

Truthful Unification Framework for Approximating Packing Integer Programs with Choices*

Yossi Azar[†]

Iftah Gamzu[‡]

Abstract

One of the most interesting research directions within the field of algorithmic mechanism design revolves the study of hard combinatorial optimization problems. In this setting, many common techniques, which are broadly used by approximation algorithms, cannot be utilized as they violate certain monotonicity properties that are imperative for truthfulness. Consequently, it seems of the essence to develop alternative algorithmic methods, which can underlie truthful mechanisms. In particular, since many hard optimization problems can be formulated as instances of integer linear programs, it seems that devising techniques that apply to integer linear programs is significantly important. This perception reinforces by the observation that one of the most general and efficient ways to approximately solve such programs, which is linear programming-based randomized rounding, generally fails to be monotone.

In this paper, we focus our attention on packing integer programs, and packing integer programs with choices. Our main findings can be briefly summarized as follows:

1. We develop a framework, which can be used as a building block to approximately solve packing integer programs and packing integer programs with choices. The framework is built upon a unification technique that approximately solves an instance of a packing integer program with choices, given algorithms that approximately solve sub-instances of it. The framework is deterministic and monotone, and hence can underlie truthful deterministic mechanisms.
2. We demonstrate the applicability of the framework by employing it to several NP-hard problems. In particular, we focus on the *bandwidth allocation problem in tree networks*, and the *multiple knapsack problem on bipartite graphs*. Notably, using the mentioned framework, we attain the first non-trivial approximation guarantees for these problems in a game theoretic setting.

*An extended abstract of this paper appeared in *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pages 833–844, 2008.

[†]Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA and School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: azar@tau.ac.il. Supported in part by the German-Israeli Foundation and by the Israel Science Foundation.

[‡]School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: iftgam@tau.ac.il. Supported by the Binational Science Foundation, by the Israel Science Foundation, and by the European Commission under the Integrated Project QAP funded by the IST directorate as Contract Number 015848.

1 Introduction

The field of *algorithmic mechanism design*, which was introduced by Nisan and Ronen [33], studies the design of protocols or mechanisms for algorithmic problems in scenarios where the input is presented by *strategic agents*. A strategic agent might be dishonest about its part of the input in order to manipulate the protocol in a way that will maximize its own gain. A primary interest of algorithmic mechanism design is in the development of efficiently computable *truthful* or *incentive compatible* protocols, which are robust against manipulation by agents, i.e., every agent is rationally motivated to truthfully report its input.

One of the most intriguing research directions within the field of algorithmic mechanism design revolves the study of hard combinatorial optimization problems. These problems are hard in a sense that any computationally-efficient algorithm designed to solve them can only attain sub-optimal guarantees, that is, they only admit approximation algorithms. The motivation for this research realm originates in two inherent difficulties that arise in the alluded setting. The first is that the most fundamental technique of mechanism design, the Vickrey-Clarke-Groves (VCG) mechanism [42, 17, 23], cannot be utilized since it must be built upon an algorithm that obtains an exact optimal solution [28, 32]. The other is that many common algorithmic methods cannot be employed as they violate certain *monotonicity* properties that are imperative for truthfulness.

In light of this state of affairs, it seems of the essence to develop modifications and alternatives to generic algorithmic techniques, which are broadly employed by approximation algorithms, so they could underlie truthful mechanisms. In particular, since many hard optimization problems can be formulated as instances of integer linear programs, it seems that devising methods to approximately solve integer linear programs is significantly important. This perception reinforces by the fact that one of the most general and efficient ways to approximately solve such programs, namely, linear programming-based randomized rounding [37, 36, 40], generally fails to be monotone (see, e.g., the discussion in [4]).

1.1 Our results

The main contribution of this paper is in presenting a framework that can be used as a building block to approximately solve packing integer programs and packing integer programs with choices. The framework is built upon a unification technique, which approximately solves an instance of a packing integer program with choices, given algorithms that approximately solve sub-instances of it. The framework is deterministic and monotone, and hence can underlie truthful deterministic mechanisms. In order to demonstrate the applicability and strength of this framework, we apply it to several hard optimization problems. Primarily, we investigate the bandwidth allocation problem in tree networks, and the multiple knapsack problem on bipartite graphs.

Truthful deterministic unification framework. We focus on truthfully approximating maximization problems, which can be posed as packing integer programs and packing integer programs with choices of the forms exhibited in Figure 1. Essentially, we concentrate on the class of packing integer programs with choices as it incorporates the class of packing integer programs as a special case. In a packing integer program with choices, there are n variables that are partitioned into ℓ pairwise-disjoint sets C_1, C_2, \dots, C_ℓ . The objective is to maximize a linear value function over these variables, subject to a set of packing constraints, and a set of choices constraints, which prohibit the selection of more than one variable from each set. From an algorithmic mechanism design point of view, there are ℓ strategic *single parameter* agents, each of which coincides with a valuation, and may be untruthful about it. The goal is to maximize the *social welfare*. One natural approach for approximating a packing integer program with choices is to partition it to sub-instances, solve each of them, and output the sub-solution that has a maximum value. Unfortunately, this approach fails to be monotone, unless the underlying sub-algorithms are monotone

and *bitonic* [31]. In order to avoid this shortcoming, we devise a deterministic unification technique, easing the requirements from the sub-algorithms, and still attaining monotonicity and provable approximation guarantee. This result appears in Section 3.

$\begin{array}{ll} \mathbf{max} & v \cdot x \\ \mathbf{s.t.} & A \cdot x \leq b \\ & x \in \{0, 1\}^n \end{array}$	$\begin{array}{ll} \mathbf{max} & v \cdot (C \cdot x) \\ \mathbf{s.t.} & A \cdot x \leq b \\ & C \cdot x \leq 1^\ell \\ & x \in \{0, 1\}^n \end{array}$
--	---

Figure 1: A packing integer program (left) and a packing integer program with choices (right). Remark that $A \in \mathbb{R}_+^{m \times n}$, $b \in \mathbb{R}_+^m$, and $v \in \mathbb{R}_+^n$ ($v \in \mathbb{R}_+^\ell$) in the packing (packing with choices) case. Also note that $C \in \{0, 1\}^{\ell \times n}$ is a *choices matrix* in which the i -th row corresponds to the variables set C_i . That is, the i -th row has ones in the entries that coincide with the variables in C_i , and zeros elsewhere. Notice that the class of packing integer programs is obtained as a special case of packing integer programs with choices when C is the $n \times n$ identity matrix.

Bandwidth allocation in tree networks. An instance of the *bandwidth allocation problem in tree networks* consists of an undirected tree T with m edges, and a set \mathcal{R} of ℓ connection requests such that every request $r \in \mathcal{R}$ is characterized by a quadruple (s_r, t_r, d_r, v_r) , where s_r and t_r are the respective *source* and *target* vertices of the request, $d_r \in (0, 1]$ is the *demand* associated with the request, and v_r is the positive *value* gained by allocating the request. The goal is to select a maximum value subset of requests $S \subseteq \mathcal{R}$ so that the aggregate demands of the requests in S , which cross any edge, does not exceed the *bandwidth capacity*, which is unit. Let P_r denote the unique simple path between s_r and t_r in T , this problem can be posed by the following packing integer program:

$$\begin{array}{ll} \text{maximize} & \sum_{r \in \mathcal{R}} v_r x_r \\ \text{subject to} & \sum_{r \in \mathcal{R} | e \in P_r} d_r x_r \leq 1 \quad \forall e \in E \\ & x_r \in \{0, 1\} \quad \forall r \in \mathcal{R} \end{array}$$

In the game theoretic version of this problem there are ℓ strategic agents, each of which controls a request and may be dishonest about its value. We study this variant using the aforesaid framework, and devise a monotone deterministic algorithm, which achieves an approximation ratio of $O(\ln \ln m)$. This result implies a corresponding truthful mechanism, and is the first non-trivial deterministic result for this problem. Further details are provided in Section 4.

Multiple knapsack on bipartite graphs. An instance of the *multiple knapsack problem on bipartite graphs* consists of a set M of m unit-capacity knapsacks, and a set \mathcal{I} of ℓ items such that every item $i \in \mathcal{I}$ is characterized by a pair (s_i, v_i) , where $s_i \in (0, 1]$ is the *size* of the item, and v_i is the positive *value* gained by placing the item in one of the knapsacks. An additional ingredient of the input is an items-knapsacks bipartite graph, which represents the assignment restrictions of the items to the knapsacks. In particular, the bipartite graph exhibits a set of admissible knapsacks $M_i \subseteq M$, for every $i \in \mathcal{I}$. The goal is to select a maximum value subset of items $S \subseteq \mathcal{I}$, along with an admissible knapsack for each selected item, so that all the items in S can be simultaneously placed in their designated knapsacks, while preserving the capacities of the knapsacks. This problem can be modelled by the following packing integer program with

choices:

$$\begin{aligned}
& \text{maximize} && \sum_{i \in \mathcal{I}} v_i \cdot \left(\sum_{j \in M_i} x_{ij} \right) \\
& \text{subject to} && \sum_{i \in \mathcal{I} | j \in M_i} s_i x_{ij} \leq 1 && \forall j \in [m] \\
& && \sum_{j \in M_i} x_{ij} \leq 1 && \forall i \in \mathcal{I} \\
& && x_{ij} \in \{0, 1\} && \forall i \in \mathcal{I}, j \in M_i
\end{aligned}$$

In the game theoretic version of this problem there are ℓ strategic agents, each of which owns an item, and may be untruthful about its value. We consider this variant using the mentioned framework, and develop a monotone deterministic 11-approximation algorithm, which implies a corresponding truthful mechanism. This result is the first deterministic result for this problem, which does not assume constant number of knapsacks. The specifics of this finding are presented in Section 5.

1.2 Related work

It seems fundamentally important to develop modifications and alternatives to common techniques, which cannot be utilized in the context of algorithmic mechanism design. However, there are only a handful of results addressing this goal. Mu’alem and Nisan [31] seem to have been the first to pay attention to the development of such general tools. Primarily, they exhibited necessary conditions for the truthful utilization of two basic building blocks, which are the **max** and **if-then-else** operators. Briest, Krysta and Vöcking [11] continued this line of research. They devised a general approach to transform a pseudo-polynomial algorithm into a monotone FPTAS, and demonstrated that primal-dual greedy algorithms may be the key to truthfully solve some integer linear programs. Later on, Lavi and Swamy [27] developed a general technique to convert an approximation algorithm in packing domains to a randomized approximation mechanism that is *truthful in expectation*. Finally, Babaioff, Lavi and Pavlov [7] presented a method that turns any given algorithm to a dominant-strategy mechanism in single parameter domains. However, their method degrades the performance guarantee of the resulting mechanism by a factor of $O(\log \rho)$, where ρ denotes the ratio between the largest and smallest valuations.

Focusing on the bandwidth allocation problem in tree networks and the multiple knapsack problem on bipartite graphs, the first observation one can make is that they are NP-hard since they generalize the *knapsack problem*. Accordingly, most of past research focused on the development of approximation algorithms for these problems. The best known result for bandwidth allocation problem in tree networks is by Lewin-Eytan, Naor and Orda [29], who presented a 5-approximation algorithm, based on the local ratio technique [9]. Recent years have also seen ever-growing line of work addressing variants of this problem using various algorithmic tools such as primal-dual approach [22], linear programming-based methods [35, 15, 8], and mixtures of several techniques [12, 13, 10]. Unfortunately, it is easy to demonstrate that most of these positive results are not monotone. Consequently, attaining a non-trivial deterministic approximation for the bandwidth allocation problem in an algorithmic mechanism design setting is still an open question, even when the underlying network is a line. On a different note, it is relevant to point out that Briest, Krysta and Vöcking [11] studied the mechanism design variant of the *unsplittable flow problem*, which is a generalization of the bandwidth allocation problem. Hence, their upper bound result follows to our case. Yet, when the demands of the requests are arbitrary, their algorithm achieves a performance guarantee that can be as high as a polynomial in m .

Turning to the multiple knapsack problem on bipartite graphs, it is known to be approximable within a factor that is slightly better than $e/(e-1) \approx 1.58$ by the work of Feige and Vondrák [20]. Similarly to the bandwidth allocation problem, past years have also seen respectable amount

of research addressing variants of the multiple knapsack problem [18, 14, 34, 21]. From an algorithmic mechanism design point of view, the only provable result is a monotone PTAS for a generalization of the problem, referred to as the *generalized assignment problem* [11]. However, this result holds only when the number of knapsacks is fixed.

2 Preliminaries

In this section, we introduce the notation and terminology to be used throughout the paper, and describe a characterization that interlinks monotone algorithms with truthful mechanisms. We begin with the notation:

- Let Π denote a problem that can be posed as a packing integer program with choices. Essentially, Π can be considered to be a collection of (infinite) input instances, each of which represented by a quadruple (v, A, C, b) , consisting of a concrete valuations vector v , constraints matrix A , choices matrix C , and constraints vector b .
- Let $\Pi|_a$ be a collection of input instances achieved by the restriction of the instances in Π according to an attribute a of the columns of the constraints matrix A . Namely, every instance $(v, A, C, b) \in \Pi$ gives rise to a sub-instance $(v|_a, A|_a, C|_a, b) \in \Pi|_a$ such that $A|_a$ consists of the subset of the columns of A that satisfy the attribute a , $C|_a$ consists of the matching subset of columns of C , the set of variables $x|_a$ comprises of the suitable subset of variables of x , and $v|_a$ is the implied subset of entries of v , i.e., the set of entries that have at least one allied variable in $x|_a$. For instance, in the bandwidth allocation problem in tree networks, every column of A represents a request. Thus, a non-trivial attribute may state “having a distance of 1”. This attribute gives rise to sub-instances that only consists of requests that have a distance of 1 between their terminals.
- Given an input instance $I = (v, A, C, b)$, let $\text{PIP}(I)$ denote the corresponding integer program instance, and let $\text{LP}(I)$ be the relaxation of $\text{PIP}(I)$ obtained by replacing the integrality constraint $x \in \{0, 1\}^n$ with $x \in [0, 1]^n$. Moreover, let $\text{OPT}_I^{\text{int}}$ and $\text{OPT}_I^{\text{frac}}$ be the values of the optimal solutions of $\text{PIP}(I)$ and $\text{LP}(I)$, respectively. Finally, let d_Π denote the integrality gap of problem Π , formally defined as $d_\Pi = \sup_{I \in \Pi} \text{OPT}_I^{\text{frac}} / \text{OPT}_I^{\text{int}}$. Note that for notational simplicity we will mark the integrality gap of $\Pi|_a$ by d_a .

We now present the notion of *monotonicity*, and then turn to describe a characterization that reduces the goal of designing truthful mechanisms to that of designing monotone algorithms. Note that the illustrated terms are refined to the specific setting considered. Thus, the keen reader is encouraged to refer to [11] and the references therein for a brief introduction to the field of algorithmic mechanism design, and more comprehensive overview of the underlying concepts.

Definition 2.1. An algorithm \mathcal{M} is said to be *monotone w.r.t.* Π if it satisfies the following property, for every $(v, A, C, b) \in \Pi$: if the solution x generated by \mathcal{M} w.r.t. $\text{PIP}(v, A, C, b)$ satisfies that agent i is chosen, i.e., $x_j = 1$ for some $j \in C_i$, then the solution \tilde{x} generated by \mathcal{M} w.r.t. $\text{PIP}(\tilde{v}, A, C, b)$, where \tilde{v} is a valuations vector in which $\tilde{v}_i \geq v_i$ and the other values are fixed, also satisfies that agent i is chosen, i.e., $\tilde{x}_{j'} = 1$ for some $j' \in C_i$ that may satisfy $j' \neq j$.

Theorem 2.2. ([31]) *If algorithm \mathcal{M} is monotone w.r.t. Π then there is a matching truthful mechanism for Π , which can be efficiently computed using \mathcal{M} .*

3 A Truthful Unification Framework

In this section, we present a deterministic unification framework that can be used to truthfully approximate maximization problems, which can be represented by the packing integer program

with choices described in Figure 1. As previously indicated, one widely accepted approach for approximating a packing integer program with choices is to solve sub-instances of it, and then to pick the best sub-solution. Unfortunately, this algorithmic tool fails to be monotone, unless the underlying algorithms are monotone and bitonic¹. The main problem in picking the best sub-solution resides in the “selection-outcome linkage”. Namely, the selection of which sub-solution to output coincides with the outcomes of the underlying algorithms. Consequently, if one of the underlying algorithms is not bitonic, e.g., if its outcome value decreases when the value of a chosen agent increases, there may be settings in which an increase in the value of this agent will result in the selection of a different sub-solution in which this agent is not chosen. In the following, we design a method that breaks this link. Prior to describing the finer details of our approach, we introduce a definition that formalizes what is a monotone algorithm that generates sub-solutions for sub-instances.

Definition 3.1. \mathcal{M} is *monotone a -restrictive c -approximation algorithm w.r.t. Π* if it is monotone w.r.t. Π , and its solution x to $\text{PIP}(I)$ satisfies the following properties, for any $I = (v, A, C, b) \in \Pi$:

- x is restricted w.r.t. the attribute a , i.e., $x_j = 1$ only if $j \in C_i$ and $v_i \in v_{|a}$.
- the value of the solution is at least $\text{OPT}_{I_a}^{\text{int}}/c$, where $I_a = (v_{|a}, A_{|a}, C_{|a}, b)$.

We are now ready to establish the main result of this section. Let a_1, a_2, \dots, a_k be a collection of *pairwise-disjoint choices-consistent* attributes w.r.t. Π . Specifically, these attributes partition every instance $(v, A, C, b) \in \Pi$ to k sub-instances $(v_{|a_1}, A_{|a_1}, C_{|a_1}, b), \dots, (v_{|a_k}, A_{|a_k}, C_{|a_k}, b)$ in a way that maintains the following two properties. The first property is *pairwise-disjointness*, which means that every column of A satisfies exactly one attribute. This property guarantees that every column of A , every column of C , and every variable of x appears in exactly one restricted sub-instance. The second property is *choices-consistent*, which means that all the variables allied to some entry of v are picked by the same attribute. This property assures that every entry of v appears in exactly one restricted sub-instance. Notice that the last attribute is trivially maintained for (pure) packing integer programs by any pairwise-disjoint collection of attributes.

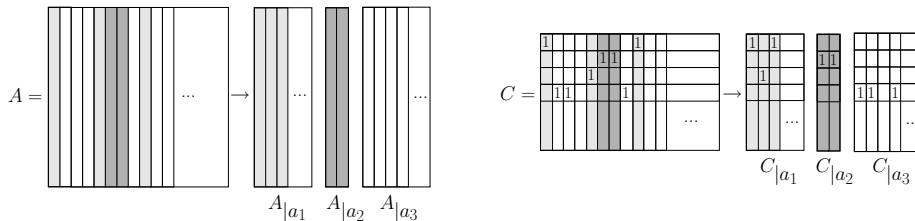


Figure 2: A schematic description of a constraints matrix, and a choices matrix partition induced by a collection of pairwise-disjoint choices-consistent attributes. Note that the color of every column corresponds to the attribute it satisfies.

Theorem 3.2. *Given a family of deterministic algorithms $\{\mathcal{M}_i\}_{i=1}^k$, where each \mathcal{M}_i is monotone a_i -restrictive c_{a_i} -approximation algorithm w.r.t. Π , MAX-Select is monotone deterministic $\sum_{i=1}^k c_{a_i} d_{a_i}$ -approximation algorithm w.r.t. Π .*

Proof. One can easily verify that algorithm MAX-Select is deterministic, and produces a feasible solution. Hence, we are left to prove that it is monotone, and that it generates a solution which

¹Informally, an algorithm is *bitonic* if its outcome value as a function of the value of any single agent i has the pattern that it does not increase as long as agent i is not chosen, and does not decrease as long as agent i is chosen. A formal definition can be found in [31, 11].

Algorithm 1 MAX-Select(I)

Input: An instance $I = (v, A, C, b) \in \Pi$

Output: A solution x

- 1: **for** $i = 1$ to k **do**
 - 2: **Calculate** $\text{OPT}_{I_i}^{\text{frac}}$ exactly by solving $\text{LP}(I_i)$, where $I_i = (v_{|a_i}, A_{|a_i}, C_{|a_i}, b)$
 - 3: **end for**
 - 4: **Let** j be the minimal index for which $\text{OPT}_{I_j}^{\text{frac}}/c_{a_j}d_{a_j} \geq \text{OPT}_{I_i}^{\text{frac}}/c_{a_i}d_{a_i}$, for every $1 \leq i \leq k$
 - 5: **Simulate** algorithm $\mathcal{M}_j(I)$ to obtain x
 - 6: **return** x
-

is a $\sum_{i=1}^k c_{a_i}d_{a_i}$ -approximation for the optimal one. We begin by analyzing the approximation ratio of the algorithm. Let ALG be the value of the solution that the algorithm, as well as the subroutine $\mathcal{M}_j(I)$, outputs. We obtain that

$$\text{OPT}_I^{\text{int}} \leq \sum_{i=1}^k \text{OPT}_{I_i}^{\text{int}} \leq \sum_{i=1}^k \text{OPT}_{I_i}^{\text{frac}} \leq \sum_{i=1}^k \frac{c_{a_i}d_{a_i}}{c_{a_j}d_{a_j}} \text{OPT}_{I_j}^{\text{frac}} \leq \sum_{i=1}^k \frac{c_{a_i}d_{a_i}}{c_{a_j}} \text{OPT}_{I_j}^{\text{int}} \leq \sum_{i=1}^k c_{a_i}d_{a_i} \text{ALG} .$$

The first inequality follows by observing that given a solution to $\text{PIP}(I)$, one can restrict it to the variables corresponding to some attribute a_i , and yield a feasible solution to $\text{PIP}(I_i)$. The third inequality is due to the selection rule of the algorithm, i.e., $\text{OPT}_{I_j}^{\text{frac}}/c_{a_j}d_{a_j} \geq \text{OPT}_{I_i}^{\text{frac}}/c_{a_i}d_{a_i}$, for every $1 \leq i \leq k$. The fourth inequality results by recalling that the integrality gap of $\Pi_{|a_j}$ is d_{a_j} . Finally, the last inequality holds since $\text{ALG} \geq \text{OPT}_{I_j}^{\text{int}}/c_{a_j}$, which is a property of \mathcal{M}_j .

We now argue that the algorithm is monotone. Consider an agent t that is chosen by the algorithm w.r.t. the instance $I = (v, A, C, b)$. Let \tilde{v} be a valuations vector in which the value of t is $\tilde{v}_t \geq v_t$, and the values of all the other agents are fixed. For the sake of monotonicity, we need to prove that t is chosen by the algorithm w.r.t. the input $J = (\tilde{v}, A, C, b)$. Since both I and J share the same constraints matrix, it follows that the attributes a_1, \dots, a_k partition them into the same sub-instances. Namely, if we let I_i and J_i be the respective restriction of I and J to attribute a_i then the last observation implies that J_j and I_j are identical up to a difference in the value of agent t , and $J_i = I_i$, for every $i \neq j$. Recall that the induced partition is choices-consistent. Consequently, one can infer that $\text{OPT}_{J_j}^{\text{frac}} \geq \text{OPT}_{I_j}^{\text{frac}}$, and $\text{OPT}_{J_i}^{\text{frac}} = \text{OPT}_{I_i}^{\text{frac}}$, for every $i \neq j$. This implies that when J is the input, the algorithm must also execute \mathcal{M}_j . Accordingly, agent t must be chosen since \mathcal{M}_j is known to be monotone w.r.t. Π . ■

Corollary 3.3. *Given a family of deterministic algorithms $\{\mathcal{M}_i\}_{i=1}^k$, where each \mathcal{M}_i is monotone a_i -restrictive c_{a_i} -approximation algorithm w.r.t. Π , algorithm MAX-Select supports a truthful deterministic mechanism that approximates Π to within a ratio of $\sum_{i=1}^k c_{a_i}d_{a_i}$.*

Before we turn to demonstrate the applicability of the framework, let us briefly outline the key goals one needs to address in order to utilize it. The first issue that needs to be dealt is the *attributes*. Namely, one should define a collection of pairwise-disjoint choices-consistent attributes, which partition every input instance of the problem under consideration. The second issue that ought to be resolved is the *algorithms*. In particular, one needs to develop a family of monotone deterministic algorithms, which are restrictive w.r.t. the different attributes. The last matter to be handled is the *integrality gaps*. That is, one should analyze the integrality gaps of the sub-problems defined w.r.t. each attribute.

4 Bandwidth Allocation Problems

In this section, we study several bandwidth allocation problems. The main result of this section is a monotone deterministic algorithm for the bandwidth allocation problem in tree networks,

which utilizes the framework displayed in Section 3, and achieves an approximation guarantee of $O(\ln \ln m)$. The specifics of this finding appears in Subsection 4.1. Furthermore, we show how to employ the mentioned framework for other closely related problems, while attaining comparable approximation ratios. These results are exhibited in Subsection 4.2.

4.1 The bandwidth allocation problem in tree networks

In the following, we investigate the bandwidth allocation problem in tree networks using a framework-guided approach. Namely, we begin by studying restricted versions of the problem under consideration in which some characteristics of the requests are guaranteed to have a certain pattern. Essentially, these characteristics are the attributes that partition every unrestricted input instance to sub-instances. For each restricted case, we design a monotone deterministic algorithm that approximately solves it, and analyze the integrality gap of the corresponding packing integer program. Later on, we show how to consolidate these results within the framework, and yield the aforementioned outcome.

Restricted demands. We consider a restricted version of the bandwidth allocation problem in tree networks in which the demand of every request is guaranteed to have a certain pattern. Particularly, we investigate instances in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$, where $i \in \mathbb{N}$. The integrality gap of this restricted version is known to be bounded by a constant, for any $i \in \mathbb{N}$, following the work of Chekuri, Mydlarz and Shepherd [15]. We now introduce a family of monotone deterministic algorithms $\{\text{BAP}_i\}_{i \in \mathbb{N}}$, which approximately solve the restricted versions of the problem. Namely, algorithm BAP_i handles restricted instances in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$. Prior to describing the algorithm, let us consider an inherently simpler scenario, in which the demand of each request is unit. This special case is equivalent to the *edge disjoint paths problem in a tree*. In general, the edge disjoint paths problem is known to be hard both in directed graphs [24, 16], and in undirected graphs [3, 2]. Nonetheless, when the underlying graph is a tree, there is a deterministic polynomial-time optimal algorithm [41], which will henceforth be referred to as algorithm wEDP . Now, we are ready to describe algorithm BAP_i . Algorithm BAP_i has two steps. First, it rounds-up all the demands of the requests to 2^{-i} . Then, it executes algorithm wEDP for 2^i times, where the input to the j -th execution of wEDP is the set of requests that have not been selected in the first $j - 1$ executions of wEDP . Informally, one may picture the second step as an iterative packing of edge disjoint paths in different layers, each of bandwidth 2^{-i} . It is worth noting that the algorithm, and its analysis are partially inspired by the technique suggested by Awerbuch et al. [5] for reducing a call admission problem in multi-wavelength scenario to that of a single-wavelength case.

Algorithm 2 $\text{BAP}_i(\mathcal{R})$

Input: A requests instance \mathcal{R} in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$

Output: A subset $S \subseteq \mathcal{R}$ of selected connection requests

- 1: **Let** $k = 2^i$
 - 2: **Let** \mathcal{R}' be the set of requests $\{(s_r, t_r, 1/k, v_r) : r \in \mathcal{R}\}$
 - 3: **for** $j = 1$ to k **do**
 - 4: **Simulate** algorithm wEDP on \mathcal{R}' to obtain S_j
 - 5: Let $\mathcal{R}' = \mathcal{R}' \setminus S_j$
 - 6: **end for**
 - 7: **return** $\bigcup_{j=1}^k S_j$
-

Prior to analyzing the performance of algorithm BAP_i , we address the *wavelength partition problem on tree networks*. This problem investigates the correspondence between a feasible set of

requests in a tree, and its partition into wavelength. Specifically, given a feasible set Q of requests, each having a demand of $1/k$, where $k \in \mathbb{N}_+$, the goal is to determine the minimum number of subsets needed for a pairwise-disjoint partition of the requests, which maintains the property that in any subset no two requests share an edge. In what follows, we establish a simple upper bound of $2k - 1$ on the number of subsets. This upper bound will be used later, when we analyze the approximation guarantee of algorithm BAP_i . Note that this problem has received considerable attention in the past, and upper bounds that are better than $2k - 1$ are known [38, 30, 25, 26, 19]. Notwithstanding, we decided to present the indicated upper bound for completeness.

Claim 4.1. *Let Q be a feasible set of requests in a tree, each having a demand of $1/k$, where $k \in \mathbb{N}_+$. There exists a partition of Q into a collection of $2k - 1$ pairwise-disjoint subsets such that in any subset no two requests share an edge.*

Proof. Let us consider a procedure that initially sorts the requests in an non-decreasing *lowest common ancestor*² distance from the root, and then iteratively assigns every request to a feasible subset with a minimal index. In what follows, we argue that this procedure generates at most $2k - 1$ subsets. Assume by contradiction that this is not the case, i.e., when the procedure is executed on Q it generates at least $2k$ subsets. Let r be the request that leads to the creation of the $2k$ -th subset, and let u be its lowest common ancestor. Notice that the $2k$ -th subset is created because r shares an edge with at least one request in each of the first $2k - 1$ subsets. Let us denote the set of requests that prevented the augmentation of r to the first $2k - 1$ subsets by B . Recall that the requests are inspected in an non-decreasing lowest common ancestor order. This implies that each of the requests in B must contain the vertex u , and must intersect r on at least one of its (possibly two) edges adjacent to u . Consequently, one can infer that there is a set of at least k requests of B that intersect a particular edge of r . This clearly contradicts the feasibility of the set Q . ■

We now turn to analyze the performance of algorithm BAP_i . For ease of presentation, we let \mathcal{R} denote an input instance in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$, and introduce the following notation:

- Let $S = \bigcup_{j=1}^k S_j$ be the set of requests selected by algorithm BAP_i w.r.t. \mathcal{R} . In addition, let Q , Q^d , and Q^u denote the set of requests appearing in an optimal solution w.r.t. \mathcal{R} , $\{(s_r, t_r, 2^{-(i+1)}, v_r) : r \in \mathcal{R}\}$, and $\{(s_r, t_r, 2^{-i}, v_r) : r \in \mathcal{R}\}$.
- Let $\text{ALG} = \sum_{r \in S} v_r$ be the value of the solution output by algorithm BAP_i w.r.t. \mathcal{R} . In a similar manner, let $\text{OPT} = \sum_{r \in Q} v_r$, $\text{OPT}^u = \sum_{r \in Q^u} v_r$, and $\text{OPT}^d = \sum_{r \in Q^d} v_r$. Notice that $\text{OPT}^d \geq \text{OPT} \geq \text{OPT}^u$.

Theorem 4.2. *Let \mathcal{R} be an instance in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$. Algorithm $\text{BAP}_i(\mathcal{R})$ is monotone, deterministic, and outputs a feasible solution whose value is at least $1/12$ of an optimal solution's value.*

Proof. It is easy to verify that algorithm BAP_i is deterministic, and outputs a feasible solution. Therefore, we are left to establish that it is monotone, and that it attains the suggested approximation guarantee. We begin by proving that the value of the solution generated by the algorithm is at least $1/12$ of an optimal solution's value. Essentially, we yield this result by proving the following two claims.

Claim I: $\text{OPT}^u \geq \text{OPT}/4$. By Claim 4.1, we know that there is a partition of Q^d into a collection of 2^{i+2} pairwise-disjoint subsets $Q_1^d, \dots, Q_{2^{i+2}}^d$ such that in any subset no two requests share an

²The *lowest common ancestor* of a request r is a vertex u in the path P_r , which is closest to the root of the tree.

edge. Consider the following grouping of the pairwise-disjoint subsets

$$Q^{(p)} = \bigcup_{j=p \cdot 2^i + 1}^{(p+1) \cdot 2^i} Q_j^d, \text{ for every integer } 0 \leq p \leq 3.$$

Since $Q^{(0)}, Q^{(1)}, Q^{(2)}$, and $Q^{(3)}$ are disjoint, it follows that the sum of the requests values in one of these sets is at least $\text{OPT}^d/4$. Let us assume without loss of generality that $Q^{(0)}$ satisfies this property, and observe that the set of requests in $Q^{(0)}$ constitute a feasible solution even when all the demands of the requests are rounded-up to 2^{-i} . Consequently, $\text{OPT}^u \geq \text{OPT}^d/4$, which in turn implies that $\text{OPT}^u \geq \text{OPT}/4$.

Claim II: $\text{ALG} \geq \text{OPT}^u/3$. By Claim 4.1, we know that there is a partition of $Q^u \setminus S$ into a collection of 2^{i+1} pairwise-disjoint subsets $Q_1^u, \dots, Q_{2^{i+1}}^u$ such that in any subset no two requests share an edge. Consider some subset Q_p^u . Recall that all the requests of Q_p^u were candidates in every execution of algorithm **wEDP**, but none of them was selected. Since algorithm **wEDP** is optimal, it follows that $\sum_{r \in Q_p^u} v_r \leq \sum_{r \in S_j} v_r$, for every $1 \leq j \leq 2^i$. Consequently,

$$\begin{aligned} \text{OPT}^u &= \sum_{r \in Q^u} v_r \leq \sum_{r \in S} v_r + \sum_{r \in Q^u \setminus S} v_r \\ &= \sum_{r \in S} v_r + \sum_{j=1}^{2^{i+1}} \sum_{r \in Q_j^u} v_r \\ &\leq \sum_{r \in S} v_r + 2 \sum_{j=1}^{2^i} \sum_{r \in S_j} v_r = 3 \sum_{r \in S} v_r = 3\text{ALG}. \end{aligned}$$

We now turn to establish that the algorithm is monotone. Consider a request r that has a value of v_r , and suppose that it is selected in the j -th execution of algorithm **wEDP**. Now, suppose that r had a value of $\tilde{v}_r \geq v_r$, and the values of all the other requests were fixed. For the sake of monotonicity, we need to prove that the algorithm would have selected r in the latter case. If r is selected during one of the first $j - 1$ executions of algorithm **wEDP** then we are done. Otherwise, let us consider the j -th execution. One can easily observe that in the first $j - 1$ executions of the algorithm, the same set of requests must be selected whether the value of r is v_r or \tilde{v}_r . This results from the optimality of algorithm **wEDP**. Correspondingly, the same set of unselected requests are candidates in the j -th execution. This implies that r must be selected in the j -th execution. \blacksquare

Narrow demands. We turn to examine the restricted version of the problem in which the demand of every request is guaranteed to be *narrow*, that is $d_r \in (0, O(1/\ln m)]$, for every $r \in \mathcal{R}$. The integrality gap of this restricted version is known to be $1 + \epsilon$ by the outcome of algorithms that employ the randomized rounding technique [37, 36, 40]. In addition, this restricted setting is a special case of the $\Omega(\ln m)$ -*bounded unsplittable flow problem*, which is known to admit a polynomial-time monotone deterministic algorithm that attains constant approximation [11, 6]. Consequently, referring to the algorithm of [11] as **BAP**, we obtain the following theorem.

Theorem 4.3. ([11]) *Let \mathcal{R} be an instance in which $d_r \in (0, 1/(100 \ln m)]$, for every $r \in \mathcal{R}$. Algorithm **BAP**(\mathcal{R}) is monotone, deterministic, and outputs a feasible solution whose value is at least $1/3$ of an optimal solution's value.*

Integration of the results within the framework. We now demonstrate how to consolidate the results obtained for the restricted versions of the problem under the umbrella of the

framework. Fundamentally, we have already attained the key ingredients needed to employ the main theorem of the framework. Let $k = c \ln \ln m + 1$, where c is a sufficiently large constant for which $2^{-(c \ln \ln m)} \leq 1/(100 \ln m)$. Let a_1, a_2, \dots, a_k be attributes that restrict input instances of the problem according to the demands of the requests. Explicitly, attribute a_1 picks out all the requests that have a demand in the range of $(2^{-1}, 1]$, attribute a_2 picks out the requests with a demand in the range of $(2^{-2}, 2^{-1}]$, and so on until attribute a_{k-1} . The last attribute, a_k , picks out all the remaining requests. Namely, requests that have a demand in the range of $(0, 1/(100 \ln m)]$. It is clear that this collection of attributes partition every input instance of the problem in a pairwise-disjoint (and trivially choices-consistent) way. In addition, let $\{\text{BAP}_i\}_{i=1}^{k-1} \cup \{\text{BAP}\}$ be the corresponding family of deterministic algorithms. Specifically, BAP_i is a monotone a_i -restrictive 12-approximation algorithm, and BAP is a monotone a_k -restrictive 3-approximation algorithm. Finally, recall that the integrality gap of each a_i -restrictive variant of the problem is constant, and the integrality gap of the a_k -restrictive variant is $1 + \epsilon$. Accordingly, and in correspondence with Theorem 3.2, we achieve the following theorem.

Theorem 4.4. *There is a monotone deterministic $O(\ln \ln m)$ -approximation algorithm for the bandwidth allocation problem in tree networks.*

4.2 Additional applications

We briefly describe how the framework can be deployed for other closely related problems.

Bandwidth allocation in uni-directional cycle networks. The bandwidth allocation problem in uni-directional cycle networks is an admission control problem in which the underlying network is a clockwise directed cycle. Chakrabarti et al. [13] described a simple procedure that reduces this problem to line networks. The idea is to partition the cycle into a line network and a single edge, then to solve a bandwidth allocation problem on a line network and a knapsack problem on the single edge, and finally to output the solution that have maximum value. This procedure guarantees an $(\alpha + 1 + \epsilon)$ -approximation, where α is the approximation ratio of the algorithm for the line network. Unfortunately, embedding both the monotone FPTAS for the knapsack problem [11] and our monotone algorithm for line networks in this procedure does not yield a monotone algorithm. Yet again, the essence of this disability lies in the fact that applying a max operator on two monotone algorithms does not guarantee monotonicity, unless both of them are also bitonic. Nevertheless, using the framework devised in Section 3, we can develop a monotone $O(\ln \ln m)$ -approximation algorithm to this problem. The main idea is to refine the attributes in a way that will “simulate” the procedure of [13]. Specifically, we choose an edge $e \in E$, and define the attributes in a similar manner to before, but with the additional requirement that they only pick out requests that do not use the edge e . Furthermore, we define an additional attribute, which picks out every request that uses the edge e . Now, the analysis follows almost identically as before.

Bandwidth allocation in bi-directional cycle networks. A bi-directional cycle is the simplest network where in addition to admission control concerns, one needs to take into consideration routing decisions. A natural attempt may be to follow the scheme suggested for the uni-directional cycle. However, this approach fails as the attributes are not choices-consistent. The crucial observation needed to overcome this difficulty is that we can remove any single edge of the cycle, and the value of the optimal solution for the resulting line network will be at least $1/2$ of the value of the optimal solution for the initial cycle network. Consequently, our monotone algorithm for line networks can be used to yield an approximation guarantee of $O(\ln \ln m)$.

5 Knapsack Problems

In this section, we consider several knapsack problems. The main contribution of this section is a monotone deterministic algorithm for the multiple knapsack problem on bipartite graphs, which employs the framework exhibited in Section 3, and attains constant approximation guarantee. The particulars of this result appear in Subsection 5.1. Furthermore, we show how to refine the analysis, and obtain improved approximation ratio for the multiple knapsack problem in Subsection 5.3.

5.1 The multiple knapsack problem on bipartite graphs

In the following, we design a relatively simple monotone deterministic algorithm that approximately solves the multiple knapsack problem on bipartite graphs to within a constant factor of 11. The relative simplicity of the algorithm enables us to utilize this problem as a stepping-stone to demonstrate the usefulness of the framework for integer packing programs with choices, and the difficulties it overcomes. For instance, in Subsection 5.2, we examine the natural approach of picking the best sub-solution as a proxy to approximately solve the instance, and demonstrate its insufficiency.

Narrow sizes. We study the restricted version of the multiple knapsack problem on bipartite graphs in which the size of every item is *narrow*, that is, $s_i \in (0, 1/2]$, for every $i \in \mathcal{I}$. We begin by presenting a monotone deterministic algorithm, formally described below, that achieves an approximation ratio of 3. Basically, this algorithm is a greedy algorithm w.r.t. a non-increasing *profit density* ratio, that is, a value to size ratio. Note that we use the phrase *knapsack j is feasible w.r.t. item i* to designate a knapsack $j \in M_i$ with a residual capacity of at least s_i .

Algorithm 3 $MKP_N(\mathcal{I})$

Input: An items instance \mathcal{I} in which $s_i \in (0, 1/2]$, for every $i \in \mathcal{I}$

Output: An (item, knapsack) pairs set S of selected items, and their assigned knapsacks

```

1: while  $\mathcal{I} \neq \emptyset$  do
2:   Remove the item  $i$  that has a maximum profit density from  $\mathcal{I}$ 
3:   Let  $j$  be a feasible knapsack w.r.t.  $i$  having a minimal index ( $\infty$  if no knapsack exists)
4:   if  $j \neq \infty$  then
5:     Add  $(i, j)$  to  $S$ 
6:   end if
7: end while
8: return  $S$ 

```

Theorem 5.1. *Let \mathcal{I} be an input instance in which $s_i \in (0, 1/2]$, for every $i \in \mathcal{I}$. Algorithm $MKP_N(\mathcal{I})$ is monotone, deterministic, and outputs a feasible solution whose value is at least $1/3$ of an optimal solution's value.*

Proof. It is easy to see that algorithm MKP_N is deterministic, and outputs a feasible solution. Hence, we are left to prove that it is monotone, and that it obtains the declared approximation guarantee. Let Q be the set of items in an optimal solution. We begin by proving that the value of the solution S is at least $1/3$ of the value of Q . Consider some knapsack j . Let Q_j be the set of items in $Q \setminus S$ that the optimal solution assigns to knapsack j , and let S_j be the set of items in S that the algorithm assigns to knapsack j . Notice that the sole reason that the items in Q_j were not selected by the algorithm resides in the fact that when they were considered, knapsack j was not feasible w.r.t. them. This implies that knapsack j was packed with items whose sum total sizes was greater than $1/2$, which have a profit density ratio that is at least as large as the

profit density ratio of every item in Q_j . Consequently, one can derive that $\sum_{i \in Q_j} v_i \leq 2 \sum_{i \in S_j} v_i$. Hence,

$$\sum_{i \in Q} v_i \leq \sum_{i \in S} v_i + \sum_{i \in Q \setminus S} v_i = \sum_{i \in S} v_i + \sum_{j=1}^m \sum_{i \in Q_j} v_i \leq \sum_{i \in S} v_i + 2 \sum_{j=1}^m \sum_{i \in S_j} v_i = 3 \sum_{i \in S} v_i .$$

We now turn to establish that the algorithm is monotone. Consider an item i , having a value of v_i , that is put by the algorithm in some knapsack. Now, suppose that i had a value of $\tilde{v}_i \geq v_i$, and the values of all the other items were fixed. For the sake of monotonicity, we need to prove that the algorithm would have packed i in some knapsack in the latter case. It is clear that the profit density ratio of i when its value is \tilde{v}_i is at least as large as the profit density ratio when its value is v_i . Hence, the greedy algorithm attends to i priorly when its value is \tilde{v}_i . In particular, this implies that when the algorithm attends to i the "part of the knapsack", which is allotted to i by the algorithm when its value is v_i , is still free. Consequently, the algorithm can pack i in some knapsack since especially it can put it in the aforementioned "part of the knapsack". ■

We now argue that the integrality gap of this variant is at most 3. Essentially, this follows from the insight that the integral solution returned by algorithm MKP_N is a 3-approximation for the optimal fractional solution, and not only for the optimal integral one. We omit the proof of this claim as it goes along the same lines as the approximation analysis in the proof of Theorem 5.1.

Theorem 5.2. *The integrality gap of the restricted version of the problem in which the size of every item is guaranteed to be narrow is at most 3.*

Wide sizes. We turn to inspect the restricted version of the multiple knapsack problem on bipartite graphs in which the size of every item is guaranteed to be *wide*, i.e., $s_i \in (1/2, 1]$, for every $i \in \mathcal{I}$. We begin by demonstrating that there is a monotone deterministic algorithm that attains an optimal outcome for this variant. The key observation one needs to make is that no pair of items can be put in the same knapsack simultaneously. This implies that one may disregard the sizes of the items, and assume that all of them are exactly unit. In consequence, this variant is equivalent to the *maximum weighted matching problem on bipartite graph*, which is known to admit a polynomial-time optimal deterministic algorithm (see, e.g., [1]). If we refer to this algorithm as MKP_W , we obtain the following theorem. Note that the monotonicity of the algorithm directly results from the optimality of the solution.

Theorem 5.3. *Let \mathcal{I} be an input instance in which $s_i \in (1/2, 1]$, for every $i \in \mathcal{I}$. Algorithm $\text{MKP}_W(\mathcal{I})$ is monotone, deterministic, and outputs a feasible solution whose value is optimal.*

Next, we determine the integrality gap of this restricted variant.

Theorem 5.4. *The integrality gap of the restricted version of the problem in which the size of every item is guaranteed to be wide is at most 2.*

Proof. Let I be a wide-sizes input instance, and let I' be an almost identical input instance in which the size and value of every item i is $(1, v_i/s_i)$ instead of (s_i, v_i) . Additionally, let $\text{OPT}(I)$ and $\text{OPT}^*(I)$ be the value of the optimal integral solution and the optimal fractional solution of input instance I , respectively. In the sequel, we prove that $\text{OPT}^*(I) \leq 2\text{OPT}(I)$. Essentially, this result is established by the following three simple claims.

Claim I: $\text{OPT}^*(I) \leq \text{OPT}^*(I')$. Observe that the size of every item in I' is at least as large as its size in I . This implies that any fractional solution to I is a valid fractional solution to I' . The claim follows by noticing that the profit density of every item in both instances is identical.

Claim II: $\text{OPT}^*(I') = \text{OPT}(I')$. Since every item in I' has a size of 1, we can think of I' as an instance of maximum weighted matching problem on bipartite graph. The linear program

formulation of this problem is known to consist of a constraint matrix that is totally unimodular. Hence, every basic solution, e.g., the optimal fractional solution, is in fact integral [39].

Claim III: $\text{OPT}(I') \leq 2\text{OPT}(I)$. Recall that all the items in I have a wide size. Thus, the value of every item in I' is at most two times the value of that item in I . The claim results by observing that the optimal integral solution of I' is a feasible integral solution to I . ■

Integration of the results within the framework. We now illustrate how to integrate the results within the confines of the framework. Let a_1 and a_2 be attributes that select all the narrow-sized items and the wide-sized items, respectively. This collection of attributes partition every input instance of the problem in a pairwise-disjoint choices-consistent way. Furthermore, let $\{\text{MKP}_N, \text{MKP}_W\}$ be the corresponding family of deterministic algorithms. Specifically, MKP_N is a monotone a_1 -restrictive 3-approximation algorithm, and MKP_W is a monotone a_2 -restrictive 1-approximation algorithm. Lastly, recollect that the integrality gap of the a_1 -restrictive variant of the problem is 3, and the integrality gap of the a_2 -restrictive variant is 2. Consequently, and in accordance with Theorem 3.2, we obtain the following theorem.

Theorem 5.5. *There is a monotone deterministic 11-approximation algorithm for the multiple knapsack problem on bipartite graphs.*

5.2 Why the natural approach fails?

In Subsection 5.1, we have developed two monotone deterministic algorithms, which attend to two complementary restricted versions of the multiple knapsack problem on bipartite graphs. A natural approach is to execute these algorithms, and then to pick the best solution. As one may anticipate, this approach achieves provable approximation guarantee. In fact, one can easily demonstrate that the approximation guarantee is equal to the sum of the approximation ratios of the underlying algorithms, i.e., 4-approximation in our case. Unfortunately, this simple approach fails to be monotone. The following theorem provides a concrete example for this insufficiency.

Theorem 5.6. *A procedure that executes MKP_N and MKP_W , and then outputs the solution with the greatest value is not monotone.*

Proof. Suppose we are given an input instance that consists of the following set of items $\mathcal{I} = \{(1/2, 5), (1/2, 5), (1/3, 3), (1/3, 3), (1/3, 3), (1, 18)\}$, and two unit-capacity knapsacks. Furthermore, the restriction graph is a complete bipartite graph, that is, every item can be put in every knapsack. Clearly, the solution returned by MKP_W w.r.t. the corresponding wide-sizes instance has a value of 18. On the other hand, one can validate that the solution generated by MKP_N w.r.t. the corresponding narrow-sizes instance has a value of 19. This solution is schematically described in Figure 3(a). Consequently, the outcome of the procedure that picks the best solution coincides with the outcome of MKP_N . Now, suppose that the value of one of the items having size $1/3$ was 4 instead of 3, and the values of all the other items were fixed. If the procedure that picks the best solution was monotone, this item would still be in the outcome. Unluckily, this is not the case. One can verify that the value of the solution generated by MKP_N w.r.t. the corresponding narrow-sizes instance has a value of 17, as schematically illustrated in Figure 3(b). On the other hand, the value of the solution returned by MKP_W w.r.t. the corresponding wide-sizes instance is still 18. Thus, the solution of the procedure coincides with the outcome of MKP_W . In particular, no item having size $1/3$ is picked. ■

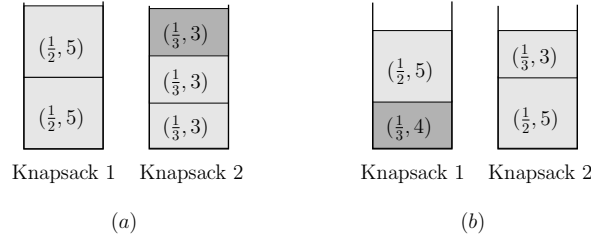


Figure 3: The outcome of MKP_N w.r.t. the two input instance described in the proof. Note that the (s, v) pair inscribed in each block corresponds to the size and value of the item, respectively.

5.3 The multiple knapsack problem

In what follows, we shortly discuss the multiple knapsack problem. Remark that the multiple knapsack problem is a special case of the multiple knapsack problem on bipartite graphs in which the restriction graph is a complete bipartite graph. This specialization enables us to refine the analysis presented in Subsection 5.1 for the narrow-sizes variant. Specifically, we can demonstrate that the approximation guarantee of algorithm MKP_N , as well as the integrality gap of the narrow-sizes variant, is 2. This implies an approximation guarantee of 6 for the problem under consideration. The key observation needed for this refinement is that when algorithm MKP_N ends, there is a subset of the items, which are the most profitable, that together occupy at least half of the capacity of every knapsack.

Acknowledgments

The authors would like to thank Danny Segev for many fruitful discussions.

References

- [1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
- [2] M. Andrews, J. Chuzhoy, S. Khanna, and L. Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. In *Proceedings 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 226–244, 2005.
- [3] M. Andrews and L. Zhang. Hardness of the undirected edge-disjoint paths problem. In *Proceedings 37th Annual ACM Symposium on Theory of Computing*, pages 276–283, 2005.
- [4] A. Archer, C. H. Papadimitriou, K. Talwar, and É. Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. In *Proceedings 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 205–214, 2003.
- [5] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. *Algorithmica*, 31(1):29–43, 2001.
- [6] Y. Azar, I. Gamzu, and S. Gutner. Truthful unsplittable flow for large capacity networks. In *Proceedings 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 320–329, 2007.
- [7] M. Babaioff, R. Lavi, and E. Pavlov. Single-value combinatorial auctions and implementation in undominated strategies. In *Proceedings 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1054–1063, 2006.

- [8] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-ptas for unsplittable flow on line graphs. In *Proceedings 38th ACM Symposium on Theory of Computing*, pages 721–729, 2006.
- [9] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.
- [10] R. Bar-Yehuda, M. Beder, Y. Cohen, and D. Rawitz. Resource allocation in bounded degree trees. In *Proceedings 14th Annual European Symposium on Algorithms*, pages 64–75, 2006.
- [11] P. Briest, P. Krysta, and B. Vöcking. Approximation techniques for utilitarian mechanism design. In *Proceedings 37th ACM Symposium on Theory of Computing*, pages 39–48, 2005.
- [12] G. Calinescu, A. Chakrabarti, H. J. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *Proceedings 9th International Integer Programming and Combinatorial Optimization Conference*, pages 439–456, 2001.
- [13] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. In *Proceedings 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 51–66, 2002.
- [14] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.
- [15] C. Chekuri, M. Mydlarz, and F. B. Shepherd. Multicommodity demand flow in a tree. In *Proceedings 30th International Colloquium on Automata, Languages and Programming*, pages 410–425, 2003.
- [16] J. Chuzhoy, V. Guruswami, S. Khanna, and K. Talwar. Hardness of routing with congestion in directed graphs. In *Proceedings 39th Annual ACM Symposium on Theory of Computing*, pages 165–178, 2007.
- [17] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 8:17–33, 1971.
- [18] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi. Approximation algorithms for the multiple knapsack problem with assignment restrictions. *Journal of Combinatorial Optimization*, 4(2):171–186, 2000.
- [19] T. Erlebach, K. Jansen, C. Kaklamanis, M. Mihail, and P. Persiano. Optimal wavelength routing on directed fiber trees. *Theoretical Computer Science*, 221(1-2):119–137, 1999.
- [20] U. Feige and J. Vondrák. Approximation algorithms for allocation problems: Improving the factor of $1 - 1/e$. In *Proceedings 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 667–676, 2006.
- [21] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 611–620, 2006.
- [22] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
- [23] T. Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973.

- [24] V. Guruswami, S. Khanna, R. Rajaraman, F. B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 67(3):473–496, 2003.
- [25] C. Kaklamanis and G. Persiano. Efficient wavelength routing on directed fiber trees. In *Proceedings 4th Annual European Symposium on Algorithms*, pages 460–470, 1996.
- [26] V. Kumar and E. J. Schwabe. Improved access to optimal bandwidth in trees. In *Proceedings 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 437–444, 1997.
- [27] R. Lavi and C. Swamy. Truthful and near-optimal mechanism design via linear programming. In *Proceedings 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 595–604, 2005.
- [28] D. J. Lehmann, L. O’Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):577–602, 2002.
- [29] L. Lewin-Eytan, J. Naor, and A. Orda. Admission control in networks with advance reservations. *Algorithmica*, 40(4):293–304, 2004.
- [30] M. Mihail, C. Kaklamanis, and S. Rao. Efficient access to optical bandwidth - wavelength routing on directed fiber trees, rings, and trees of rings. In *Proceedings 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 548–557, 1995.
- [31] A. Mu’alem and N. Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. In *Proceedings 18th National Conference on Artificial Intelligence*, pages 379–384, 2002.
- [32] N. Nisan and A. Ronen. Computationally feasible vcg mechanisms. In *Proceedings 2nd ACM conference on Electronic Commerce*, pages 242–252, 2000.
- [33] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.
- [34] Z. Nutov, I. Beniaminy, and R. Yuster. A $(1-1/e)$ -approximation algorithm for the generalized assignment problem. *Operations Research Letters*, 34(3):283–288, 2006.
- [35] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Proceedings 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 879–888, 2000.
- [36] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.
- [37] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.
- [38] P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings 26th ACM Symposium on Theory of Computing*, pages 134–143, 1994.
- [39] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1986.
- [40] A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29(2):648–670, 1999.
- [41] R. E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985.

- [42] W. Vickery. Counterspeculation, auctions and competitive sealed tender. *Journal of Finance*, 16:8–37, 1961.