the dynamic programming algorithm of Section 2.1 with $L = 1/r_0$ according to Theorem 1 to find a feasible solution that is of maximum weight among those consisting of only big requests. Such a procedure takes polynomial-time. (In fact, $O(n^{1+1/r_0})$.) We also apply an approximation algorithm with performance guarantee of $a - br_0$ to the set of small requests. We output the better of the two solutions as our approximate solution.

Let α denote the proportion of weight that is contributed by big requests in some optimal solution. Then our algorithm has a performance guarantee of at least $\max\{\alpha, (1-\alpha)(a - br_0)\} \geq (a - br_0)/(1 + a - br_0)$, which is independent of α. By selecting r_0 arbitrarily small we can approach $a/(1+a)$ arbitrarily closely.

In the rest of this section, we assume proportional weights. Therefore, all the s_i's and t_i's assume general non-negative integer values.

Applying our approximation framework to the algorithm of Bar-Noy *et al.* (1999) with a performance guarantee of $1/2 - r$, we obtain a polynomial-time approximation algorithm with a performance guarantee of $(1/3 - \epsilon)$, for any $\epsilon > 0$, which is independent of r. This in turn gives rise to a polynomial-time approximation algorithm A_1 with an improved performance guarantee of $\max\{1/2 - r, 1/3 - \epsilon\}$. Nevertheless, we now investigate a much simpler and faster algorithm.

Algorithm A_2

Arrange all the requests in a priority list in order of non-increasing lengths $t_i - s_i$. Then in turn remove the first request in the current list and accept it if and only if it can be accommodated into the current system.

Now consider the performance of algorithm A_2. For a set $N' \subseteq N$ of requests and integer time point $t \in [1, T]$, let the total size of requests in N' at time t be

$$m_t(N') = \sum_{\substack{j \in N' \\ t \in (s_j, t_j]}} d_j.$$

Let $\hat{m}_t(N') = \min\{m_t(N'), K\}$ and let the *coverage* of a request set N' be

$$c(N') = \sum_{t=1}^{T} \hat{m}_t(N').$$

Note that $c(N') \leq w(N')$, where $w(N')$ denotes the total weight of requests in N', and the coverage of a request set N' is an upper bound on the total weight of any feasible solution for N', and it is the total weight of all requests in N' if N' itself is feasible. Let $A_2(N)$ denote the set of requests accepted by Algorithm A_2.

Lemma 1. $c(A_2(N))/c(N) \geq (1 - r)/3$.

Proof. Let $E = A_2(N)$. We introduce a set E' of *shadow requests*, and show that:

$$c(E) \geq \frac{1-r}{3} c(E'), \tag{i}$$

and

$$c(E') \geq c(N), \tag{ii}$$

which then imply the theorem.

For each request $e \in E$, we define a corresponding shadow request e' with start time $2s_e - t_e$, finish time $2t_e - s_e$ and size $d_e/(1-r)$. For a set $F \subseteq E$ of requests, let $F' = \{e' : e \in F\}$. Note that the newly defined start times may be negative and the finish times may exceed T, but this does not affect our following discussion. Also note that shadow request e' is a combined result of stretching request e uniformly along its two sides to triple its length and stretching its size by a factor of $(1-r)^{-1}$. Consequently, the weight of e' is $3(1-r)^{-1}$ times that of e. Hence

$$c(E) = w(E) = \frac{1-r}{3} w(E') \geq \frac{1-r}{3} c(E').$$

In the following, it suffices to validate inequality (ii) by showing that $\hat{m}_t(E') \geq \hat{m}_t(N)$ for all $t = 1, \ldots, T$. We distinguish two cases.

Case 1. Time t is not contained in the reservation period of any request rejected by Algorithm A_2, i.e., $t \notin (s_j, t_j]$ for all $j \in N \setminus E$. In this case, it is evident that $\hat{m}_t(E') \geq \hat{m}_t(E) = \hat{m}_t(N)$.

Case 2. Time t is contained in the reservation period of a rejected request. Let $j \in N \setminus E$ be a request such

that $t \in (s_j, t_j]$. Since request j is rejected, there exists a time $t' \in (s_j, t_j]$ and a subset $F \subseteq E$ of accepted requests, each of which has a reservation period containing t' and has a length of at least as large as request j, such that $m_{t'}(F) > K - d_j \geq (1-r)K$. Clearly, for each $e \in F$, its corresponding shadow request e' has a reservation period that entirely covers the reservation period $(s_j, t_j]$ of request j and hence contains time $t \in (s_j, t_j]$. Therefore,

$$\hat{m}_t(E') \geq \hat{m}_t(F') = \hat{m}_{t'}(F') = \min\{m_{t'}(F'), K\},$$
$$= \min\{(1-r)^{-1} m_{t'}(F), K\} = K \geq \hat{m}_t(N).$$

■

Theorem 2. *For a BAP with proportional weights, greedy algorithm A_2 runs in $O(n^2)$ time and has a performance guarantee of $(1-r)/3$. The bound is tight.*

Proof. A subset N' of N is a set of accepted requests in a feasible solution for N if and only if $m_t(N') \leq K$ for all $t \in [1, T]$. Since $A_2(N)$ is feasible, its value is $c(A_2(N))$. On the other hand, since the coverage of request set N is an upper bound on the total weight of any feasible solution for N, it follows from Lemma 1 that the performance guarantee of the algorithm is as claimed.

We now consider the running time. Sort all s_i's and t_i's in a single list in non-decreasing order. Let the resulting list be $u_1 \leq u_2 \leq \cdots \leq u_{2n}$. Sort all requests in a priority list in order of non-increasing lengths. After these two sortings, which takes $O(n \log n)$ time, Algorithm A_2 updates the solution by checking the accommodation for each request in the priority list. To check whether a request can be accommodated into an existing partial feasible solution, it suffices to keep an updated $2n$-dimensional vector $(m_{u_1}(E), \ldots, m_{u_{2n}}(E))$, where $E \subseteq N$ is the set of requests accepted to date. In other words, Algorithm A_2 builds a set of accepted requests step-by-step through recording the total size of accepted requests at each start and end point. Therefore, the process of updating takes $O(n^2)$ time.

To see that the bound is tight, consider the following set of $n = 4((1-r)K + 1)$ requests. The first $(1-r)K + 1$ requests, each having a size of one unit, all start at

$$\frac{1}{3}T - 1,$$

and finish at

$$\frac{2}{3}T + 1.$$

The remaining $3((1-r)K + 1)$ requests are equally divided into three groups, each having $(1-r)K$ requests of unit size and one request of size rK, have two endpoints

$$\left[0, \frac{1}{3}T\right], \left[\frac{1}{3}T, \frac{2}{3}T\right],$$

and

$$\left[\frac{2}{3}T, T\right],$$

respectively. Clearly, the greedy algorithm A_2 accepts the first $(1-r)K + 1$ requests with a total value of

$$\left(\frac{1}{3}T + 2\right)((1-r)K + 1),$$

while the optimal solution is to accept the last $3((1-r)K + 1)$ requests with a total value of TK. Therefore, the ratio goes to $(1-r)/3$ when K and T become large. ■

Combining Algorithms A_1 and A_2, we obtain the following.

Corollary 1. *For a BAP with proportional weights, there is a polynomial-time approximation algorithm with a performance guarantee of $\max\{1/3 - \epsilon, 1/2 - r, (1-r)/3\}$ for any $\epsilon > 0$.*

4. The SAP approximation

In this section we present a polynomial-time approximation algorithm for the SAP of general weights, assuming that $q = \max_{j \in N} d_j = rK$ is bounded. Since we can assume without loss of generality that $K < \sum_i d_i \leq nq$, we have $K = O(n)$. For an integer $\lambda > 0$, define the *λ-restriction* of the SAP as the problem of packing a subset of rectangles in N into $m = K/(\lambda q)$ identical rectangular *frames*, each having a width of λq and a length of T, with the objective of maximizing the total weight of packed rectangles. (For simplicity we assume that m is an integer.) Clearly the total weight of an optimal solution to the λ-restriction is no more than that of an optimal solution to the original SAP. However, we show that the difference of the two is relatively small. On the other hand, we establish that λ-restriction can be formulated as a problem that can be approximated efficiently.

Lemma 2. *Let OPT and OPT_λ be the optimal values of SAP and its λ-restriction, respectively. Then*

$$OPT_\lambda \geq (1 - 2/\lambda) OPT.$$

Proof. Consider an optimal solution to the original SAP and let $E \subseteq N$ be the set of rectangles that are packed in the optimal solution. Divide the storage frame horizontally into subframes, each having a width of λq. We show that, in the optimal solution, it is possible to give up some rectangles whose total weight is at most $2OPT/\lambda$ and then to rearrange the others in such a way that none intersect the division lines.

A rectangle in E is said to be *in* a subframe, if it intersects the interior of the subframe but not the subframe

above it. Note that any rectangle in E is in exactly one subframe. Let us divide each subframe horizontally into λ *bands* of width q each. Define the weight of a band of a subframe to be the sum of weights of the rectangles that are in the subframe and intersect the interior of the band. Since each item intersects at most two bands, the total weight of all bands in a given subframe is at most twice the total weight of the rectangles that are in the subframe. From each subframe find a band of minimum weight, then discard from the subframe all those items that contribute to the weight of the band and insert in the freed area those rectangles whose interior intersect the bottom line of the subframe. Thus, we obtain a feasible solution to the λ-restriction problem by giving up weight at most $2OPT/\lambda$. ■

We have seen that λ-restriction is a good approximation to the SAP. Now we further show that λ-restriction itself can be well-approximated. To this end, let us reformulate it into another problem. For a given constant $k \geq 1$, a subset I of the set N of requests is said to be *k-independent* if accepting the requests in I is a feasible solution to the SAP with capacity k. For example, in the uniform case, I is 1-independent if and only if, for each pair of requests i, $j \in I$, the (discrete) time intervals $[s_i, t_i]$ and $[s_j, t_j]$ are disjoint. As before, let $w(I) = \sum_{i \in I} w_i$ be the total weight of requests in I. Given any collection $\mathcal{S}$ of k-independent sets, let $f(\mathcal{S})$ be the total weight of all the requests that are in at least one k-independent set of $\mathcal{S}$. Then

$$f(\mathcal{S}) = w(\cup_{I \in \mathcal{S}} I) = \sum_{I \in \mathcal{S}} w(I) - \sum_{i \in N} (n_i(\mathcal{S}) - 1)^+ w_i,$$

where $n_i(\mathcal{S})$ is the total number of appearances of request i in all sets of $\mathcal{S}$ and $x^+ = \max\{x, 0\}$. Let us now consider the following problem.

$$\text{(P): Maximize } f(\mathcal{S}) \text{ subject to } |S| \leq m.$$

It is easy to see that the λ-restriction of SAP is equivalent to Problem (P) with $m = K/(\lambda q)$ and $k = \lambda q$.

We claim that in Problem (P) f is a submodular set function, i.e., it satisfies the following condition: For any k-independent sets $\mathcal{S}$ and $\mathcal{T}$,

$$f(\mathcal{S} \cup \mathcal{T}) + f(\mathcal{S} \cap \mathcal{T}) \leq f(\mathcal{S}) + f(\mathcal{T}).$$

In fact, for any $i \in N$, if we let $g_i(\mathcal{S}) = (n_i(\mathcal{S}) - 1)^+ w_i$, then it is easy to check that g_i is supermodular, i.e., it satisfies the above displayed condition with f replaced by g_i and $\leq$ replaced by $\geq$. Therefore, Problem (P) is to maximize a non-decreasing (with respect to the order of set containment) submodular function subject to a cardinality constraint, which is NP-hard if f is a general submodular function (Corneujols *et al.*, 1977; Nemhauser and Wolsey, 1988). It is proved in Corneujols *et al.* (1977) that a greedy algorithm is guaranteed to deliver a solution of value at least $1 - e^{-1}$ times the optimum. Actually, this approximation result can be established significantly more easily for Problem (P) as follows.

The greedy algorithm

Suppose we have accepted j disjoint k-independent sets $I_1, \ldots, I_j$, $0 \leq j < m$. Apply the dynamic programming method of Section 2.2 to the set $N - \cup_{i=1}^{j} I_i$ of remaining requests and to the capacity k, we obtain a k-independent set I_{j+1}. A feasible solution is obtained by accepting the requests in $\cup_{i=1}^{m} I_i$.

The running time of the greedy algorithm is $O(mn(kn)^k)$. Let $p_i = w(I_i)$, $i = 1, \ldots, m$. Let $\hat{p}$ and p^* be the values of the greedy solution and of an optimal solution to Problem (P), respectively. Then we have the following lemma.

Lemma 3. $\hat{p} \geq (1 - (1 - 1/m)^m)p^* > (1 - e^{-1})p^*$.

Proof. Assume that the best solution consists of m disjoint k-independent sets $I_1^*, \ldots, I_m^*$. According to the greedy algorithm, we have

$$w(I_j) \geq \max_{1 \leq i \leq m} w\left(I_i^* \Big\backslash \bigcup_{1 \leq l \leq j-1} I_l\right) \geq \frac{1}{m} \sum_{1 \leq i \leq m} w\left(I_i^* \Big\backslash \bigcup_{1 \leq l \leq j-1} I_l\right),$$

or equivalently, using the fact that the sets I_i^* are k-independent and thus contain each element of $\cup I_l$ at most once,

$$p_j + \frac{1}{m} \sum_{l=1}^{j-1} p_l \geq \frac{1}{m} p^*,$$

for $j = 1, \ldots, m$. Let $\lambda_j = ((m-1)/m)^{m-j}$, $j = 1, \ldots, m$. Multiply the above inequality by λ_j and then sum up over all $j = 1, \ldots, m$, to get

$$\hat{p} = \sum_{i=1}^{m} p_i \geq \frac{1}{m} \left(\sum_{j=1}^{m} \lambda_j \right) p^* = \left(1 - \left(\frac{m-1}{m}\right)^m\right) p^*. \quad ■$$

Now we are ready to state the following main result.

Theorem 3. *For a SAP with bounded request sizes and for any fixed* ϵ $(0 < \epsilon < 1)$, *there is a polynomial-time approximation algorithm with performance guarantee of* $(1 - e^{-1})(1 - \epsilon)$.

Proof. Assume the sizes of requests are bounded by constant q. Our approximation algorithm works according to the magnitude of capacity K. If $K \leq 4q/\epsilon$, then we use the dynamic programming algorithm in Section 2.2 to find an optimal solution in $O(n(Kn)^K) = O(n^{4q/\epsilon+1})$ time. Now assume $K > 4q/\epsilon$ or $2r < \epsilon/2$, where $r = q/K$. We consider the λ-restriction of the problem with $\lambda = K/(mq)$, where $m = 1 + \epsilon/(4r)$. Note that for simplicity we assume without loss of generality that both λ and m are integers. As we have seen, this problem is equivalent to Problem (P) with $k = K/m$. We use the greedy algorithm to approximate an optimal solution to (P) and accept the greedy solution A as an approximate solution to the original SAP. It is evident that the running time is

$$O\left(Kn\left(\frac{K}{m}n\right)^{K/m}\right),$$

which is bounded by $O(n^{4q/\epsilon+2})$ since $K/m < 4q/\epsilon = O(1)$ and $K = O(n)$, as we mentioned at the beginning of this section. According to Lemmas 2 and 3, the ratio between the value of the approximate solution and the optimal value is at least $(1-e^{-1})(1-(2mq)/K) = (1-e^{-1})(1-2r-\epsilon/2) > (1-e^{-1})\ (1-\epsilon)$. Therefore, whatever the value of K is, we are guaranteed to get in $O(n^{4q/\epsilon+2})$ time a solution that is at least as big as $(1-e^{-1})(1-\epsilon)$ times the optimal. ■

Acknowledgement

Bo Chen gratefully acknowledges the partial support he received via a ESRC Management Research Fellowship and the support of the Academic Study Group.

References

Arkin, E.M. and Silverberg, E.B. (1987) Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, **18**, 1–8.

Bar-Noy, A., Canetti, R., Kutten, S., Mansour, Y. and Schieber, B. (1999) Bandwidth allocation with preemption. *SIAM Journal on Computing*, **28**, 1806–1828.

Bar-Noy, A., Bar-Yehuda, B., Freund, A., Naor, J. and Schieber, B. (2000) A unified approach to approximating resource allocation and scheduling, in *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, (to appear in *Journal of ACM*).

Chandra, A.K., Hirschberg, D.S. and Wong, C.K. (1976) Approximation algorithms for some generalised knapsack problems. *Theoretical Computer Science*, **3**, 293–304.

Coffman, Jr., E.G. (1983) An introduction to combinatorial models of dynamic storage allocation. *SIAM Review*, **25**, 311–325.

Corneujols, G., Fisher, M.L. and Nemhauser, G.L. (1977) Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms. *Management Science*, **23**, 789–810.

Frieze, A.M. and Clarke, M.R.B. (1984) Approximation algorithms for the *m*-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *European Journal of Operational Research*, **15**, 100–109.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

Gergov, J. (1999) Algorithms for compile-time memory optimization, in *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, pp. 907–908.

Harms, J.J. and Wong, J.W. (1995) Performance modeling of a channel reservation service. *Computer Networks and ISDN Systems*, **27**, 1487–1497.

Nemhauser, G. and Wolsey, L.A. (1988) *Integer and Combinatorial Optimization*, Wiley, New York.

Papadimitriou, C.H. and Steiglitz, K. (1982) *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

Phillips, C., Uma, R.N. and Wein, J. (2000) Off-line admission control for general scheduling problems, in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, pp. 879–888.

Robson, J.M. (1974) Bounds for some functions concerning dynamic storage allocation. *Journal of Association for Computing Machinery*, **21**, 491–499.

Szlusarek, M. (1987a) An off-line storage allocation algorithm. *Information Processing Letters*, **24**, 71–75.

Ślusarek, M. (1987b) NP-completeness of storage allocation. *Jagiellonian University Scientific Papers: Informatics*, **3**, 7–18.

Biographies

Bo Chen is a Senior Lecturer at the Warwick Business School, University of Warwick. He has a Ph.D. degree in Operations Research from Erasmus University. His current research interests include scheduling theory and applications, and combinatorial optimization. He held a 3-year Management Research Fellowship from the Economic and Social Research Council of the UK.

Refael Hassin is a Professor at the Department of Statistics and Operations Research, Tel Aviv University. He has a Ph.D. degree in Operations Research from Yale University. His main research interests include network flows, combinatorial optimization, and models of equilibrium behavior in queueing systems.

Michal Tzur is a Senior Lecturer in the Department of Industrial Engineering, Tel Aviv University. She has a Ph.D. degree in Operations Research and Management Science from Columbia University. Before joining Tel Aviv University, she spent 3 years at the Operations and Information Management Department of the Wharton School, University of Pennsylvania. Her research interests are in the areas of supply chain management, inventory management, operations scheduling and production planning.

Contributed by the Scheduling Department