Resource Allocation in Bounded Degree Trees

Reuven Bar-Yehuda[1], Michael Beder[1], Yuval Cohen[1], and Dror Rawitz[2]

[1] Department of Computer Science, Technion, Haifa 32000, Israel
{reuven@cs, sbederml@cs, scyuval@t2}.technion.ac.il
[2] Caesarea Rothschild Institute, University of Haifa, Haifa 31905, Israel
rawitz@cri.haifa.ac.il

Abstract. We study the *bandwidth allocation problem* (BAP) in bounded degree trees. In this problem we are given a tree and a set of connection requests. Each request consists of a path in the tree, a bandwidth requirement, and a weight. Our goal is to find a maximum weight subset S of requests such that, for every edge e, the total bandwidth of requests in S whose path contains e is at most 1. We also consider the *storage allocation problem* (SAP), in which it is also required that every request in the solution is given the same contiguous portion of the resource in every edge in its path. We present a deterministic approximation algorithm for BAP in bounded degree trees with ratio $(2\sqrt{e}-1)/(\sqrt{e}-1)+\varepsilon < 3.542$. Our algorithm is based on a novel application of the *local ratio technique* in which the available bandwidth is divided into narrow strips and requests with very small bandwidths are allocated in these strips. We also present a randomized $(2+\varepsilon)$-approximation algorithm for BAP in bounded degree trees. The best previously known ratio for BAP in general trees is 5. We present a reduction from SAP to BAP that works for instances where the tree is a line and the bandwidths are very small. It follows that there exists a $(2+\varepsilon)$-approximation algorithm for SAP in the line. The best previously known ratio for this problem is 7.

1 Introduction

The problems. We study the *bandwidth allocation problem* (BAP) in trees. In this problem we are given a tree $T = (V, E)$, where $m = |E|$, and a set J of n connection requests from clients. Each request j is associated with a *path* in the tree that is denoted by P_j (P_j is a set of edges), and a weight $w(j)$ that may be gained by accommodating it. It also has a bandwidth requirement, or *demand*, $d_j \in [0, 1]$. Given a parameter $\delta \in (0, 1)$, a request j is called δ-*narrow* (or simply *narrow*) if $d_j \leq \delta$. Otherwise, it is called δ-*wide* (*wide*). An instance in which all requests are narrow is called a *narrow instance*, and an instance in which all requests are wide is called a *wide instance*. A feasible solution, or a *schedule*, is a subset $S \subseteq J$ of connection requests such that, for every edge e, the total demand of requests whose path contains e is at most 1. That is, S is feasible if $\sum_{j \in S : e \in P_j} d_j \leq 1$, for every edge e. Our goal is to find a schedule with maximum total weight. When the given tree T is a line (i.e., a tree with two leaves) it is convenient to use temporal terms. In this case, the path of a request

Y. Azar and T. Erlebach (Eds.): ESA 2006, LNCS 4168, pp. 64–75, 2006.

becomes a time interval, and our goal is to find a maximum weight subset $S \subseteq J$ such that at any given time the total bandwidth is bounded by 1.

We also consider the *storage allocation problem* (SAP) which is a variation of BAP with two additional constraints: (i) the specific portion of the resource allocated to a request cannot change between edges (or over time), and (ii) the allocation must be contiguous. Hence, given a SAP instance, a solution can be described by a set of requests $S \subseteq J$ and an assignment of every request $j \in S$ to a specific portion of the resource. Formally, a solution consists of the set of requests S and a height function $h : S \rightarrow [0, 1]$ such that the following constraints are satisfied: (i) $h(j)+d_j \leq 1$ for every $j \in S$, and (ii) for every two requests $j, k \in S$ such that $j \neq k$ and $P_j \cap P_k \neq \emptyset$ either $h(j) + d_j \leq h(k)$ or $h(k) + d_k \leq h(j)$. That is, we require that the portion of the resource assigned to j is within the range $[0, 1]$, and that no two requests occupy the same portion of the resource on the same edge (or at the same time). Observe that a feasible SAP schedule is a feasible BAP schedule, while the converse may not be true. Hence, the BAP optimum is at least as large as the SAP optimum of a given problem instance.

In the line topology a request j can be represented by an axis-parallel rectangle, whose length is $|P_j|$ (or the duration of the request, in temporal terms), and whose height is d_j. The rectangles are allowed to move vertically but not horizontally. We wish to select a maximum weight subset of rectangles that can be placed within a strip of height 1 such that no two rectangles overlap. A natural application of SAP in the line arises in a multi-threaded environment, where threads require contiguous memory allocations for fixed time intervals.

Related Work. Both BAP and SAP are NP-hard even in the line since they contain *knapsack* as the special case in which the paths of all requests share an edge. BAP in the line with unit demands is the problem of finding an independent set in a weighted interval graph which is solvable in polynomial time (see, e.g., [1]). BAP in the tree with unit demands is the problem of finding a maximum weight independent set of paths in a tree. This problem is also solvable in polynomial time [2]. Notice that BAP and SAP are equivalent in the case of unit demands.

The special case of BAP in the line, where all requests have the same length (or duration), was studied by Arkin and Silverberg [3]. Bar-Noy et al. [4] considered an online version of BAP in which the weight of a request is proportional to the area of the rectangle it induces. Phillips et al. [5] developed a 6-approximation algorithm for BAP in the line. Leonardi at al. [6] observed that SAP in the line can be used to model the problem of scheduling requests for remote medical consulting on a shared satellite channel. They obtained a 12-approximation algorithm for SAP in the line. Bar-Noy et al. [7] used the local ratio technique to improve the ratios for BAP and SAP in the line to 3 and 7, respectively.

Chen et al. [8] studied the special cases of BAP and SAP, where all demands are multiples of $1/K$ for some integer K. They developed dynamic programming algorithms for both problems that compute optimal solutions for wide instances. Their algorithm for BAP easily extends to general wide instances of BAP. However, in the case of SAP, the running time of the algorithm depends on K that may be exponential in the input size. They presented an approximation algorithm for BAP with proportional weights. They also presented an approximation algorithm with ratio $\frac{e}{e-1} + \varepsilon$, for any $\varepsilon > 0$, for a special case of SAP in which $d_j = i/K$ for some $i \in \{1, \ldots, q\}$ where q is a constant.

Calinescu et al. [9] developed a randomized approximation algorithm for BAP in the line with expected performance ratio of $2 + \varepsilon$, for every $\varepsilon > 0$. They obtained their results by dividing the given instance into a wide instance and a narrow instance. They use dynamic programming to compute an optimal solution for the wide instance, and a randomized LP-based algorithm to obtain a $(1 + \varepsilon)$-approximate solution for the narrow instance. They also present a 3-approximation algorithm for BAP that is different from the one from [7].

The more general version of BAP in which the edge capacities are not uniform is called the *unsplittable flow problem* (UFP). Chakrabarti et al. [10] presented the first $O(1)$-approximation algorithm for UFP in the line by extending the approach of [9] to the non-uniform capacity case. However, their 13-approximation algorithm works under the *no-bottleneck assumption* which states that the maximum demand is not larger than the minimum edge capacity. Chekuri et al. [11] used an LP-based deterministic algorithm instead of a randomized algorithm to obtain a $(2 + \varepsilon)$-approximation algorithm for UFP in the line and a 48-approximation algorithm for UFP in trees both under the no-bottleneck assumption.

Lewin-Eytan et al. [12] studied the *admission control problem* in the tree topology. The problem instance in this case is similar to a BAP-instance. However, each request is also associated with a time interval. A feasible schedule is a set of connection requests such that at any given time, the total bandwidth requirement on every edge in the tree is at most 1. The goal is to find a feasible schedule with maximum total weight. Clearly, the admission control problem in trees is a generalization of BAP. Lewin-Eytan et al. [12] presented a divide and conquer $(5 \log n)$-approximation algorithm for admission control in trees. It divides the set of requests using the temporal dimension, and conquers a set of requests whose time intervals overlap using a local ratio 5-approximation algorithm. This is in fact a 5-approximation algorithm for BAP in the general tree topology.

Our results. We consider BAP in the tree topology where the maximum degree of a vertex in the given tree is a constant. For wide instances of BAP we provide a polynomial time dynamic programming algorithm that extends the algorithms from [8, 9]. We present an approximation algorithm for narrow instances of BAP in general trees (respectively, in lines) with ratio $\frac{\sqrt{e}}{\sqrt{e}-1} + \varepsilon$ (respectively, $\frac{e}{e-1} + \varepsilon$), for every $\varepsilon > 0$. This algorithm is based on a novel application of the *local ratio technique* in which the available bandwidth is divided into narrow strips, and requests with very small demands are allocated in these strips. By combining the two algorithms we get an approximation algorithm for BAP in bounded degree trees with ratio $\frac{2\sqrt{e}-1}{\sqrt{e}-1} + \varepsilon < 3.542$ (respectively, $\frac{2e-1}{e-1} + \varepsilon < 2.582$). We also present a randomized $(1 + \varepsilon)$-approximation algorithm, for every $\varepsilon > 0$, for narrow instances of BAP in bounded degree trees that extends the $(1 + \varepsilon)$-approximation algorithm for BAP in the line from [9]. This implies a randomized $(2 + \varepsilon)$-approximation algorithm for BAP in bounded degree trees.

For wide instances of SAP in bounded degree trees we provide a polynomial time dynamic programming algorithm that extends the algorithm from [8]. We present a reduction from SAP to BAP that works on very narrow instances in the line topology. The reduction is based on an algorithm for the *dynamic storage allocation problem* by Buchsbaum et al. [13]. This reduction implies a $(2+\varepsilon)$-approximation algorithm for SAP in the line.

2 Preliminaries

Definitions and Notation. We denote the optimum of a given problem instance by OPT. Given a schedule S, we denote by $w(S)$ the total weight of S, i.e., $w(S) = \sum_{j \in S} w(j)$.

Throughout the paper we assume that the given tree T is rooted, and we denote the root by r. We also assume that the maximum degree of a vertex in T is a constant. We denote the maximum degree by Δ, i.e., $\Delta = \max_u \deg(u)$.

The *peak* of a request j is the vertex in jth path that is closest to the root r. We denote the peak of j by peak(j). We denote by $E(j)$ the set of edges in P_j that are incident on peak(j). $E(j)$ contains either two edges or one edge. We define a partial order on the requests as follows. For requests j and ℓ we write $j \prec \ell$ if peak(j) is an ancestor of peak(ℓ). We denote by $A(\ell)$ the set of requests j such that peak(j) is an ancestor of peak(ℓ), i.e., $A(\ell) = \{j : j \prec \ell\}$.

Henceforth, we assume that the requests are topologically ordered according to the partial order. That is, we assume that if $j < k$, then $k \not\prec j$. In other words, $j < k$ if peak(k) is not found on the path for peak(j) to the root r.

Observation 1. *Let ℓ be a request, and let $S \subseteq J$ be a feasible solution such that $\ell \not\prec j$ for every $j \in S$. Then, $S \cup \{\ell\}$ is a feasible solution if the load on e is at most $1 - d_\ell$ for every $e \in E(\ell)$.*

Narrow and Wide Instances. Given a parameter $\delta \in (0,1)$, we can divide a given instance into a narrow instance and a wide instance. We denote the corresponding sets of requests by R_N and R_W, respectively.

Lemma 1. *Let S_N and S_W be an r_1-approximate solution with respect to R_N and a r_2-approximate solution with respect to R_W, respectively. Then, the solution of greater weight is an $(r_1 + r_2)$-approximation for the original instance.*

Proof. Let S^* be an optimal solution for the original instance. Either $w(S^* \cap R_N) \geq \frac{r_1}{r_1+r_2} w(S^*)$ or $w(S^* \cap R_W) \geq \frac{r_2}{r_1+r_2} w(S^*)$. Hence, either $w(S_N) \geq \frac{1}{r_1} \frac{r_1}{r_1+r_2} w(S^*) = \frac{1}{r_1+r_2} w(S^*)$ or $w(S_W) \geq \frac{1}{r_2} \frac{r_2}{r_1+r_2} w(S^*) = \frac{1}{r_1+r_2} w(S^*)$. □

The Local Ratio Technique. The local ratio technique [14, 15, 16, 7] is based on the Local Ratio Theorem, which applies to optimization problems of the following type. The input is a non-negative weight vector $w \in \mathbb{R}^n$ and a set of feasibility constraints $\mathcal{F}$. The problem is to find a solution vector $x \in \mathbb{R}^n$ that maximizes (or minimizes) the inner product $w \cdot x$ subject to the constraints $\mathcal{F}$.

Theorem 1 (Local Ratio [7]). *Let $\mathcal{F}$ be a set of constraints and let w, w_1, and w_2 be weight vectors such that $w = w_1 + w_2$. Then, if x is r-approximate both with respect to $(\mathcal{F}, w_1)$ and with respect to $(\mathcal{F}, w_2)$, for some r, then x is also an r-approximate solution with respect to $(\mathcal{F}, w)$.*

3 Bandwidth Allocation

In this section we consider the bandwidth allocation problem in bounded degree trees. For the special case of wide instances we present a polynomial time dynamic programming algorithm that computes optimal solutions. For the special case of narrow instances we present a deterministic approximation algorithm whose ratio is $1/(1 - 1/\sqrt{e} - \varepsilon) < 2.542$. We note that the algorithm for narrow instances works even in the case of general trees. In the line topology the approximation ratio of this algorithm is $1/(1 - 1/e - \varepsilon) < 1.582$. By Lemma 1 it follows that there is a 3.542-approximation algorithm for BAP in bounded degree trees, and a 2.582-approximation algorithm for BAP in the line topology. For narrow instances we also present a randomized LP-based $(1 + \varepsilon)$-approximation algorithm that extends the $(1 + \varepsilon)$-approximation algorithm by Calinescu et al. [9] for narrow instances of BAP in the line. Using the dynamic programming algorithm from Section 3 it follows from Lemma 1 that there is a randomized $(2 + \varepsilon)$-approximation algorithm for BAP on bounded degree trees.

3.1 Dynamic Programming Algorithm for Wide Instances

We present a polynomial time dynamic programming algorithm for BAP on wide instances and in bounded degree trees. This algorithm extends the algorithms for BAP in the line topology by Chen et al. [8] and by Calinescu et al. [9].

In order to solve the problem on wide instances we consider a variation of BAP in which we are given a constant L that limits the number of requests per edge. We present a dynamic programming algorithm whose running time is $O(m \cdot n^{\Delta \cdot L})$. Clearly, if S is a feasible solution for a wide instance, then there are at most $1/\delta$ requests in S that go through e for any edge e. Hence, the running time of this algorithm is $O(m \cdot n^{\Delta/\delta})$.

We use the following notation. Consider a vertex u_0 in the tree. Let T_0 be the subtree whose root is u_0, and let e_0 the edge that is going from u_0 to its parent. u_0's children are denote by $u_1, \ldots, u_k$. Also, let e_i be the edge connecting u_i and u_0 for $i \in \{1, \ldots, k\}$. See example in Figure 1. Using this notation, we refer to a set of requests S_i as *proper with respect to a vertex* u_i if (1) $e_i \in P_j$ for every $j \in S_i$, (2) $\sum_{j \in S_i : e_i \in P_j} d_j \leq 1$, and (3) $|S_i| \leq L$. Given a proper set S_0 with respect to u_0, the sets $S_1, \ldots, S_k$ are said to be *compatible* with S_0 if (1) S_i is proper with respect to u_i for every i, and (2) for every j and $i, i' \in \{0, \ldots, k\}$, if $e_i, e_{i'} \in P_j$, then either $j \in S_i, S_{i'}$ or $j \notin S_i, S_{i'}$.

The dynamic programming table is of size $O(m \cdot n^L)$, and it is defined as follows. For a vertex u_0 and a set of requests S_0 that is proper with respect to u_0, the state $\Pi(u_0, S_0)$ is the maximum weight of a set $S' \subseteq J(T_0)$, where $J(T_0)$

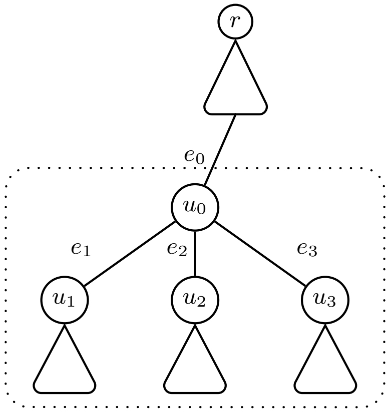

Fig. 1. u_0 and its children; T_0 is marked by the dotted line

contains requests j such that P_j is fully contained in T_0, such that $S_0 \cup S'$ is feasible. We initialize the table by setting $\Pi(u_0, S_0) = 0$ for every leaf u_0 and a proper set S_0. We compute the rest of the entries by using:

$$\Pi(u_0, S_0) = \max_{S_1,\ldots,S_k \text{ are compatible with } S_0} \left\{ w(\cup_{i=1}^{k} S_k \setminus S_0) + \sum_{i=1}^{k} \Pi(u_i, S_i) \right\}$$

when u_0 is an internal node. The weight of an optimal solution is $\Pi(r, \emptyset)$.

We show that the running time of the dynamic programming algorithm is $O(m \cdot n^{L \cdot \Delta})$. To compute each entry $\Pi(u_0, S_0)$ we need to go through all the possibilities of sets $S_1, \ldots, S_k$ that are compatible with S_0. There are no more than $\sum_{i=1}^{L} \binom{n}{i} = O(n^L)$ possibilities of choosing a proper set S_i that is compatible with $S_0, \ldots, S_{i-1}$. Hence, the number of possibilities is $O(n^{L \cdot (\Delta - 1)})$. Hence, the total running time is $O(m \cdot n^L n^{L \cdot (\Delta-1)}) = O(m \cdot n^{L \cdot \Delta})$.

Note that the computation of $\Pi(u_0, S_0)$ can be modified so as to compute a corresponding solution. This can be done by keeping track on which option was taken in the recursive computation. Afterwards an optimal solution can be reconstructed in a top down manner.

3.2 Local Ratio Algorithm for Narrow Instances

In this section we consider the special case of narrow instances. For this case we present a deterministic approximation algorithm whose ratio is $1/(1-1/\sqrt{e}-\varepsilon) < 2.542$. In the line topology the ratio is $1/(1-1/e-\varepsilon) < 1.582$.

Scheduling Requests in Layers. Throughout this section we assume that all the requests in the given instance are δ-narrow for some small constant $\delta > 0$. Let α be a constant such that $\delta < \alpha \leq 1$. We assume that α is significantly larger than δ. In this section we show how to construct an approximate solution that uses at most α of the capacity of every edge in the given tree. Henceforth we refer to a tree whose edges has capacity α as an *α-layer*, or simply a *layer*. (Note that, in general, α may be larger than one.) Using these terms, in this section we present an algorithm that computes a solution that resides in an α-layer. We note that the approximation ratio of the algorithm is with respect to the original problem in which the capacity of the edges is 1.

Algorithm **Layer** is a local ratio algorithm that computes a $(1 + 2/(\alpha - \delta))$-approximate solution S such that the total demand of requests in S on any edge is at most α. Algorithm **Layer** is recursive and works as follows. If there are no requests, then it returns $\emptyset$. Otherwise, it chooses a request ℓ such that $\ell \not\prec j$ for every $j \neq \ell$. This can be done by choosing a request whose peak is furthest away from the root. It constructs a new weight function w_1, and solves the problem recursively on $w_2 = w - w_1$ and the set of jobs with positive weight that is denoted by J^+. Note that $\ell \notin J^+$. Then, it adds ℓ to the solution that was computed recursively only if feasibility is maintained.

Algorithm 1. Layer(J, w)

1: **if** $J = \emptyset$ **then** return $\emptyset$
2: Let $\ell \in J$ be a request such that $\ell \not\prec j$ for every $j \in J \setminus \{\ell\}$
3: Define $w_1(j) = w(\ell) \cdot \begin{cases} 1 & j = \ell, \\ \frac{d_j}{\alpha-\delta} & j \neq \ell, P_j \cap P_\ell \neq \emptyset, \\ 0 & \text{otherwise,} \end{cases}$ and $w_2 = w - w_1$
4: Let J^+ be the set of positive weighted requests
5: $S' \leftarrow$ **Layer**(J^+, w_2)
6: **if** $\sum_{j \in S' : e \in P_j} d_j \leq \alpha - d_\ell$ for every $e \in E(\ell)$ **then** $S \leftarrow S' \cup \{\ell\}$
7: **else** $S \leftarrow S'$
8: return S

Observe that due to Lines 6–7 and Obs. 1 the total demand of requests from the computed solution on any edge is at most α. Also, the running time of the algorithm is clearly polynomial, since the number of recursive calls is at most n. In fact, using similar arguments to those used in [7] it can be implemented to run in $O(n \log n)$ time.

Lemma 2. *Algorithm **Layer** computes a $(1+\frac{2}{\alpha-\delta})$-approximate solution S that resides within an α-layer.*

Proof. The proof is by induction on the number of recursive calls. In the base case ($J = \emptyset$) the computed solution is optimal. For the inductive step, we assume that $w(S')$ is $(1 + \frac{2}{\alpha-\delta})$-approximate with respect to J^+ and w_2. If this is the case, then S is also $(1 + \frac{2}{\alpha-\delta})$-approximate with respect to J and w_2, since $w_2(\ell) = 0$. We show that S is also $(1 + \frac{2}{\alpha-\delta})$-approximate with respect to J and w_1. Due to Lines 6-7 either $\ell \in S$ or $S \cup \{\ell\}$ is infeasible. If $\ell \in S$, then $w_1(S) \geq w(\ell)$. Otherwise, $\sum_{j \in S' : e \in P_j} d_j > \alpha - d_\ell$ for some edge $e \in E(\ell)$, and therefore $w_1(S) \geq w(\ell) \cdot \frac{\alpha - d_\ell}{\alpha - \delta} \geq w(\ell)$. On the other hand, we show that $w_1(T) \leq w(\ell) \cdot (1 + 2/(\alpha - \delta))$, for every solution T. Let $j \in T$ be a request whose path intersects P_ℓ. Since $\ell \not\prec j$ for every $j \in J$ either peak(j) is an ancestor of peak(ℓ) or $peak(j) = peak(\ell)$. Hence, P_j contains at least one edge from $E(\ell)$. It follows that

$$w_1(T) \leq w(\ell) \cdot \max\{1 + 2(1 - d_\ell)/(\alpha - \delta), 2/(\alpha - \delta)\} \leq w(\ell) \cdot (1 + 2/(\alpha - \delta))$$

which means that S is $(1 + \frac{2}{\alpha - \delta})$-approximate with respect to J and w_1. This completes the proof since by the Local Ratio Theorem we get that S is $(1 + \frac{2}{\alpha - \delta})$-approximate with respect to J and w as well. □

When the given tree is a line we may choose one of the leafs to be the root, and in this case $|E(\ell)| = 1$. Hence, in Lemma 2, it can be shown that S is $(1 + \frac{1}{\alpha - \delta})$-approximate with respect to J and w_1. It follows that Algorithm **Layer** computes $(1 + 1/(\alpha - \delta))$-approximate solutions that reside within an α-layer.

Iterative Approximation in Layers. Algorithm **Multi-Layer** iteratively use Algorithm **Layer** with $\alpha = 1/k$ to schedule requests in $1/k$-layers for some large constant k, such that δ is significantly smaller than $1/k$.

Algorithm 2. Multi-Layer(J, w)

$J_1 \leftarrow J$
for $i = 1$ to k **do**
  $S_i \leftarrow$ **Layer**(J_i, w)
  $J_{i+1} \leftarrow J_i \setminus S_i$
end for
Return $\cup_{i=1}^{k} S_i$

Algorithm **Multi-Layer** computes feasible solutions, since Algorithm **Layer** computes feasible solutions each residing in a $1/k$-layer. The running time is polynomial, since Algorithm **Layer** is invoked a constant number of times.

We use the following notation. Let S^* be an optimal solution to the original instance. Let OPT_i denote the optimal value with respect to J_i. We denote $F_i = \cup_{l=1}^{i} S_l$. Hence, F_k is the computed solution. We also define $r \triangleq 1 + \frac{2}{1/k - \delta}$.

Lemma 3. $w(F_k) \geq (1 - (1 - 1/r)^k) \cdot \text{OPT}$.

Proof. We show that $w(F_i) \geq (1 - (1 - 1/r)^i) \cdot \text{OPT}$ for every i. We prove the claim by induction on i. The base case $(i = 0)$ is trivial. For the inductive step, we assume that $w(F_{i-1}) \geq (1 - (1 - 1/r)^{i-1}) \cdot \text{OPT}$. The set $S^* \setminus F_{i-1}$ is feasible with respect to $J_i = J \setminus F_{i-1}$, and therefore, $\text{OPT}_i \geq w(S^* \setminus F_{i-1})$. By Lemma 2 it follows that S_i is r-approximate with respect to J_i. Hence,

$$w(S_i) \geq \frac{\text{OPT}_i}{r} \geq \frac{w(S^* \setminus F_{i-1})}{r} \geq \frac{w(S^*) - w(F_{i-1})}{r} = \frac{\text{OPT} - w(F_{i-1})}{r} .$$

It follows that $w(F_i) = w(F_{i-1}) + w(S_i) \geq (1-1/r) \cdot w(F_{i-1}) + \text{OPT}/r$. Putting it together with the induction hypothesis we get that

$$w(F_i) \geq (1-1/r) \cdot (1-(1-1/r)^{i-1}) \cdot \text{OPT} + \text{OPT}/r = \text{OPT} \cdot (1-(1-1/r)^i)$$

and the lemma follows. □

If δ is significantly smaller than $1/k$ it follows that $\lim_{k\to\infty,\delta\to 0}(1-1/r(\delta,k))^k) = 1/\sqrt{e}$. Hence, we may choose a sufficiently large constant k and then a sufficiently small constant δ such that $(1-1/r)^k \leq 1/\sqrt{e}+\varepsilon$. Hence, for every constant $\varepsilon > 0$ there exist $\delta > 0$ and k such that Algorithm **Multi-Layer** computes solutions that are $(1/(1-1/\sqrt{e}-\varepsilon))$-approximate. In the line we get a ratio of $1/(1-1/e-\varepsilon)$ by setting $r \triangleq 1 + \frac{1}{1/k-\delta}$.

3.3 Randomized Algorithm for Narrow Instances

In this section we present a randomized LP-based $(1+\varepsilon)$-approximation algorithm that extends the $(1+\varepsilon)$-approximation algorithm by Calinescu et al. [9] for BAP in the line topology.

BAP can be formalized as follows:

$$\begin{array}{lll} \max & \sum_{j\in R} w(j)x_j & \\ \text{s.t.} & \sum_{j\,:\,e\in P_j} d_j x_j \leq 1 & \forall e \in E \\ & x_j \in \{0,1\} & \forall j \in J \end{array} \qquad \text{(IP-BAP)}$$

The LP-relaxation of IP-BAP is obtained by replacing the integrality constraints by: $0 \leq x_j \leq 1$ for every $j \in J$, and is denoted by LP-BAP.

Next, we present a randomized approximation algorithm for narrow instances of BAP. Specifically, we show that for every $\varepsilon < 1/6$ there exists $\delta > 0$ small enough such that the algorithm computes $1/(1-6\varepsilon)$-approximate solutions. The approximation algorithm is described as follows. First, we solve LP-BAP. Denote by x^* the computed optimal solution, and let $\text{OPT}^* = \sum_{j\in J} w(j)x_j^*$. We choose independently at random the variables $Y_j \in \{0,1\}$, for $j \in J$, where $\Pr[Y_j = 1] = (1-\varepsilon)x_j^*$. Next we define the random variables $Z_j \in \{0,1\}$, $j \in J$. The Z_js are considered in a top down manner. That is, Z_j is defined only after Z_ℓ was defined for every $\ell \prec j$. The Z_js are defined as follows:

$$Z_j = \begin{cases} 1 & \text{if } Y_j = 1 \text{ and } \sum_{i\,:\,Z_i=1\wedge e\in P_i} d_i \leq 1 - d_j \text{ for every } e \in E(j), \\ 0 & \text{otherwise.} \end{cases}$$

Notice that the Z_js are dependent, and they can be computed in the order $Z_1,\ldots,Z_n$ since if $i < j$ then $j \notin A(i)$ (or, $j \not\prec i$).

Let $Z = \{j : Z_j = 1\}$. Z is a feasible solution, and $\text{E}[w(Z)] = \sum_{j\in J} w(j) \cdot \Pr[Z_j = 1]$. In the full version of the paper we show that $\Pr[Z_j = 1] \geq (1-3\varepsilon)x_i^*$. It follows that $\text{E}[w(Z)] \geq \sum_{j\in J} w(j) \cdot (1-3\varepsilon)x_j^* = (1-3\varepsilon)\text{OPT}^*$. Furthermore, Markov's inequality implies that $\Pr[w(Z) \geq (1-6\varepsilon) \cdot \text{OPT}^*] \geq 1/2$. Hence, we can use repetition in order to amplify the probability of obtaining at least a weight of $(1-6\varepsilon) \cdot \text{OPT}^*$.

4 Storage Allocation

In this section we consider SAP in the line. For the special case of wide instances we present polynomial time dynamic programming algorithm that computes optimal solutions. Note that this algorithm works even in the case of bounded degree tree. For narrow instances of SAP in the line we present a general reduction from SAP to BAP. That is, given a narrow SAP instance in the line we show how to find an approximate solution using an algorithm for BAP. Thus, using the $(1+\varepsilon)$-approximation algorithm for narrow instances of BAP in the line from [11] we may obtain a $(1+\varepsilon)$-approximation algorithm for narrow instances of SAP. By Lemma 1, there exists a $(2+\varepsilon)$-approximation algorithm for SAP, for any $\varepsilon > 0$.

4.1 Storage Allocation on Wide Instances

We show how to extend the dynamic programming algorithm from Sect. 3.1 to solve wide instances of SAP in bounded degree trees. The running time of the modified algorithm is $O(m \cdot n^{2L\cdot\Delta})$, where L is an upper bound on the number of requests per edge. If S is a feasible solution for a wide instance of SAP, then there are at most $1/\delta$ requests in S that go through e for any edge e. Hence, the running time of this algorithm in the line topology is $O(m \cdot n^{4/\delta})$, where m is the number of edges in the line.

Our algorithm is based on the following simple observation.

Observation 2. *There exists an optimal solution such that, for every request j, either $h(j) = 0$ or there exists a request $j' \neq j$ such that $P_j \cap P_{j'} \neq \emptyset$ and $h(j) = h(j') + d_{j'}$.*

Proof. Given an optimal solution S^*, simply apply "gravity" on S^*. □

Let S^* be an optimal solution. By Obs. 2 we may assume that the height $h(j)$ of every request j is the sum of demands of some (possibly empty) subset of requests. Since S^* contains at most L requests per edge, the number of possible heights is bounded by $\sum_{i=0}^{L} \binom{n}{i} = O(n^L)$. Let H be the set of possible heights. In the full version of the paper we extend the definition of a proper set (see Sect. 3.1) to a *proper pair* (S_i, h_i), where h_i is a height function. Since H is of polynomial size, the number of possible proper pairs with respect to some vertex is polynomial as well. The rest of the details are given in the full version.

4.2 Reduction for Narrow Instances

Our reduction relies on a closely related problem to SAP called the *dynamic storage allocation problem* (DSA). Similarly to SAP in the line, in DSA we are given a set of rectangles that can only move vertically. The goal is to minimize the total height required to pack all rectangles such that no two rectangles overlap. Formally, a DSA solution is an assignment $h : S \to \mathbb{R}^+$ such that for every $j \neq k$ and $P_j \cap P_k \neq \emptyset$ either $h(j) + d_j \leq h(k)$ or $h(k) + d_k \leq h(j)$. Our goal is to minimize $\max_{j \in J} \{h(j) + d_j\}$.

We use the following result for DSA. Buchsbaum et al. [13] presented a polynomial time algorithm that computes a solution whose cost is at most $(1+O((\frac{D}{\text{LOAD}})^{1/7}))\cdot\text{LOAD}$, where $D = \max_j \{d_j\}$, and $\text{LOAD} = \max_e\{\sum_{j:e\in P_j} d_j\}$. Observe that when $D = o(\text{LOAD})$ the cost of the solution is $(1 + o(1)) \cdot \text{LOAD}$.

Lemma 4. *For every constant $\beta > 0$ there exists $\delta > 0$ such that if S is a feasible* BAP *solution to some δ-narrow instance, then S can be transformed into a* SAP *solution that fits into a $(1 + \beta)$-layer in polynomial time.*

Proof. Let S' be the solution computed by algorithm of Buchsbaum et al. [13]. It follows that $\text{LOAD}(S') \leq \text{LOAD}(S)+C\cdot D^{1/7}\cdot\text{LOAD}(S)^{6/7}$ for some constant C. Since $\text{LOAD}(S) \leq 1$, there exists δ small enough such that $\text{LOAD}(S') \leq 1+\beta$. □

Our reduction uses the notion of an α-layer. Recall that an α-layer is a tree (or a line) in which the capacity of the edges is α. In the case of BAP, this means that if a solution fits into an α-layer then the total demand on every edge is at most α. In the case of SAP, such a solution S must satisfy the following constraints: (i) $h(j) + d_j \leq \alpha$ for every $j \in S$, and (ii) for every two requests $j, k \in S$ such that $j \neq k$ and $P_j \cap P_k \neq \emptyset$ either $h(j) + d_j \leq h(k)$ or $h(k) + d_k \leq h(j)$.

Let Algorithm **BAP** be an approximation algorithm for narrow BAP instances in the line such that for every $\varepsilon' > 0$ there exists $\delta > 0$ such that the algorithm computes $r/(1-\varepsilon')$-approximate solutions. Algorithm **SAP** is our approximation algorithm for narrow SAP instances in the line that uses Algorithm **BAP**. We show that for every $\varepsilon > 0$ there exists $\delta > 0$ such that it computes $r/(1 - \varepsilon)$-approximate solutions for δ-narrow instances. We assume that δ is small enough such that (i) Algorithm **BAP** computes $r/(1-\varepsilon/4)$-approximate solutions on δ-narrow instances, and (ii) the conditions of Lemma 4 are satisfied with $\beta = \varepsilon/4$. We also assume that $\delta < \varepsilon/4$. Algorithm **SAP** starts by calling Algorithm **BAP** in order to obtain a BAP solution. Using Lemma 4, it transforms this solution into a SAP solution that fits into a $(1+\beta)$-layer, where $\beta = \varepsilon/4$. Then, it removes a small part of it in order to obtain a feasible solution for SAP.

Algorithm 3. SAP(J, w)

$S \leftarrow \textbf{BAP}(J, w)$
Compute an assignment h for S in a $(1 + \beta)$-layer using Lemma 4
Divide the $(1 + \beta)$-layer into β-layers
Let S_i be the requests that intersect the ith layer
$k \leftarrow \text{argmin}_i w(S_i)$
$S' \leftarrow S \setminus S_k$
For $j \in S'$, let $h'(j) = \begin{cases} h(j) & h(j) < (k-1)\beta, \\ h(j) - \beta & h(j) \geq k\beta, \end{cases}$
Return (S', h')

The computed solution is feasible since the removal of S_k leaves one β-layer empty, and this allows us to condense the assignment h such that the remaining requests fit in a layer of height 1. Furthermore, since $\delta < \varepsilon/4$ each request j can be contained in at most two β-layers. Hence, $\sum_i w(S_i) \leq 2w(S)$. It follows that $w(S_k) \leq 2w(S)/\lceil 1/\beta \rceil = 2w(S)/\lceil 4/\varepsilon \rceil < 3\varepsilon \cdot w(S)/4$ and therefore $w(S') > (1 - 3\varepsilon/4) \cdot w(S)$. Since $w(S) \geq \textsc{opt} \cdot (1 - \varepsilon/4)/r$, it follows that $w(S') > (1 - 3\varepsilon/4)(1 - \varepsilon/4)/r \cdot \textsc{opt} \geq (1 - \varepsilon)/r \cdot \textsc{opt}$ as required. (Recall that the BAP optimum is at least as large as the SAP optimum.) Finally, the running time is polynomial, because given ε all parameters except n are constants.

References

1. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press (1980)
2. Tarjan, R.E.: Decomposition by clique separators. Discrete Mathematics **55** (1985) 221–232
3. Arkin, E.M., Silverberg, E.B.: Scheduling jobs with fixed start and end times. Discrete Applied Mathematics **18** (1987) 1–8
4. Bar-Noy, A., Canetti, R., Kutten, S., Mansour, Y., Schieber, B.: Bandwidth allocation with preemption. SIAM J. Comp. **28** (1999) 1806–1828
5. Phillips, C., Uma, R.N., Wein, J.: Off-line admission control for general scheduling problems. Journal of Scheduling **3** (2000) 365–381
6. Leonardi, S., Marchetti-Spaccamela, A., Vitaletti, A.: Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In: 20th Conference on Foundations of Software Technology and Theoretical Computer Science. Volume 1974 of LNCS. (2000) 409–420
7. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Shieber, B.: A unified approach to approximating resource allocation and schedualing. J. ACM **48** (2001) 1069–1090
8. Chen, B., Hassin, R., Tzur, M.: Allocation of bandwidth and storage. IIE Transactions **34** (2002) 501–507
9. Calinescu, G., Chakrabarti, A., Karloff, H.J., Rabani, Y.: Improved approximation algorithms for resource allocation. In: 9th International Integer Programming and Combinatorial Optimization Conference. Volume 2337 of LNCS. (2002) 401–414
10. Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. In: 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems. Volume 2462 of LNCS. (2002) 51–66
11. Chekuri, C., Mydlarz, M., Shepherd, B.: Multicommodity demand flow in a tree. In: 30th Annual International Colloquium on Automata, Languages and Programming. Volume 2719 of LNCS. (2003) 410–425
12. Lewin-Eytan, L., Naor, J., Orda, A.: Admission control in networks with advance reservations. Algorithmica **40** (2004) 293–403
13. Buchsbaum, A.L., Karloff, H., Kenyon, C., Reingold, N., Thorup, M.: OPT versus LOAD in dynamic storage allocation. SIAM J. Comp. **33** (2004) 632–646
14. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. Annals of Discrete Mathematics **25** (1985) 27–46
15. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J. Disc. Math. **12** (1999) 289–297
16. Bar-Yehuda, R.: One for the price of two: A unified approach for approximating covering problems. Algorithmica **27** (2000) 131–144