Improved Approximation Algorithms for Resource Allocation

Gruia Calinescu[1], Amit Chakrabarti[2⋆], Howard Karloff[3], and Yuval Rabani[4⋆⋆]

[1] Department of Computer Science, Illinois Institute of Technology, Stuart Building, 10 West 31st Street, Chicago, IL 60616.
E-mail: calinesc@cs.iit.edu
[2] Department of Computer Science, Princeton University, Princeton, NJ 08544.
E-mail: amitc@cs.princeton.edu
[3] AT&T Labs — Research, 180 Park Ave., Florham Park, NJ 07932.
E-mail: howard@research.att.com
[4] Computer Science Department, Technion — IIT, Haifa 32000, Israel.
Email: rabani@cs.technion.ac.il

Abstract. We study the problem of finding a most profitable subset of n given tasks, each with a given start and finish time as well as profit and resource requirement, that at no time exceeds the quantity B of available resource. We show that this NP-hard Resource Allocation problem can be $(1/2-\varepsilon)$-approximated in polynomial time, which improves upon earlier approximation results for this problem, the best previously published result being a 1/4-approximation. We also give a simpler and faster 1/3-approximation algorithm.

1 Introduction

We consider the following optimization problem. Suppose we have a limited supply of one reusable resource and are given a set of n tasks each of which occupies a fixed interval of time and requires a given amount of the resource. Further, each task has an associated profit which we obtain if we schedule that task. Our goal is to select a subset of the n tasks to schedule, so that the resource available is sufficient at all times for all simultaneously scheduled tasks, and so that the profit obtained is maximized. Let us call this problem the Resource Allocation Problem or rap for short.

⋆ Part of this work was done while visiting AT&T Labs — Research. Work at Princeton University supported by NSF Grant CCR-99817 and ARO Grant DAAH04-96-1-0181.

⋆⋆ Part of this work was done while visiting AT&T Labs — Research. Work at the Technion supported by Israel Science Foundation grant number 386/99, by BSF grants 96-00402 and 99-00217, by Ministry of Science contract number 9480198, by EU contract number 14084 (APPOL), by the CONSIST consortium (through the MAGNET program of the Ministry of Trade and Industry), and by the Fund for the Promotion of Research at the Technion.

This abstractly specified problem actually occurs in several concrete guises. For instance, we may be given a set of network sessions, with fixed start and finish times, that compete for a limited amount of bandwidth between two fixed endpoints in the network. Leonardi et al. [6] observe that a research project on scheduling requests for remote medical consulting on a satellite channel requires solving a slight variant of this very problem. Hall and Magazine [5] were interested in maximizing the value of a space mission by deciding what set of projects to schedule during the mission, where an individual project typically occupies only a part of the mission. Due to its various guises, RAP is known by several other names such as BANDWIDTH ALLOCATION PROBLEM, RESOURCE CONSTRAINED SCHEDULING and CALL ADMISSION CONTROL.

To formulate the problem precisely, we are given an integer B (the total available amount of the shared resource) and a collection of n tasks, with starting times s_i, finishing times t_i, resource requirements b_i, where $b_i \leq B$, and profits p_i. All these numbers are nonnegative integers and $s_i < t_i$ for all i. We want to identify a subset $S \subseteq [n]$ of tasks to schedule which maximizes $\sum_{i \in S} p_i$ among those S satisfying the following constraint:

$$\forall t \quad \sum_{i:\, [s_i, t_i) \ni t} b_i \;\leq\; B \;.$$

RAP is a natural generalization of the (polynomial-time solvable) problem of finding a maximum weight independent set in a weighted interval graph.

1.1 Prior Work

There has been a flurry of research activity focusing either on RAP itself or on problems with a similar flavor. All tend to be NP-hard; in RAP, setting each $s_i = 0$ and $t_i = 1$ gives KNAPSACK as a special case of RAP. Accordingly, research on RAP strives to obtain polynomial-time approximation algorithms. Since it is not known whether RAP is MaxSNP-hard, the possibility of a polynomial-time approximation scheme (PTAS) remains open.

Using LP rounding techniques, Phillips et al. [8] obtain a 1/6-approximation algorithm[5] for RAP. Actually they solve a more general problem in which each task occupies not a fixed interval of time but has a fixed length and a window of time in which it is allowed to slide. Using the local ratio technique, Bar-Noy et al. [2] obtain results which imply a 1/4-approximation algorithm for RAP. Bar-Noy [3] has since informed us that the techniques in [2] can in fact yield a 1/3-approximation algorithm. Using different ideas, Chen et al. [4] have recently obtained a 1/3-approximation for RAP in the special case where the profit of each task i equals its "area" $(t_i - s_i)b_i$.

The STORAGE ALLOCATION PROBLEM is a related problem in which, in addition, the resource must be allocated to the scheduled tasks in contiguous

[5] Since RAP is a maximization problem, approximation factors of algorithms for it will be at most 1 — the larger the factor, the better.

blocks, which must not change during the lifetime of the task. Both Bar-Noy et al. [2] and Leonardi et al. [6] study this problem and the latter paper obtains a 1/12-approximation algorithm, which is the current best. The former paper obtains results for several related problems.

1.2 Our Results

We present three algorithms for restricted versions of RAP. Suitably combining these algorithms leads to approximation algorithms for the general case. The best approximation ratio that we can obtain improves upon previous results mentioned above.

Theorem 1.1. *Consider* RAP *with the restriction that every task satisfies* $b_i > \delta B$*, for a constant* $0 < \delta \leq 1$*. There is an algorithm that solves the problem exactly and runs in time* $O(n^{2/\delta + O(1)})$*.*

Theorem 1.2. *For every* δ*,* $0 < \delta \leq 0.976$*, there is a randomized* $O(n^2 \log^2 n)$*-time algorithm for the special case of* RAP *in which all* $b_i \leq \delta B$*, which achieves a* $(1 - 4\varepsilon)$*-approximation with high probability, where* $\varepsilon = \sqrt{(8/3)\delta \ln(1/\delta)}$*.* [6]

Since $\lim_{\delta \to 0} \delta \ln(1/\delta) = 0$, we conclude that for any $\varepsilon > 0$ there is a $\delta > 0$ such that there is a randomized, polynomial-time, $(1 - 4\varepsilon)$-approximation algorithm if all $b_i \leq \delta B$.

Theorem 1.3. *The restriction of* RAP *to inputs having all* $b_i \leq B/2$ *has a deterministic polynomial-time* $1/2$*-approximation algorithm.*

In what follows, we shall refer to our algorithms which prove the above theorems as the *Large Tasks Algorithm*, the *Small Tasks Algorithm* and the *List Algorithm*, respectively.

To obtain results for unrestricted RAP, we can "combine" the algorithms for special cases in a certain manner. The combined algorithm's running time will be the sum of the running times of the constituents. Combining the Large Tasks and Small Tasks algorithms gives the following result.

Theorem 1.4. *Given any constant* $\varepsilon > 0$*, there is a randomized polynomial-time algorithm that approximates* RAP *within a factor of at least* $1/2 - \varepsilon$*, with high probability. The exponent in the running time is* $\mathrm{poly}(1/\varepsilon)$*.* (*More precisely, the exponent is a constant times the smaller of the two positive* δ*'s that satisfy* $\varepsilon = \sqrt{\delta \log(1/\delta)}$*.*)

One can trade approximation guarantee for running time, and simplicity, by combining the List Algorithm with a well-known interval graph algorithm (details in Section 4):

Theorem 1.5. *There is a deterministic* $1/3$*-approximation algorithm for* RAP *with running time* $O(n^2 \log^2 n)$*.*

The technique of the proof of Theorem 1.5 is very simple and quite different from that used by Bar-Noy [3] to obtain the same approximation ratio.

[6] This statement is vacuous unless $\delta < 0.0044$.

1.3 Organization of the Rest of the Paper

The remainder of the paper proves the above theorems. In Section 2 we make some basic definitions and explain how to combine algorithms. Section 3 describes the Large Tasks Algorithm, the Small Tasks Algorithm and the recipe for combining them to obtain Theorem 1.4. Finally, Section 4 describes the List Algorithm and proves Theorem 1.5.

2 Preliminaries

Assumption 2.1 *Throughout the paper we shall assume that the tasks are numbered* 1 *to* n *in order of increasing starting time, ties broken arbitrarily, i.e.,* $i < j \Rightarrow s_i \leq s_j$.

Definition 2.2. *We say that task* i *is* active *at an instant* t *of time* (*where* t *is any real number*) *if* $t \in [s_i, t_i)$. *Also, for each* i, *we define* S_i *to be the set of tasks in* $\{1, 2, \ldots, i\}$ *that are active at time* s_i: $S_i = \{j \leq i \,:\, t_j > s_i\}$. *Note that* $i \in S_i$.

Notice that a task is considered active at its start time s_i but not at its finish time t_i; this prevents task i from competing for resources with a task which ends at time s_i.

We observe that RAP can be easily expressed as an integer linear program as follows:

$$\text{Maximize } \sum_{i=1}^{n} p_i x_i \tag{1}$$

$$\text{s.t. } \sum_{j \in S_i} b_j x_j \leq B, \qquad 1 \leq i < n\ , \tag{2}$$

$$\text{and } x_i \in \{0, 1\}, \qquad 1 \leq i \leq n\ . \tag{3}$$

This integer program has a natural LP relaxation, obtained by replacing (3) by the constraint

$$0 \leq x_i \leq 1, \qquad 1 \leq i \leq n\ . \tag{4}$$

We shall refer to the LP given by (1), (2) and (4) as LPMAIN.

Definition 2.3. *A set* $U \subseteq \{1, \ldots, n\}$ *of tasks is called a* packing *if its characteristic vector satisfies the constraints* (2)–(3).

2.1 Solving the LP

We note that LPMAIN can be solved by a min-cost flow algorithm, based on the following previously-known construction similar to the one described in [1]. Construct the network $\mathcal{N}$, with vertex set $V(\mathcal{N}) = \{1, 2, \ldots, n, n+1\}$. For every $1 \leq i \leq n$, add an arc $(i, i+1)$ of capacity B and cost 0. For every task i, define $r_i = \min\{j : 1 \leq j \leq n \text{ and } s_j \geq t_i\}$, with the convention that the minimum over an empty set is $n+1$. Then add an arc (r_i, i) of capacity b_i and cost $-p_i/b_i$. These are all the arcs of $\mathcal{N}$. A valid flow in $\mathcal{N}$ with flow on the arc (r_i, i) equal

to f_i corresponds to a feasible solution to LPMAIN with $x_i = f_i/b_i$ and has cost $\sum_{i=1}^{n} p_i x_i$. Therefore a min-cost flow in $\mathcal{N}$ corresponds to an optimum solution to LPMAIN.

The best known strongly polynomial-time min-cost flow algorithm, due to Orlin [7], solves LPMAIN in $O(n^2 \log^2 n)$ time.

2.2 Combining Two Algorithms For Restricted RAP

Suppose algorithm $\mathcal{A}_L$ works when each $b_i > \tau$ and yields an exact (optimal) solution, and algorithm $\mathcal{A}_S$ works when each $b_i \leq \tau$ and yields an α-approximation for some constant $\alpha \leq 1$. The dividing point τ may depend on B.

1. Divide the tasks in the input instance into *small tasks* (those with $b_i \leq \tau$), and *large tasks* (those with $b_i > \tau$).
2. Using $\mathcal{A}_S$, compute an α-approximation to the optimal packing that uses small tasks alone.
3. Using $\mathcal{A}_L$, compute an optimal packing that uses large tasks alone.
4. Of the two packings obtained in Steps 2 and 3, choose the one with greater profit.

Note that in any problem instance, if OPT denotes the profit of the optimal packing and OPT_L (respectively, OPT_S) denotes the profit of the optimal packing using only large (respectively, small) tasks, then

$$\text{either } \mathsf{OPT}_S \geq \frac{1}{1+\alpha} \cdot \mathsf{OPT} \quad \text{or } \mathsf{OPT}_L \geq \frac{\alpha}{1+\alpha} \cdot \mathsf{OPT} \ .$$

Therefore the above combined algorithm achieves an approximation ratio of at least $\alpha/(1+\alpha)$.

3 The Large and Small Tasks Algorithms

In this section, we prove Theorems 1.1 and 1.2. For the former, we give a dynamic programming algorithm that does not consider LPMAIN at all. For the latter, we solve LPMAIN and use a randomized rounding scheme to obtain a good approximation. We then derive Theorem 1.4 from these two results as indicated above.

3.1 Dynamic Programming for the Large Tasks

We describe how to solve RAP exactly, provided each $b_i > \delta B$. We begin with some definitions.

Let U be a packing. We shall denote the profit of U, i.e., the sum of profits of all tasks in U, by $p(U)$. If $U \neq \emptyset$, we denote the highest numbered task in U by $\mathrm{top}(U)$; this is a task in U which starts last.

Definition 3.1. *For a packing $U \neq \emptyset$, we define its* kernel $\ker(U)$ *to be the set of all tasks in U which are active at the time when* $\mathrm{top}(U)$ *starts:* $\ker(U) = \{i \in U : [s_i, t_i) \ni s_{\mathrm{top}(U)}\} = \{i \in U : t_i > s_{\mathrm{top}(U)}\}$. *We also define* $\ker(\emptyset) = \emptyset$.

Definition 3.2. *A packing U is called a* pile *if* $\ker(U) = U$.

A pile is a clique in the underlying interval graph and it is a feasible solution to the RAP instance. Any kernel is a pile. Now define the real function f on piles V as follows:

$$f(V) = \max \{p(U) : U \text{ is a packing with } \ker(U) = V\} .$$

Note that $f(\emptyset) = 0$. Clearly, if we can compute $f(V)$ for all V, we are done. We show below how to do this efficiently.

For any packing $U \neq \emptyset$, let $U' = U \setminus \{\mathrm{top}(U)\}$. For a pile $V \neq \emptyset$, let $\mathcal{T}(V)$ denote the following set of piles:

$$\mathcal{T}(V) \;=\; \left\{W \supseteq V' : W \text{ is a pile and } t_j \leq s_{\mathrm{top}(V)} \text{ for all } j \in W \setminus V')\right\} . \quad (5)$$

Clearly $\mathcal{T}(V) \neq \emptyset$, since $V' \in \mathcal{T}(V)$. In words, $\mathcal{T}(V)$ consists of all piles obtained from V by removing its highest numbered ("rightmost") task and adding in zero or more tasks that end exactly when or before this removed task starts. We shall need some facts about the functions defined so far:

Lemma 3.3. *In any instance of* RAP*, the following hold:*
(i) *For any packing $U \neq \emptyset$,* $\mathrm{top}(U) = \mathrm{top}(\ker(U))$.
(ii) *For any packing $U \neq \emptyset$,* $\ker(U') \supseteq (\ker(U))'$.
(iii) *Let $V \neq \emptyset$ be a pile and $W \in \mathcal{T}(V)$. If $W \neq \emptyset$ then* $\mathrm{top}(W) < \mathrm{top}(V)$.
(iv) *Let $V \neq \emptyset$ be a pile, $W \in \mathcal{T}(V)$, and U be a packing with* $\ker(U) = W$. *Then $U \cup \{\mathrm{top}(V)\}$ is a packing.*
(v) *Let $V \neq \emptyset$ be a pile, $W \in \mathcal{T}(V)$, and U be a packing with* $\ker(U) = W$. *Then* $\ker(U \cup \{\mathrm{top}(V)\}) = V$.
(vi) *Let $V \neq \emptyset$ be a pile and X be a packing with* $\ker(X) = V$. *Then* $\ker(X') \in \mathcal{T}(V)$.

Proof. The proofs are straightforward.
(i) This is trivial from the definitions.
(ii) When U is a singleton this is trivial, so we assume that U has at least two elements; so $U' \neq \emptyset$.

Let $k = \mathrm{top}(U)$ and $k' = \mathrm{top}(U') < k$. Let $j \in (\ker(U))'$; by part (i), this means $j \in \ker(U) \setminus \{k\}$, i.e., $j \in \ker(U) \subseteq U$ and $j < k$; therefore $j \in U'$ and in particular $j \leq k'$. By Assumption 2.1 we have $s_j \leq s_{k'}$ and since $j \in \ker(U)$, we have $t_j > s_k \geq s_{k'}$; thus j is active when $k' = \mathrm{top}(U')$ starts and so $j \in \ker(U')$.
(iii) For any $j \in W$, by (5), either $j \in V'$ whence $j < \mathrm{top}(V)$, or else $j \in W \setminus V'$ and hence $s_j < t_j \leq s_{\mathrm{top}(V)}$ whence $j < \mathrm{top}(V)$. Thus, in particular, $\mathrm{top}(W) < \mathrm{top}(V)$.
(iv) First of all, note that if $\ker(U) = W = \emptyset$, then $U = \emptyset$ and $\{\mathrm{top}(V)\}$ is trivially a packing, since every singleton is. So we may assume $W \neq \emptyset$ and $U \neq \emptyset$.

Let $k = \mathrm{top}(V)$ and $\ell = \mathrm{top}(U)$. By parts (i) and (iii), $\ell = \mathrm{top}(U) = \mathrm{top}(\ker(U)) = \mathrm{top}(W) < \mathrm{top}(V) = k$. Thus $\mathrm{top}(U \cup \{k\}) = \max\{\ell, k\} = k$. This means that at any time $\tau < s_k$, the set $U \cup \{k\}$ of tasks satisfies the feasibility constraint (2) at time τ (because U is given to be a packing). It remains to show that the feasibility constraint is satisfied at time s_k as well. To this end, we shall show that the set of tasks in $U \cup \{k\}$ which are active at time s_k is a subset of V; since V is a packing, this will complete the proof.

Let $j \in U \cup \{k\}$ be active at time s_k. If $j = k$ we immediately have $j \in V$. Otherwise, since j is active at time s_k, and $k > \ell$ as shown above, $t_j > s_k \geq s_\ell$. Also, $j \leq \mathrm{top}(U) = \ell$, so $s_j \leq s_\ell$. Thus j is active at time s_ℓ, so $j \in \ker(U) = W$.

Suppose $j \notin V$. Then $j \notin V'$ and by (5) we see that $t_j \leq s_k$, a contradiction. Therefore $j \in V$ and we are done.

(v) First, note that $W = \emptyset$ implies $U = \emptyset$ and, by (5), $V' = \emptyset$, which means V is a singleton: $V = \{\mathrm{top}(V)\}$. The statement is now trivial. So we may assume $W \neq \emptyset$ and $U \neq \emptyset$.

Let $k = \mathrm{top}(V)$. As shown above, $\mathrm{top}(U \cup \{k\}) = k$. Suppose $j \in \ker(U \cup \{k\})$. Then j is active at time s_k. In the proof of part (iv) we have already seen that this implies $j \in V$. Thus $\ker(U \cup \{k\}) \subseteq V$.

Now suppose $j \in V$. Since V is a pile, j is active at time $s_{\mathrm{top}(V)} = s_k$. As shown above, $k = \mathrm{top}(U \cup \{k\})$, so j is active at the start time of job $\mathrm{top}(U \cup \{k\})$. We claim that $j \in U \cup \{k\}$. Indeed, if $j \neq k$, then $j \in V \setminus \{k\} = V'$. Since $W \in \mathcal{T}(V)$, by definition of $\mathcal{T}(V)$ (see (5)) we have $W \supseteq V'$. Thus $j \in V' \subseteq W \subseteq U$, which proves the claim. This in turn shows that $j \in \ker(U \cup \{k\})$. Thus $V \subseteq \ker(U \cup \{k\})$.

(vi) Since V is nonempty, so is X. Let $k = \mathrm{top}(V) = \mathrm{top}(X)$. By part (ii), $\ker(X') \supseteq (\ker(X))' = V'$. Let $j \in \ker(X') \setminus V'$. Looking at (5), in order to prove that $\ker(X') \in \mathcal{T}(V)$, we must show that $t_j \leq s_k$. Suppose not. Since $j \in \ker(X') \subseteq X' \subseteq X$, we have $j \leq k$. This means j is active at time s_k, since we've assumed that $t_j > s_k$; therefore $j \in \ker(X) = V$. But $j \notin V'$; so $j = k$. However, $j \in X' = X \setminus \{k\}$, which is a contradiction. □

The key to the dynamic programming algorithm is the following lemma:

Lemma 3.4. *Let $V \neq \emptyset$ be a pile. Then*

$$f(V) = p_{\mathrm{top}(V)} + \max_{W \in \mathcal{T}(V)} f(W) \ . \tag{6}$$

Proof. Let R denote the right-hand side of (6).

We first establish that $f(V) \geq R$. Let $W \in \mathcal{T}(V)$ be a pile that maximizes $f(W)$ among all such W, and let U be a packing with $\ker(U) = W$ that maximizes $p(U)$ among all such U. If $W = \emptyset$ we have $R = p_{\mathrm{top}(V)}$. Since V is a pile, it is a packing with kernel V, so $f(V) \geq p(V) \geq p_{\mathrm{top}(V)} = R$.

So assume $W \neq \emptyset$. Then $U \neq \emptyset$. Now $R = p_{\mathrm{top}(V)} + f(W) = p_{\mathrm{top}(V)} + p(U)$. By parts (i) and (iii) of Lemma 3.3, $\mathrm{top}(U) = \mathrm{top}(\ker(U)) = \mathrm{top}(W) < \mathrm{top}(V)$. Therefore $\mathrm{top}(V) \notin U$. By parts (iv) and (v) of Lemma 3.3, $U \cup \{\mathrm{top}(V)\}$ is a packing with kernel V. By definition of f this means

$$f(V) \ \geq \ p\big(U \cup \{\mathrm{top}(V)\}\big) \ = \ p(U) + p_{\mathrm{top}(V)} \ = \ R \ .$$

Now we establish that $f(V) \leq R$. Let X be a packing with $\ker(X) = V$ that maximizes $p(X)$ among all such X; then $f(V) = p(X)$. Also, X is nonempty since V is. By part (i) of Lemma 3.3, $\text{top}(X) = \text{top}(V) = k$, say. Since $k \in X$, $p(X) = p_k + p(X \setminus \{k\}) = p_k + p(X') \leq p_k + f(\ker(X'))$, where the inequality follows from the definition of f. By part (vi) of Lemma 3.3, $\ker(X') \in \mathcal{T}(V)$. Therefore

$$f(V) \;=\; p(X) \;\leq\; p_k + f(\ker(X')) \;\leq\; p_k + \max_{W \in \mathcal{T}(V)} f(W) \;=\; R\;,$$

which completes the proof of the lemma. □

We now describe the dynamic programming algorithm.

Proof (of Theorem 1.1). Compute $f(V)$ for all piles V in increasing order of $\text{top}(V)$, using formula (6). By part (iii) of Lemma 3.3, every pile $W \in \mathcal{T}(V)$, $W \neq \emptyset$, satisfies $\text{top}(W) < \text{top}(V)$ so that when computing $f(V)$ we will have already computed $f(W)$ for every W involved in (6). Then compute the profit of the optimal packing as the maximum of $f(V)$ over all piles V.

The above algorithm computes only the profit of an optimal packing, but it is clear how to modify it to find an optimal packing as well.

The running time is clearly at most $\text{poly}(n)$ times a quantity quadratic in the number of piles. Since each task has $b_i > \delta B$, the size of a pile is at most $1/\delta$ and so the number of distinct nonempty piles is at most $\sum_{k=1}^{\lfloor 1/\delta \rfloor} \binom{n}{k} \leq (1+n)^{1/\delta}$. This gives a running time of $O(n^{2/\delta + O(1)})$, as claimed. □

3.2 Randomized Rounding for the Small Tasks

In this subsection we prove Theorem 1.2. Accordingly, we assume that the tasks in the input instance satisfy $b_i \leq \delta B$ and $\delta \leq 0.976$. We set

$$\varepsilon = \sqrt{\frac{8}{3}\,\delta \ln(1/\delta)}\;. \tag{7}$$

We shall describe an algorithm, based on randomized rounding of an LP solution, which returns a solution to RAP whose expected performance ratio is at least $1 - 2\varepsilon$. Repeating this algorithm several times one can get a $(1-4\varepsilon)$-approximation with high probability, thereby proving the theorem.

If $1 - 4\varepsilon \leq 0$, there is nothing to prove, so we may assume that $\varepsilon < 1/4$. Since $f(\delta) = \sqrt{(8/3)\delta \ln(1/\delta)}$ is increasing on $(0, 1/e)$, decreasing on $(1/e, 1]$, $f(0.976) > 1/4$, $f(0.0044) > 1/4$, and $\delta \leq 0.976$, we infer that

$$\delta < 0.0044\;. \tag{8}$$

We solve LPMAIN using the algorithm indicated in Section 2.1. Suppose this gives us an optimal fractional solution $(x_1, x_2, \ldots, x_n)$ with profit OPT^*. We then choose independent random variables $Y_i \in \{0, 1\}$, $i = 1, 2, ..., n$, with

$\Pr[Y_i = 1] = (1-\varepsilon)x_i$. Now, if we define the (dependent) random variables $Z_1, Z_2, ..., Z_n$, in that order, as follows:

$$Z_i = \begin{cases} 1, & \text{if } Y_i = 1 \text{ and } \sum_{j \in S_i \setminus \{i\}} b_j Z_j \leq B - b_i \ , \\ 0, & \text{otherwise} \ , \end{cases}$$

then clearly $\{i : Z_i = 1\}$ is a packing, and its expected profit is $\sum_{i=1}^{n} p_i \cdot \Pr[Z_i = 1]$. We shall now lower bound the probability that $Z_i = 1$. For this purpose we need a simple probabilistic fact and a tail estimate, collected together in the following lemma.

Lemma 3.5. *Let $X_1, X_2, \ldots, X_m$ be independent random variables and let $0 \leq \beta_1, \beta_2, ..., \beta_m \leq 1$ be reals, where for $i \in \{1, 2, \ldots, m\}$, $X_i = \beta_i$ with probability p_i, and $X_i = 0$ otherwise. Let $X = \sum_i X_i$ and $\mu = E[X]$. Then*
(i) $\sigma(X) \leq \sqrt{\mu}$.
(ii) *For any λ with $0 < \lambda < \sqrt{\mu}$, $\Pr[X > \mu + \lambda\sqrt{\mu}] < \exp\left(-\frac{\lambda^2}{2}\left(1 - \lambda/\sqrt{\mu}\right)\right)$.*

Proof. For part (i):

$$\begin{aligned} \sigma^2(X) &= \sum_{i=1}^{m} \left(E[X_i^2] - (E[X_i])^2\right) \\ &\leq \sum_{i=1}^{m} E[X_i^2] \\ &\leq \sum_{i=1}^{m} E[X_i] \\ &= \mu \ , \end{aligned}$$

where the second inequality follows from $\beta_i \leq 1$.

To prove part (ii), put $t = \ln\left(1 + \lambda/\sqrt{\mu}\right) \geq 0$. Trivially, $t \geq \lambda/\sqrt{\mu} - \lambda^2/(2\mu)$ since $0 \leq \lambda/\sqrt{\mu} < 1$. By Markov's inequality, $\Pr\left[X > \mu + \lambda\sqrt{\mu}\right] = \Pr\left[e^{tX} > e^{t\mu + t\lambda\sqrt{\mu}}\right] < E[e^{tX}]/\exp(t\mu + t\lambda\sqrt{\mu})$. Now

$$\begin{aligned} E[e^{tX}] &= E\left[\prod_{i=1}^{m} e^{tX_i}\right] \\ &= \prod_{i=1}^{m} E\left[e^{tX_i}\right] \\ &= \prod_{i=1}^{m} \left(1 - p_i + p_i e^{t\beta_i}\right) \\ &\leq \prod_{i=1}^{m} \exp\left(p_i(e^{t\beta_i} - 1)\right) \\ &= \exp\left(\sum_{i=1}^{m} p_i \left(e^{t\beta_i} - 1\right)\right) \ . \end{aligned}$$

Further,

$$\begin{aligned}\sum_{i=1}^{m} p_i \left(e^{t\beta_i} - 1\right) &= \sum_{i=1}^{m} p_i \left(t\beta_i + \frac{1}{2!}(t\beta_i)^2 + \frac{1}{3!}(t\beta_i)^3 + \cdots\right) \\ &\leq \sum_{i=1}^{m} p_i \beta_i \left(e^t - 1\right) \\ &= \mu \left(e^t - 1\right) \ ,\end{aligned}$$

where the inequality follows from the fact that each $\beta_i \leq 1$. This gives

$$\begin{aligned}\Pr\left[X > \mu + \lambda\sqrt{\mu}\right] &< \exp\left(\mu(e^t - 1) - t\mu - t\lambda\sqrt{\mu}\right) \\ &\leq \exp\left(\lambda\sqrt{\mu} - \lambda\sqrt{\mu} + \lambda^2/2 - \lambda^2 + \lambda^3/(2\sqrt{\mu})\right) \\ &= \exp(-\lambda^2/2 + \lambda^3/(2\sqrt{\mu})) \ ,\end{aligned}$$

where the last inequality follows from the lower bound on t. □

Lemma 3.6. *Under conditions* (7) *and* (8), $\Pr[Z_i = 1] \geq (1 - 2\varepsilon)x_i$.

Proof. Conditions (7) and (8) imply $\varepsilon > 57\delta$. This fact will be used twice below.

Fix an i. We shall estimate the probability $\pi_i = \Pr[Z_i = 0 \mid Y_i = 1]$. Notice that Z_j does not depend on Y_i when $j < i$. Since $Z_j \leq Y_j$ for all j and $b_i \leq \delta B$, we now have

$$\pi_i \ = \ \Pr\left[\sum_{j \in S_i \setminus \{i\}} b_j Z_j > B - b_i\right] \ \leq \ \Pr\left[\sum_{j \in S_i \setminus \{i\}} \frac{b_j Y_j}{\delta B} > \frac{1-\delta}{\delta}\right] . \qquad (9)$$

Now the random variables $\{b_j Y_j/(\delta B)\}_{j \in S_i \setminus \{i\}}$, and $\beta_j = b_j/(\delta B)$, $p_j = (1-\varepsilon)x_j$ satisfy the conditions of Lemma 3.5. Let Y be the sum of the new random variables and let $\mu = E[Y]$. We have

$$\mu \ = \ \sum_{j \in S_i \setminus \{i\}} \frac{b_j}{\delta B} \cdot (1-\varepsilon)x_j \ = \ \frac{1-\varepsilon}{\delta} \sum_{j \in S_i \setminus \{i\}} \frac{b_j x_j}{B} \ \leq \ \frac{1-\varepsilon}{\delta} \ , \qquad (10)$$

by constraint (2) of LPMAIN. We now consider two cases.

Case 1: $\mu < (7/8)(1-\delta)/\delta$. Part (i) of Lemma 3.5 gives $\sigma(Y) \leq \sqrt{\mu}$. Using (9) and Chebyshev's inequality we get

$$\begin{aligned}\pi_i &\leq \Pr\left[Y > \frac{1-\delta}{\delta}\right] \\ &\leq \Pr\left[\left|Y - \mu\right| > \frac{1}{8} \cdot \frac{1-\delta}{\delta} \cdot \frac{\sigma(Y)}{\sqrt{\mu}}\right] \\ &\leq \frac{64\delta^2\mu}{(1-\delta)^2} \ .\end{aligned}$$

By our assumption about μ, this is less than $56\delta/(1-\delta)$ and now by (8), $\pi_i < 57\delta$.

Case 2: $\mu \geq (7/8)(1-\delta)/\delta$. Set λ such that $\mu + \lambda\sqrt{\mu} = (1-\delta)/\delta$. Then λ, considered as a function of μ, is decreasing. From this and (10) we have

$$\lambda \;=\; \frac{\frac{1-\delta}{\delta} - \mu}{\sqrt{\mu}} \;\geq\; \frac{\frac{1-\delta}{\delta} - \frac{1-\varepsilon}{\delta}}{\sqrt{\frac{1-\varepsilon}{\delta}}} \;=\; \frac{\varepsilon - \delta}{\sqrt{\delta(1-\varepsilon)}} \;\geq\; \frac{\varepsilon - \delta}{\sqrt{\delta}} \;\geq\; \frac{56}{57}\frac{\varepsilon}{\sqrt{\delta}} \;.$$

Also, $1 - \lambda/\sqrt{\mu} = 2 - (1-\delta)/\delta\mu \geq 6/7$, by the assumption about μ, and further, we trivially get $\lambda < \sqrt{\mu}$. By (9) and part (ii) of Lemma 3.5, applied to the variables $\{b_j Y_j/(\delta B)\}_{j \in S_i \setminus \{i\}}$ and their sum Y, we obtain

$$\begin{aligned}
\pi_i \;&\leq \Pr[Y > \mu + \lambda\sqrt{\mu}] \\
&< \exp\left(-\frac{\lambda^2}{2}(1 - \lambda/\sqrt{\mu}) \right) \\
&< \exp\left(-\frac{1}{2}\left(\frac{56}{57}\right)^2 \frac{\varepsilon^2}{\delta}\frac{6}{7} \right) \\
&< \delta \;.
\end{aligned}$$

In either case, $\pi_i < 57\delta$. Hence,

$$\Pr[Z_i = 1] = (1-\pi_i)\cdot\Pr[Y_i = 1] \geq (1-57\delta)(1-\varepsilon)x_i \geq (1-\varepsilon)^2 x_i \geq (1-2\varepsilon)x_i \;,$$

which completes the proof of the lemma. □

Proof (of Theorem 1.2). The randomized rounding procedure computes a packing; let the random variable P denote its profit. As noted in the comments preceding Lemma 3.5, $E[P] = \sum_{i=1}^{n} p_i \cdot \Pr[Z_i = 1]$ which, by Lemma 3.6, is at least $(1-2\varepsilon)\mathsf{OPT}^* \geq (1-2\varepsilon)\mathsf{OPT}$. However, P never exceeds OPT. Markov's inequality now implies $\Pr[P \geq (1-4\varepsilon)\mathsf{OPT}] \geq 1/2$. By repetition, we can now suitably amplify the probability of obtaining at least $1-4\varepsilon$ of the profit. The bound on the running time follows easily from Subsection 2.1. □

3.3 Proof of Theorem 1.4

We have already described in Subsection 2.2 how to combine the two algorithms described above. In the terminology of that section, we would have $\alpha = 1-4\varepsilon$ and $\tau = \delta B$ where ε and δ are related according to (7). As argued at the end of that section, this would lead to an approximation ratio of $\alpha/(1+\alpha) = (1-4\varepsilon)/(2-4\varepsilon) \geq 1/2 - 2\varepsilon$ for general RAP.

Given an ε, in order to achieve this $1/2 - 2\varepsilon$ approximation, we solve (7) for δ (we use the smaller δ obtained) and use this δ in the Large and Small Tasks Algorithms.

4 The List Algorithm

In this section we give a fast and simple 1/2-approximation algorithm in the case when each $b_i \leq B/2$; this will prove Theorem 1.3. It is inspired by a similar "coloring" algorithm of Phillips et al [8]. Our algorithm has the advantage that it does not have to round to a large common denominator; therefore it is faster and obtains a 1/2-approximation, rather than a $(1/2 - \varepsilon)$-approximation.

Our algorithm generates a list of packings with the property that the "average" of the packings in the list has large profit. To be precise, we first solve LPMAIN and obtaining an optimal fractional solution $(x_1, x_2, \ldots, x_n)$ whose profit is $\sum_{i=1}^{n} p_i x_i$. Our rounding algorithm will produce a list $U_1, U_2, \ldots, U_m$ of sets of tasks, together with non-negative real weights $x(U_1), \ldots, x(U_m)$ for these sets. These sets and weights will satisfy the following properties:

1. Each set U_k is a packing.
2. $0 \leq x(U_k) \leq 1$ for each set U_k.
3. For each i we have $\sum_{k:\, U_k \ni i} x(U_k) = x_i$.
4. $\sum_{k=1}^{m} x(U_k) \leq 2$.

Define the profit of a set to be the sum of the profits of the constituent tasks. Our algorithm will select a maximum profit set from the list it constructs. Let P be the profit of the set picked by the algorithm. Then, assuming that the above properties hold, $\frac{1}{2}\sum_{i=1}^{n} p_i x_i = \frac{1}{2}\sum_{i=1}^{n}\sum_{k:\, U_k \ni i} p_i x(U_k) = \sum_{k=1}^{m} \frac{1}{2} x(U_k) \sum_{i: i \in U_k} p_i \leq \sum_{k=1}^{m} \frac{1}{2} x(U_k) \cdot P \leq P$, where the first equality follows from Property 3, the first inequality from Property 1 and the final inequality from Property 4. Therefore we indeed have a 1/2-approximation algorithm.

We now describe the procedure for generating the sets U_k and weights $x(U_k)$. Initialize a list $\mathcal{L}$ of sets to an empty list. Then consider the tasks in the order $1, 2, \ldots, n$. For task i:

1. If $x_i = 0$ proceed to the next task. (If there are no more tasks, stop.)
2. Search $\mathcal{L}$ for a set not containing i to which i can be added without violating Property 1.
3. If no such set exists, create a new set $V = \{i\}$ with weight $x(V) = x_i$ and add V to $\mathcal{L}$. Set $x_i = 0$ and return to step 1.
4. Otherwise, suppose $U \in \mathcal{L}$ is such a set.
   - 4a. If $x_i < x(U)$ then decrease $x(U)$ to $x(U) - x_i$, create a new set $V = U \cup \{i\}$ with weight $x(V) = x_i$ and add V to $\mathcal{L}$. Set $x_i = 0$ and return to step 1.
   - 4b. If $x_i \geq x(U)$, add i to U and decrease x_i to $x_i - x(U)$. Return to step 1.

Lemma 4.1. *After processing all the tasks as above, $\mathcal{L}$ will hold a list of tasks satisfying all four properties.*

Proof. Let $\hat{x}_i$ denote the *original* values of x_i that were input to the above procedure. It is easy to verify that the procedure maintains the following invariant:

$$x_i + \sum_{j:U_j \ni i} x(U_j) = \hat{x}_i, \qquad 1 \le i \le n \ . \tag{11}$$

Properties 1 and 2 are clearly satisfied. Property 3 follows from (11) and the fact that after all tasks have been processed, each $x_i = 0$.

It remains to prove that Property 4 holds; we shall show that it holds as an invariant of the procedure. A new set may be added to the list $\mathcal{L}$ either from step 3 or step 4a. In the latter case, the weight of a set is split amongst itself and the newly created set, leaving the sum of all weights unaffected. In the former case, the newly created set is a singleton consisting of task i, say. Consider the list $\mathcal{L}$ immediately after this singleton is added. Note that every task j in a set in $\mathcal{L}$ satisfies $j \le i$. For each set $U \in \mathcal{L}$ let $b(U)$ be the sum of b_j over all $j \in U$ such that task j is active at the time when task i starts (i.e., $s_j \le s_i < t_j$, since $j \le i$). Then

$$\sum_{U \in \mathcal{L}} b(U)x(U) = \sum_{U \in \mathcal{L}} \sum_{\substack{j \in U \\ s_j \le s_i < t_j}} b_j x(U) \tag{12}$$

$$= \sum_{\substack{j: j \le i, \\ s_j \le s_i < t_j}} b_j \sum_{U \in \mathcal{L}: U \ni j} x(U) \tag{13}$$

$$= \sum_{\substack{j: j \le i, \\ s_j \le s_i < t_j}} b_j \hat{x}_j \tag{14}$$

$$\le B \ . \tag{15}$$

Equation (13) holds because we process tasks in the order $1, \ldots, n$, (14) holds because of (11) and the fact that when task i is being processed we have $x_j = 0$ for all $j < i$, and (15) holds because of constraint (2) of LPMAIN.

Let $\mathcal{L}'$ be the sublist of $\mathcal{L}$ consisting of those sets which contain i. For $U \in \mathcal{L}'$ we clearly have $b(U) \ge b_i$. For $U \in \mathcal{L} \setminus \mathcal{L}'$, since we're in the case when step 3 is executed, it follows that task i did not fit into any set in $\mathcal{L} \setminus \mathcal{L}'$ and so $b(U) > B - b_i$. Therefore we get

$$B \ \ge \ \sum_{U \in \mathcal{L}'} b_i x(U) + \sum_{U \in \mathcal{L} \setminus \mathcal{L}'} (B - b_i) x(U) \ = \ b_i \hat{x}_i + (B - b_i) \sum_{U \in \mathcal{L} \setminus \mathcal{L}'} x(U)$$

where the equality follows from (11). Consider the last expression; if we have $\sum_{U \in \mathcal{L} \setminus \mathcal{L}'} x(U) > \hat{x}_i$, then the expression is a decreasing function of b_i. Since $b_i \le B/2$, we get

$$B \ \ge \ \frac{B}{2} \hat{x}_i + \frac{B}{2} \sum_{U \in \mathcal{L} \setminus \mathcal{L}'} x(U) \ . \tag{16}$$

If, on the other hand, we have $\sum_{U \in \mathcal{L} \setminus \mathcal{L}'} x(U) \le \hat{x}_i \le 1$, then (16) clearly holds. Thus (16) always holds, and applying (11) to it gives $B \ge \frac{B}{2} \sum_{U \in \mathcal{L}} x(U)$, whence Property 4 follows. □

Proof (of Theorem 1.5). We combine two RAP algorithms as described in Section 2.2, with $\alpha = 1/2$ and $\tau = B/2$ in the terminology of that section. For one half we use the above List Algorithm.

For the other half, observe that if each task in a RAP instance has $b_i > B/2$, then two tasks which are active together (at some instant of time) cannot both be in a packing. Therefore, a packing in this case is simply an independent set in the underlying interval graph of the RAP instance; so the problem reduces to (weighted) MAX-INDEPENDENT-SET for interval graphs. This problem is well-known to be solvable (exactly) in $O(n \log n)$ time (see, e.g., [2]).

The combination gives us an approximation guarantee of $\alpha/(1+\alpha) = 1/3$.

Finally, consider the running time of the List Algorithm. Note that the size m of list $\mathcal{L}$ is at most n, since each task creates at most one new set to be added to $\mathcal{L}$. This means that the List Algorithm does at most $O(n^2)$ work after solving the LP. From Subsection 2.1 we know that the LP can be solved in $O(n^2 \log^2 n)$ time. This completes the proof. □

References

1. E. M. Arkin, E. B. Silverberg. Scheduling jobs with fixed start and end times. Discrete Applied Mathematics, **18** (1987), 1–8.
2. Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, Baruch Schieber. A unified approach to approximating resource allocation and scheduling. In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (2000), 735–744.
3. Amotz Bar-Noy. Private communication (2001).
4. Bo Chen, Rafael Hassin, Michal Tzur. Allocation of bandwidth and storage. IIE Transactions, **34** (2002), 501–507.
5. N. G. Hall, M. J. Magazine. Maximizing the value of a space mission. European Journal of Operational Research, **78** (1994), 224–241.
6. Stefano Leonardi, Alberto Marchetti-Spaccamela, Andrea Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In Proceedings of the 20th conference on Foundations of Software Technology and Theoretical Computer Science (2000), 409–420.
7. J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. Operations Research, **41** (1993), 338–350.
8. Cynthia Phillips, R. N. Uma, Joel Wein. Off-line admission control for general scheduling problems. In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (2000), 879–888.