On Allocating Goods to Maximize Fairness*

Deeparnab Chakrabarty† Julia Chuzhoy‡ Sanjeev Khanna§

September 23, 2009

Abstract

We consider the Max-Min Allocation problem: given a set $\mathbf{A}$ of m agents and a set $\mathbf{I}$ of n items, where agent $A \in \mathbf{A}$ has utility $u_{A,i}$ for item $i \in \mathbf{I}$, our goal is to allocate items to agents so as to maximize fairness. Specifically, the utility of an agent is the sum of its utilities for the items it receives, and we seek to maximize the minimum utility of any agent. While this problem has received much attention recently, its approximability has not been well-understood thus far: the best known approximation algorithm achieves an $\tilde{O}(\sqrt{m})$-approximation, and in contrast, the best known hardness of approximation stands at 2.

Our main result is an algorithm that achieves an $\tilde{O}(n^{\epsilon})$-approximation for any $\epsilon = \Omega(\frac{\log\log n}{\log n})$ in time $n^{O(1/\epsilon)}$. In particular, we obtain poly-logarithmic approximation in quasi-polynomial time, and for every constant $\epsilon > 0$, we obtain an n^{ϵ}-approximation in polynomial time. An interesting technical aspect of our algorithm is that we use as a building block a linear program whose integrality gap is $\Omega(\sqrt{m})$. We bypass this obstacle by iteratively using the solutions produced by the LP to construct new instances with significantly smaller integrality gaps, eventually obtaining the desired approximation.

We also investigate the special case of the problem, where every item has non-zero utility for at most two agents. This problem is hard to approximate up to any factor better than 2. We give a factor 2-approximation algorithm.

*Preliminary version appeared in FOCS 2009, and also as an arXiv technical report, arXiv:0901.0205v1, Jan. 2009

†Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada. Email: deepc@math.uwaterloo.ca.

‡Toyota Technological Institute, Chicago, IL 60637. Email: cjulia@tti-c.org. Supported in part by NSF CAREER award CCF-0844872.

§Dept. of Computer & Information Science, University of Pennsylvania, Philadelphia, PA 19104. Email: sanjeev@cis.upenn.edu. Supported in part by a Guggenheim Fellowship, an IBM Faculty Award, and by NSF Award CCF-0635084.

1 Introduction

We consider the problem of allocating indivisible goods to a set of agents, with the objective to maximize the minimum utility of any agent. Formally, we are given a set $\mathbf{A}$ of m agents, a set $\mathbf{I}$ of n indivisible items, and non-negative utilities $u_{A,i}$ for each agent A and item i. The total utility of an agent A for a subset $S \subseteq \mathbf{I}$ of items is $u_A(S) := \sum_{i \in S} u_{A,i}$, that is, the utility function is additive. An allocation of items is a function $\Phi : \mathbf{A} \to 2^{\mathbf{I}}$ such that every item is allocated to at most one agent, that is, $\Phi(A) \cap \Phi(A') = \emptyset$ whenever $A \neq A'$. The Max-Min Allocation problem is to find an allocation Φ of items which maximizes $\min_{A \in \mathbf{A}} \{u_A(\Phi(A))\}$.

The Max-Min Allocation problem arises naturally in the context of *fair* allocation of indivisible resources where maximizing the utility of the least 'happy' person is arguably a suitable notion of fairness. Woeginger [17] and Epstein and Sgall [11] gave polynomial time approximation schemes (PTAS) for the special case where all agents have identical utilities for the items. Woeginger [18] also gave an FPTAS for the case where the number of agents, m, is a constant. The first non-trivial approximation algorithm for the general Max-Min Allocation problem is due to Bezakova and Dani [6], achieving a factor $(n - m + 1)$-approximation. They also showed that the problem is hard to approximate up to any factor smaller than 2.

Bansal and Sviridenko [3] introduced a *restricted version* of the Max-Min Allocation problem, called the *Santa Claus* problem, where each item has the same utility for a subset of agents and 0 utility for the rest. In other words, for each agent A and item i, either $u_{A,i} = u_i$, or $u_{A,i} = 0$, where the value u_i only depends on the item i. They proposed an LP relaxation for the problem, referred to as the *configuration* LP, and used it to give an $O(\log \log m / \log \log \log m)$-approximation for the Santa Claus problem. Subsequently, Feige [12] showed a constant upper bound on the integrality gap of the configuration LP for the Santa Claus problem. However his proof is non-constructive and does not give an approximation algorithm. More recently, Asadpour, Feige and Saberi [1] provided an alternative non-constructive proof of a factor-5 upper bound on the integrality gap of the LP.

As for the general Max-Min Allocation problem, Bansal and Sviridenko [3] showed that the configuration LP has an integrality gap of $\Omega(\sqrt{m})$ in this setting. Recently, Asadpour and Saberi [2] gave an $O(\sqrt{m} \log^3 m)$ approximation algorithm for the problem using the same LP relaxation. This is the best approximation algorithm known for the problem prior to our work, while the best current hardness of approximation factor is 2 [6]. The main result of our paper is an $\tilde{O}(n^{\epsilon})$-approximation algorithm for any $\epsilon = \Omega(\log \log n / \log n)$ for the general Max-Min Allocation problem, whose running time is $n^{O(1/\epsilon)}$. This implies a quasi-polynomial time poly-logarithmic approximation to the general Max-Min Allocation problem, and for any constant $\epsilon > 0$, gives an n^{ϵ}-approximation in polynomial time. Interestingly, one of our main building blocks is an LP-relaxation whose integrality gap is $\Omega(\sqrt{m})$. We bypass this obstacle by iteratively using the LP solutions to construct a sequence of new instances with diminishing integrality gaps.

We also investigate a special case of Max-Min Allocation where each item has positive utility for at most *two* agents. We call this special case the 2-*restricted* Max-Min Allocation problem. When the two positive utilities are identical for both agents, we call it a *uniform* 2-restricted instance. To the best of our knowledge, prior to our work, the approximability status of the 2-restricted Max-Min Allocation problem has been the same as that of the the general Max-Min Allocation; for the uniform 2-restricted Max-Min Allocation the algorithm and analysis of Bansal and Sviridenko for the Santa Claus problem implies a factor-4 approximation. We give a 2-approximation algorithm for the non-uniform 2-restriced Max-Min Allocation.

Remark: We have recently learned that independently of our work, Bateni, Charikar, and Guruswami [4, 5] showed approximation algorithms for special cases of the MAX-MIN ALLOCATION problem. They consider the setting where all utilities $u_{A,i} \in \{0, 1, \infty\}$. Let G_∞ be the graph whose vertices correspond to items and agents, and there is an edge between an agent A and an item i iff $u_{A,i} = \infty$. They achieve an $O(n^\epsilon)$-approximation in $n^{O(1/\epsilon)}$ time and a polylogarithmic approximation in quasi-polynomial time for the following two special cases: (1) when the degree of every item in G_∞ is at most 2, and (2) when graph G_∞ contains no cycles. They also give a 4-approximation for the 2-restricted MAX-MIN ALLOCATION and show that the uniform 2-restricted MAX-MIN ALLOCATION is NP-hard to approximate to a factor better than 2. Finally, they show that the general MAX-MIN ALLOCATION is equivalent to the 3-restricted MAX-MIN ALLOCATION, where every item has a non-zero utility for at most 3 agents.

Remark: We note that the previous results had m, the number of agents, as the parameter in the approximation factor, while our result is an $\tilde{O}(n^\epsilon)$ approximation. Note that n could be much larger than m. However, as an easy corollary, we get, for any constant $\epsilon > 0$, a factor $O(m^\epsilon)$-approximation algorithm in quasi-polynomial time.

Overview of Results and Techniques. Our main result is summarized in the following theorem:

Theorem 1 *For any $\epsilon \geq \frac{9 \log\log n}{\log n}$, there is an $n^{O(1/\epsilon)}$-time algorithm to compute an $O(n^\epsilon \log n)$-approximate solution for the* MAX-MIN ALLOCATION *problem. In particular, there is an $O(\log^{10} n)$-approximation algorithm whose running time is $n^{O(\frac{\log n}{\log\log n})}$.*

Fix an ϵ such that the desired approximation factor is $\tilde{O}(n^\epsilon)$. Our algorithm starts by guessing the value M of the optimal solution. We then define a class of structured MAX-MIN ALLOCATION instances that we call *canonical instances*, and show that any input instance can be transformed into a canonical one, with an $O(\log n)$ loss in the approximation ratio. In a canonical instance, there are only two types of agents – *heavy agents* whose utility for any item is either 0 or M, and *light agents*. Each light agent has a distinct heavy item for which it has utility M, and for every other item, the utility is either M/t or 0, where $t \geq n^\epsilon$ is a large integer. The items with a utility of M/t for a light agent are referred to as the *light items*. Next we transform the problem of assigning items to agents into a network flow problem, by means of *private items*. Each agent is assigned at most one distinct private item, for which it has utility M. The private item of a light agent is its unique heavy item, and we fix some maximal assignment of remaining items to heavy agents. Of course, if every agent is assigned a private item, we immediately obtain a near-optimal solution. So assume that some agents do not get assigned any private items; such agents are called *terminals*. A key observation is that if the optimal solution value is M, then, given any assignment of private items, there always exists a way of *re-assigning* private items such that every terminal is assigned a heavy item. Re-assignment means a heavy agent "frees" its private item if it gets another heavy item while a light agent frees its private item if it gets t light items. These freed-up private items can then be re-assigned. Thus, given any allocation of private items, we can construct a flow network with the property that there exists an *integral* flow satisfying certain constraints (for instance, out-flow of 1 for light agents implies an in-flow of t). We then design a linear programming relaxation to obtain a feasible fractional flow solution for the above network. Our LP relaxation has size $n^{O(1/\epsilon)}$ when the desired approximation ratio is $\tilde{O}(n^\epsilon)$. However, the integrality gap of the LP relaxation is $\Omega(\sqrt{m})$, and thus directly rounding the LP-solution cannot give a better than $O(\sqrt{m})$ approximation factor. This brings us to another key technical idea. Although the LP has a large integrality gap, we show that we can obtain a better approximation algorithm by performing LP-rounding in phases. In each

phase we solve the LP and run a rounding algorithm to obtain a solution which is *almost feasible*: all terminals get heavy items but some items might be allocated twice. From this almost-feasible solution, we recover a new assignment of private items and hence a new instance of the LP, one that has a much smaller number of terminals than the starting instance. We thus show that in poly$(1/\epsilon)$ phases, either one of these instances will certify that the optimal solution cost is smaller than the guessed value M, or we will get an $\tilde{O}(n^{\epsilon})$-approximation. We note that Theorem 1 can also be used to obtain an approximation in terms of number of agents. We prove the following corollary.

Corollary 1 *For any fixed $\epsilon > 0$, there is a quasi-polynomial time m^{ϵ}-approximation algorithm for* Max-Min Allocation.

Our second result is about 2-restricted Max-Min Allocation instances.

Theorem 2 *There is a 2-approximation algorithm for the non-uniform 2-restricted* Max-Min Allocation *problem.*

The 2-restricted Max-Min Allocation can be cast as an orientation problem on (non-uniformly) weighted graphs. Our main technical lemma is a generalization of Eulerian orientations to weighted graphs. At a high level, we show that the edges of any (non-uniformly) weighted graph can be oriented such that the total weight coming into any vertex w.r.t. the orientation is greater than half the total weight incident on the vertex in the undirected graph minus the maximum weight edge incident on the vertex. Note that in the case of unweighted graphs, these orientations correspond to Eulerian orientations. We also prove that for any $\delta > 0$, even the uniform 2-restricted Max-Min Allocation is NP-hard to approximate to within a factor of $(2 - \delta)$.

Related Work. The Max-Min Allocation problem falls into the broad class of resource allocation problems which are ubiquitous in computer science, economics and operations research. When the resources are divisible, the fair allocation problem, also dubbed as *cake-cutting problems*, has been extensively studied by economists and political scientists with entire books (for example, [7]) written on the subject. However, the *indivisible* case has come into focus only recently.

The complexity of resource allocation problems also depends on the complexity of the utility functions of agents. The utility functions we deal with in this paper are additive – for every agent A, the total utility of a set S of items is simply $u_A(S) := \sum_{i \in S} u_{A,i}$. More general utility functions have been studied in the literature, with two major examples being *submodular utilities*, where for every agent A and any two subsets S, T of items, $u_A(S) + u_A(T) \geq u_A(S \cup T) + u_A(S \cap T)$, and *sub-additive utilities*, where $u_A(S) + u_A(T) \geq u_A(S \cup T)$. Note that submodular utilities are a special case of the sub-additive utilities. Khot and Ponnuswami [14] gave a $(2m - 1)$-approximate algorithm for Max-Min Allocation with sub-additive utilities. Recently, Goemans et.al. [13] using the $\tilde{O}(\sqrt{m})$-approximation algorithm of Asadpour and Saberi [2] as a black box, gave a $\tilde{O}(\sqrt{n}m^{1/4})$-approximation for Max-Min Allocation with submodular utilities. We note that using our main theorem above, the algorithm of [13] gives a $\tilde{O}(n^{1/2+\epsilon})$-approximation for submodular Max-Min Allocation in time $n^{O(1/\epsilon)}$. We remark here that nothing better than the factor 2 hardness is known for Max-Min Allocation even with the general sub-additive utilities.

Max-Min Allocation may be viewed as a dual problem to the minimum makespan machine scheduling. Lenstra, Shmoys and Tardos [15] gave a factor 2-approximation algorithm for the problem, and also showed the problem is NP-hard to approximate to a factor better than 3/2. Closing this gap has been one of the challenging problems in the field of approximation algorithms. Recently

Ebenlendr et.al. [10] studied a very restricted setting where each job can only be assigned to two machines, and moreover it takes the same time on both. For this case, authors give a factor (7/4)-approximation algorithm, and show that even this special case is NP-hard to approximate better than a factor 3/2. Our investigation of the 2-restricted Max-Min Allocation is inspired by [10].

2 Preliminaries

We are given a set $\mathbf{A}$ of m agents, a set $\mathbf{I}$ of n indivisible items, and non-negative utility $u_{A,i}$ for each agent A and item i. The utility of agent A for a subset $S \subseteq \mathbf{I}$ of items is $u_A(S) := \sum_{i \in S} u_{A,i}$. The Max-Min Allocation problem is to find an allocation $\Phi : \mathbf{A} \to 2^{\mathbf{I}}$ of items to agents which maximizes $\min_{A \in \mathbf{A}} \{u_A(\Phi(A))\}$, while $\Phi(A) \cap \Phi(A') = \emptyset$ for $A \neq A'$. Given an allocation Φ, we let $\min_{A \in \mathbf{A}} \{u_A(\Phi(A))\}$ denote the value of the allocation. For an agent A and an item i, we use interchangeably the phrases "A has utility γ for item i" and "item i has utility γ for A" to indicate that $u_{A,i} = \gamma$. We say that item i is *wanted* by agent A iff $u_{A,i} > 0$.

We assume that we are given $\epsilon \geq 9 \log\log n / \log n$, and so $n^\epsilon \geq \Omega(\log^9 n)$. Our goal is to produce an $\tilde{O}(n^\epsilon)$-approximate solution in $n^{O(1/\epsilon)}$ time. We use M to denote the (guessed) value of the optimal solution. Our algorithm either produces a solution of value at least $M/\tilde{O}(n^\epsilon)$, or returns a certificate that $M > \mathsf{OPT}$.

In what follows in this section, we perform a series of simplifying operations and convert the Max-Min Allocation problem to an equivalent (up to losing certain factor in the approximation ratio) problem of finding a collection of paths in a directed network satisfying certain properties.

Polynomially Bounded Utilities. We first show that we can assume w.l.o.g. that all utilities are polynomially bounded. We give a simple transformation that ensures that each non-zero utility value is between 1 and $2n$, with at most a factor 2 loss in the optimal value. We can assume that any non-zero utility value is at least 1; otherwise, we can scale up all utilities appropriately, and that the maximum utility is at most M (the optimal solution value). For each agent A and item i, we define its new utility as follows. If $u_{A,i} < M/2n$ then $u'_{A,i} = 0$; otherwise

$$u'_{A,i} = u_{A,i} \cdot \frac{2n}{M}.$$

Since the optimal solution value in the original instance is M, the optimal solution value in the new instance at most $2n$. Moreover, it is easy to see that this value is at least n: consider any agent A and the subset $\mathbf{S}$ of items assigned to A by OPT. The total utility of $\mathbf{S}$ for A is at least M, and at least $M/2$ of the utility is received from items i for which $u_{A,i} \geq M/2n$. Therefore, the new utility of set $\mathbf{S}$ for A is at least n. It is easy to see that any α-approximate solution to the transformed instance implies a (2α)-approximate solution to the original instance.

Canonical Instances. It will be convenient to work with a structured class of instances that we refer to as *canonical instances*. Given $\epsilon = \Omega(\log\log n / \log n)$ and M, we say that an instance $\mathcal{I}$ of Max-Min Allocation is ϵ-*canonical*, or simply, *canonical* iff:

- All agents are partitioned into two sets, a set $\mathbf{L}$ of light agents and a set $\mathbf{H}$ of heavy agents.
- Each heavy agent $A \in \mathbf{H}$ has a subset $\Gamma_{\mathrm{H}}(A)$ of items, where for each $i \in \Gamma_{\mathrm{H}}(A)$, $u_{A,i} = M$, and for each $i \notin \Gamma_{\mathrm{H}}(A)$, $u_{A,i} = 0$.

- Each light agent $A \in \mathbf{L}$ is associated with
  - a *distinct* item $h(A)$ that has utility M for A and is referred to as the *heavy item* for A; if $A \neq A'$ then $h(A) \neq h(A')$,
  - a parameter $N_A \geq n^\epsilon$, and
  - a set $\Gamma_{\mathrm{L}}(A)$ of items, referred to as the *light items* for A. Each item in $\Gamma_{\mathrm{L}}(A)$ has a utility of M/N_A for A. If $i \notin \Gamma_{\mathrm{L}}(A) \cup \{h(A)\}$ then $u_{A,i} = 0$.

Given an assignment of items to agents in the canonical instance, we say that a heavy agent A is *satisfied* iff it is assigned one of the items in $\Gamma_{\mathrm{H}}(A)$, and we say that a light agent A is α-*satisfied* (for some $\alpha \geq 1$) iff it is either assigned item $h(A)$, or it is assigned at least N_A/α items from the set $\Gamma_{\mathrm{L}}(A)$. In the latter case we say that A is *satisfied by light items*. A solution is called α-*approximate* iff all heavy agents are satisfied and all light agents are α-satisfied. A canonical instance is 1-satisfiable if there is an assignment satisfying all heavy agents and 1-satisfies all light agents. The next lemma shows that we can restrict our attention to canonical instances, at the cost of losing a logarithmic factor in the approximation ratio.

Lemma 1 *Given $\epsilon = \Omega(\log\log n/\log n)$ and an instance $\mathcal{I}$ of the* Max-Min Allocation *problem with optimal solution value M, we can produce in polynomial time a canonical instance $\mathcal{I}'$ such that $\mathcal{I}'$ has a solution that 1-satisfies all agents. Moreover, any α-approximate solution to $\mathcal{I}'$ can be converted into a $\max\{O(\alpha\log n), O(n^\epsilon \log n)\}$-approximate solution to $\mathcal{I}$.*

Proof: Given an instance $\mathcal{I}$ of the Max-Min Allocation problem, we create a canonical instance $\mathcal{I}'$ as follows. Define $s = \lfloor \log(M/(n^\epsilon \log n)) \rfloor$. Recall that $M \leq 2n$, so $s \leq \log n$ for large enough n. For each agent in $\mathcal{I}$, the canonical instance $\mathcal{I}'$ will contain $2s+1$ new agents, for a total of $m(2s+1)$ agents. Let $\mathbf{I}$ be the set of items in $\mathcal{I}$. The set $\mathbf{I}'$ of items for the instance $\mathcal{I}'$ will contain $\mathbf{I}$ as well as $2ms$ additional items that we define later.

Specifically, for each agent B in $\mathcal{I}$, we create the following collection of new agents and items:

- A heavy agent $H_0(B)$ and s light agents $L_1(B), \ldots, L_s(B)$ where each light agent $L_j(B)$ is associated with value $N_{L_j(B)} = M/(s \cdot 2^j) \geq M/(s \cdot 2^s) \geq n^\epsilon$.
- For each $j \in \{1, \ldots, s\}$, if the utility of item $i \in \mathbf{I}$ for B is $2^{j-1} \leq u_{B,i} < 2^j$, then agent $L_j(B)$ has utility $s \cdot 2^j = M/N_{L_j(B)}$ for i. If $i \in \mathbf{I}$ is an item for which $u_{B,i} \geq 2^s$, then $H_0(B)$ has utility M for item i.
- Additionally, for each light agent $L_j(B)$ there is a heavy item $h(L_j(B))$, which has utility M for it, and also a heavy agent $H_j(B)$, who has utility M for $h(L_j(B))$.
- Finally, we have a set of s items $Y_B = \{i_1(B), \ldots, i_s(B)\}$ such that each item in Y_B has utility M for each of the agents in $\{H_0(B), H_1(B), \ldots, H_s(B)\}$, the set of heavy agents for B.

This completes the definition of the canonical instance $\mathcal{I}'$.

Now let Φ be an optimal allocation for the instance $\mathcal{I}$. We will use Φ to construct an allocation Φ' for $\mathcal{I}'$ that 1-satisfies all agents. Consider some agent B in the original instance. The optimal solution Φ assigns a set $\Phi(B)$ of items to B. We can partition $\Phi(B)$ into $(s+1)$ sets $\varphi_0, \varphi_1, ..., \varphi_s$,

as follows. The set $\varphi_0 \subseteq \Phi(B)$ contains all items i with $u_{B,i} \geq 2^s$. For each $j \in \{1, \ldots, s\}$, set φ_j contains all items i with $2^{j-1} \leq u_{B,i} < 2^j$. Assume first that $\varphi_0 \neq \emptyset$, and let i be any item in φ_0. Then we assign item i to heavy agent $H_0(B)$. The remaining s heavy agents corresponding to B are assigned one item from Y_B each. The light agents $L_j(B)$ are assigned their heavy items $h(L_j(B))$. All agents corresponding to B are now 1-satisfied. Assume now that $\varphi_0 = \emptyset$. Then there is a $j \in \{1, \ldots, s\}$, such that $u_B(\varphi_j) \geq M/s$. Since each item in φ_j has utility at most 2^j for B, we have that $|\varphi_j| \geq M/(s \cdot 2^j) = N_{L_j(B)}$. We assign all items in φ_j to $L_j(B)$, and $h(L_j(B))$ is assigned to the heavy agent $H_j(B)$. Now the remaining s heavy agents are assigned one item of Y_B each. For each one of the remaining light agents, we assign $h(L_{j'}(B))$ to $L_{j'}(B)$. Therefore, the canonical instance has a solution that 1-satisfies all agents.

Conversely, consider now any α-approximate solution Φ' for the canonical instance $\mathcal{I}'$. Let B be some agent in the original instance. Consider the corresponding set of heavy agents $H_0(B), \ldots, H_s(B)$. Since there are only s items in the set Y_B, at least one of the heavy agents is not assigned an item from this set. Assume first that it is the heavy agent $H_0(B)$. Then it must be assigned some item $i \in \mathbf{I}$ for which agent B has utility at least $2^s \geq M/(2n^\epsilon \log n)$. We then assign item i to agent B. Otherwise, assume it is $H_j(B)$, for $j \neq 0$ that is not assigned an item from Y_B. Then $H_j(B)$ is assigned item $h(L_j(B))$, and so $L_j(B)$ must be assigned a set S' of at least $N_{L_j(B)}/\alpha = M/(s \cdot 2^j \cdot \alpha)$ items, each of which has a utility of at least 2^{j-1} for B. We then assign the items in S' to B. Since $s \leq \log n$, this set has utility at least $M/(2\alpha \log n)$ for B. Thus, we obtain a $\max\{O(n^\epsilon \log n), O(\alpha \log n)\}$-approximate solution. □

From now on, we assume that we are working with a canonical instance.

Private Items and Flow Solutions. One of the basic concepts of our algorithm is that of private items and flow solutions defined by them. Throughout the algorithm we maintain a partial assignment π of private items to agents; partial in the sense that not all agents will be assigned private items. We will always maintain the property that no two agents share a private item. Furthermore, a private item of an agent, if defined, will give utility M to the agent. Such an assignment is called *good* iff for *every* light agent $A \in \mathbf{L}$, its private item is $\pi(A) = h(A)$. We denote by $\mathbf{S}$ the set of items that do not serve as private items. The set of heavy agents that have private items is denoted by $\mathbf{H}_{\mathsf{pvt}}$, and the remaining heavy agents are called *terminals* and are denoted by $\mathbf{T}$.

The initial assignment π of private items to heavy agents is obtained as follows. We create a bipartite graph $G = (U, V, E)$, where $U = \mathbf{H}$, V is the set of items that do not serve as private items for light agents, and E contains an edge between $A \in U$ and $i \in V$ iff $i \in \Gamma_{\mathrm{H}}(A)$. We compute a maximum matching in G that determines the assignment of private items to heavy agents. To simplify notation, we say that for a terminal $A \in \mathbf{T}$, $\pi(A)$ is undefined and $\{\pi(A)\} \triangleq \emptyset$.

The Flow Network. Given a canonical instance $\mathcal{I}$ and an assignment π of private items, we define the corresponding **directed** flow network $N(\mathcal{I}, \pi)$ as follows. The set of vertices is $\mathbf{A} \cup \mathbf{I} \cup \{s\}$. Source s connects to every vertex i where $i \in \mathbf{S}$. If agent $A \in \mathbf{A}$ has a private item and $i = \pi(A)$, then vertex A connects to vertex i. If A is a heavy agent and $i \in \Gamma_{\mathrm{H}}(A) \setminus \{\pi(A)\}$, then vertex i connects to vertex A. If A is a light agent and $i \in \Gamma_{\mathrm{L}}(A)$ then vertex i connects to vertex A. Note that every agent A has out-degree at most 1 and every item has in-degree at most 1. Let $N(\mathcal{I}, \pi)$ denote the resulting network. We have the following constraints on the flow in this network.

C1. All flow originates at the source s.

C2. Each terminal agent $A \in \mathbf{T}$ receives one flow unit.

C3. For each heavy agent $A \in \mathbf{H}$, if the outgoing flow is 1 then the incoming flow is 1; otherwise both are 0.

C4. For each item $i \in \mathbf{I}$, if the outgoing flow is 1 then the incoming flow is 1; otherwise both are 0.

C5. For each light agent $A \in \mathbf{L}$, if the outgoing flow is 1 then the incoming flow is exactly N_A.

An *integral flow* obeying the above conditions is called a *feasible flow*. We say that a flow is α-*feasible* iff constraints (C1)–(C4) hold, while constraint (C5) is relaxed as below:

C6. For each light agent $A \in \mathbf{L}$, if the outgoing flow is 1 then the incoming flow is at least N_A/α.

The next lemma shows that the problem of α-satisfying all agents is equivalent to the problem of finding an α-feasible flow.

Lemma 2 *Let $\pi : \mathbf{A} \setminus \mathbf{T} \to \mathbf{I}$ be any good assignment of private items. An assignment that 1-satisfes the agents of $\mathcal{I}$ implies a feasible flow in $N(\mathcal{I}, \pi)$. Moreover, an integral α-feasible flow in $N(\mathcal{I}, \pi)$ implies an α-approximate solution for the canonical instance $\mathcal{I}$.*

Proof: Fix a 1-satisfying assignment to agents of $\mathcal{I}$. We assume w.l.o.g. that all items in $\mathbf{I} \setminus \mathbf{S}$ are assigned: otherwise if $i \in \mathbf{I} \setminus \mathbf{S}$ is not assigned by the solution, we can assign it to the unique agent $A \in \mathbf{A}$ such that $\pi(A) = i$, and un-assign all the items that were originally assigned to A. Let $A \in \mathbf{H}$ be a heavy agent. If it is assigned an item $i \neq \pi(A)$, then we send one flow unit from vertex i to vertex A. If $A \notin \mathbf{T}$, then it also sends one flow unit to $\pi(A)$. Consider now a light agent $A \in \mathbf{L}$. If it is not assigned $\pi(A)$, then there is a subset $S' \subseteq \Gamma_{\mathrm{L}}(A)$ of N_A items assigned to A. Each of these items sends one flow unit to A, and A sends one flow unit to $\pi(A)$. Finally, each item $i \in \mathbf{S}$ that participates in the assignment receives one flow unit from s. It is easy to see that this is a feasible flow.

For the second part, fix any integral α-feasible flow. Consider any agent A that may be heavy or light. We simply assign to A every item that sends flow to it. If there is no such item, we assign $\pi(A)$ to A. It is easy to verify that this is an α-approximate solution, since every terminal receives one flow unit and all other agents that do not have any flow going through them can be assigned their private items. □

Let $\mathbf{I}^*$ be the set of items and let $\mathbf{H}^*$ be the set of heavy agents reachable from s by a path that does not contain light agents. A useful property of our initial assignment of private items is that $\mathbf{H}^*$ does not contain any terminals (otherwise we could have increased the matching). Throughout the algorithm, the assignment of private items to $\mathbf{H}^*$ does not change, and the above property is preserved. Given any pair v, v' of vertices, we say that a path p which starts at v and ends at v', connects v to v' *directly* if it does not contain any light agents as intermediate vertices (however we allow v and v' to be light agents under this definition). We say that v is *directly connected* to v' if such a path exists. Similarly, given an integral flow solution, we say that v sends flow directly to v' iff the flow is sent via a path p that does not contain light agents as intermediate vertices.

An Equivalent Formulation. A simple path p is called an *elementary path* iff it does not contain any light agents as intermediate vertices, and either a) originates and terminates at light agents, or b) originates at a light agent and terminates at a terminal, or c) originates at the source s and

terminates at a light agent. It is easy to verify that the problem of finding an α-feasible flow is *equivalent* to finding a collection $\mathcal{P}$ of elementary paths such that:

- $\hat{\text{C}}1$. All paths in $\mathcal{P}$ are internally-disjoint (i.e. they do not share intermediate vertices).
- $\hat{\text{C}}2$. Each terminal has exactly one path $p \in \mathcal{P}$ terminating at it.
- $\hat{\text{C}}3$. For each light agent $A \in \mathbf{L}$, there is at most one path $p \in \mathcal{P}$ starting at A, and if such a path exists, then there are at least N_A/α paths in $\mathcal{P}$ terminating at A.

In the next two sections, we prove the following theorem. As we show below, this implies our main theorem, Theorem 1.

Theorem 3 *For any $\epsilon \geq \frac{9 \log\log n}{\log n}$, given an ϵ-canonical instance $\mathcal{I}$, there is an algorithm running in time $n^{O(1/\epsilon)}$, which either shows that $\mathcal{I}$ is not 1-satisfiable; or finds a good assignment π of private items, and finds a collection of paths $\mathcal{P}$ in $N(\mathcal{I}, \pi)$ satisfying ($\hat{\text{C}}1$),($\hat{\text{C}}2$) and ($\hat{\text{C}}3$), with $\alpha = O(\frac{\log n}{\epsilon^6})$.*

From this theorem and the discussions preceding that, we can prove Theorem 1.

Proof of Theorem 1: Let M be the largest value for which the ϵ-canonical instance, $\mathcal{I}$, is 1-satisfiable. Lemma 1 shows that $M \geq \mathsf{OPT}$. Theorem 3, Lemma 2 and the discussion subsequent to it gives an α-approximate solution to $\mathcal{I}$. Using Lemma 1 we get an $\max\{O(n^\epsilon \log n), O(\alpha \log n)\} = O(n^\epsilon \log n)$ (when $\epsilon \geq \frac{9 \log\log n}{\log n}$) factor approximation for the Max-Min Allocation problem. □

3 Almost Feasible Solutions via an LP relaxation

In this section and the next, we prove Theorem 3. We start, in this section, with an LP relaxation for the problem of finding elementary paths satisfying properties ($\hat{\text{C}}1$)–($\hat{\text{C}}3$) and then present a rounding algorithm. The LP relaxation is feasible whenever $\mathcal{I}$ is 1-satisfiable. Our rounding algorithm, however, does not produce a feasible solution. Instead, we obtain an *almost feasible* solution in that ($\hat{\text{C}}2$) and ($\hat{\text{C}}3$) hold, but for some paths and some heavy agents there may be *two* elementary paths containing them. Similarly, a light agent A could have two elementary paths originating at A: one terminating at another light agents, one terminating at a terminal.

The fact that we are unable to obtain a feasible solution is not surprising: the LP that we construct has an $\Omega(\sqrt{m})$ integrality gap, as we show in Section A in the Appendix. Surprisingly we manage to bypass this obstacle in our final algorithm presented in Section 4, and are able to complete the proof of Theorem 3. The following theorem summarizes the properties of the almost feasible solution that we obtain.

Theorem 4 *Let $\mathcal{I} = (\mathbf{A}, \mathbf{I})$ be any 1-satisfiable canonical instance, and let $\pi : \mathbf{A} \setminus \mathbf{T} \to \mathbf{I}$ be a good assignment of private items to non-terminal agents, such that $N(\mathcal{I}, \pi)$ does not contain direct paths from source s to any terminal. Let $\alpha = O(h^5 \log n)$. Then we can find, in $n^{O(1/\epsilon)}$ time, two collections $\mathcal{P}_1$ and $\mathcal{P}_2$ of elementary paths in $N(\mathcal{I}, \pi)$ with the following properties.*

- *D1. All paths in $\mathcal{P}_1$ terminate at the terminals and all paths in $\mathcal{P}_2$ terminate at light agents. Moreover, each terminal lies on exactly one path in $\mathcal{P}_1$.*

D2. All paths in $\mathcal{P}_1$ are completely vertex disjoint, and paths in $\mathcal{P}_2$ are internally vertex-disjoint but may share endpoints. A non-terminal agent or an item may appear in both $\mathcal{P}_1$ and $\mathcal{P}_2$.

D3. For each light agent $A \in \mathbf{L}$, there is at most one path in $\mathcal{P}_1$ and at most one path in $\mathcal{P}_2$ that originates at A (so, in total, there may be two paths in $\mathcal{P}_1 \cup \mathcal{P}_2$ originating at A).

D4. If there is a path $p \in \mathcal{P}_1 \cup \mathcal{P}_2$ originating at some light agent $A \in \mathbf{L}$, then at least N_A/α paths in $\mathcal{P}_2$ terminate at A.

The rest of this section is devoted to the proof of Theorem 4. We start by an observation on the structure of the optimal solutions in Section 3.1. Next, we present in Section 3.2 a useful technical tool we call BS tree decomposition, that was shown by Bansal and Sviridenko [3]; its proof is presented here for completeness. In Section 3.3 we show how to obtain almost feasible solutions for a restricted simple class of instances, which immediately gives us an $\tilde{O}(\sqrt{n})$-approximation for the whole problem. This section is not a part of the proof of Theorem 4, and we mostly present it to demonstrate some of the ideas used in our main algorithm. Finally, in Section 3.4 we prove Theorem 4.

3.1 Structured Near-Optimal Solutions for Canonical Instances.

Given a canonical instance $\mathcal{I}$ and a good assignment π of private items, an optimal solution OPT to $\mathcal{I}$ defines a feasible integral flow in $N(\mathcal{I}, \pi)$. Consider graph G obtained from $N(\mathcal{I}, \pi)$ as follows: we remove the source s and all its adjacent edges. We also remove all edges that do not carry any flow in OPT, and finally remove all isolated vertices. The resulting graph is then a collection of disjoint trees. Each such tree τ has a terminal $t \in T$ as its root, that has one incoming edge. Each heavy agent in the tree has one incoming and one outgoing edge, and each light agent A has N_A incoming edges and one outgoing edge. The leaves are items in $\mathbf{S}$. Such a solution is called an *h-layered forest* iff for every tree τ, the number of light agents on any leaf-to-root path is the same; we denote it by $h(\tau)$. Moreover, $h(\tau) \leq h$ for all trees τ. It is convenient to work with layered solutions, and we now show that for any canonical instance, there exists an $(h+1)$-approximate h-layered solution for $h = 9/\epsilon$.

Lemma 3 *For every ϵ-canonical instance $\mathcal{I}$, there is an h-layered forest that defines an $(h+1)$-approximate solution, for $h = 9/\epsilon$ and $\epsilon \geq \frac{9 \log\log n}{\log n}$.*

Proof: We will start with an optimal solution OPT for $\mathcal{I}$, and convert it into an h-layered solution in which every light agent will be $(h+1)$-satisfied. Consider any tree τ in the collection of disjoint trees in the graph G corresponding to OPT. We transform it into an h-layered tree in h iterations.

A light agent $A \in \mathbf{L}$ that belongs to τ is called a *level-1 agent* iff it receives at least $N_A/(h+1)$ flow units directly from items in $\mathbf{S}$. Let $\mathbf{L}_1(\tau)$ be the set of all level-1 light agents of τ. Consider some agent $A \in \mathbf{L}_1(\tau)$. For each child v of A in τ, if v does not lie on an elementary path connecting an item of $\mathbf{S}$ to A, then we remove v together with its sub-tree from τ.

In general, for $j > 1$, a light agent A is a *level-j agent* iff it does not belong to levels $1, \ldots, j-1$ and it receives at least $N_A/(h+1)$ flow units directly from level-$(j-1)$ agents. In iteration j, we consider the set $\mathbf{L}_j(\tau)$ of level-j agents. Let $A \in \mathbf{L}_j(\tau)$ be any such agent, and let v be any child of A in τ. If v lies on a elementary path p connecting some level-$(j-1)$ agent to A in τ, then we do nothing. Otherwise, we remove v and its subtree from τ. We claim that after iteration h is completed, all

remaining light agents in τ belong to $\mathcal{L} = \cup_{j=1}^{h} \mathbf{L}_j(\tau)$. Thus we can convert every tree τ into an h-layered tree, obtaining a $(h+1)$-approximate h-layered solution.

Claim 1 *After h iterations, every light agent A remaining in τ belongs to $\mathcal{L}$.*

Proof: Assume otherwise. Then we can find a light agent $A \notin \mathcal{L}$ in τ, such that all light agents in the sub-tree of A belong to $\mathcal{L}$ (Start with arbitrary agent A remaining in τ, such that $A \notin \mathcal{L}$. If there is a light agent A' in the subtree of A that does not belong to $\mathcal{L}$, then continue with A'. When this process stops we have agent A as required).

Agent A receives at least N_A flow units in the original tree, but it does not belong to $\mathbf{L}_j(\tau)$ for $1 \leq j \leq h$. That is, it receives less than $N_A/(h+1)$ flow units directly from light agents in $\mathbf{L}_j(\tau)$ for $1 \leq j \leq h-1$ and less than $N_A/(h+1)$ flow units directly from $\mathbf{S}$. It follows that it must receive at least $N_A/(h+1) \geq n^{\epsilon}/(h+1)$ units of flow directly from agents in $\mathbf{L}_h$. Each one of these agents receives at least $n^{\epsilon}/(h+1)$ flow from agents in $\mathbf{L}_{h-1}(\tau)$ and so on. So for each $j : 1 \leq j \leq h$, the sub-tree of A contains at least $(n^{\epsilon}/(h+1))^{h-j+1}$ agents from $\mathbf{L}_j(\tau)$, and in particular it contains $(n^{\epsilon}/(h+1))^h$ agents from $\mathbf{L}_1(\tau)$. We now show that by our choice of ϵ and h, $(n^{\epsilon}/(h+1))^h > m$ which would be a contradiction.

Recall that $n^{\epsilon} \geq \log^9 n$ and $1 > \epsilon \geq 9 \log\log n / \log n$. So $9 \leq (h = 9/\epsilon) \leq \log n / \log\log n$. Thus, for n large enough $(h+1) \leq 2h \leq 2\log n/(\log\log n) \leq \log n \leq n^{\epsilon/9}$ giving us $(n^{\epsilon}/(h+1))^h \geq (n^{8\epsilon/9})^h \geq n^8 \geq n > m$. □ □

From now on we can focus on h-layered instances. For simplicity, we scale down all values N_A for $A \in \mathbf{A}$ by a factor of $(h+1)$, so that an optimal h-layered solution can 1-satisfy all agents. Note that this increases the approximation ratio of our algorithm by a factor of $(h+1)$. Our LP relaxation will be feasible if there exists a h-layered solution which 1-satisfies all agents.

3.2 The BS Tree Decomposition

One of the tools we use in our algorithm is a tree-decomposition theorem of Bansal and Sviridenko [3]. We remark that this theorem has no connection to the trees induced in the flow network $N(\mathcal{I}, \pi)$ by a feasible solution. The setup for the theorem is the following. We are given an **undirected** bipartite graph $G = (\mathbf{A}, \mathbf{I}, E)$ where $\mathbf{A}$ is a set of agents, $\mathbf{I}$ is a set of items and E contains an edge (A, i) iff A has utility M for i. Additionally, every agent A is associated with a value $0 \leq x_A \leq 1$. (We can think of x_A as the extent to which A is satisfied by light items in a fractional solution). We are also given a fractional assignment $y_{A,i}$ of items, such that:

$$\forall i \in I \qquad \sum_{A \in \mathbf{A}} y_{A,i} \leq 1 \tag{1}$$

$$\forall A \in \mathbf{A} \qquad \sum_{(A,i) \in E} y_{A,i} = 1 - x_A \tag{2}$$

$$\forall (A,i) \in E \qquad 0 \leq y_{A,i} \leq 1 \tag{3}$$

The theorem of [3] shows that such an assignment can be decomposed into a collection of disjoint trees in graph G. For each such tree τ, the summation of values x_A for agents in τ is at least $\frac{1}{2}$,

and moreover if $\mathbf{A}(\tau)$ is the set of agents of τ and $\mathbf{I}(\tau)$ is the set of items of τ, then for each agent $A \in \mathbf{A}(\tau)$, it is possible to satisfy all agents in $\mathbf{A}(\tau) \setminus \{A\}$ by items in $\mathbf{I}(\tau)$.

Theorem 5 *([3]) There exists a decomposition of $G = (\mathbf{A}, \mathbf{I}, E)$ into a collection of disjoint trees $T_1, T_2, \ldots, T_s$, such that, for each tree T_j, either (1) T_j contains a single edge between some item i and some agent A, or (2) the degree of each item $i \in \mathbf{I}(T_j)$ is 2 and $\sum_{A \in \mathbf{A}(T_j)} x_A > 1/2$.*

Corollary 2 *For each tree T_j in the decomposition, for each agent $A \in \mathbf{A}(T_j)$, it is possible to satisfy all agents in $\mathbf{A}(T_j) \setminus \{A\}$ by items in $\mathbf{I}(T_j)$.*

Proof: Root tree T_j at vertex A. Now every agent $A' \neq A$ is assigned item i, where i is the parent of A' in T_j. Since the degree of every item is at most 2, this is a feasible assignment. □

Proof: (of Theorem 5) We remove from E all edges $(A, i) \in E$ with $y_{A,i} = 0$. Let $E^* \subseteq E$ be the set of edges (A, i) with $y_{A,i} = 1$. We remove from G edges in E^* together with their endpoints. These will be the trees of type (1) in the statement of the theorem. We now perform the following three steps.

Step 1: Converting G into a forest. We will transform values $y_{A,j}$ so that the set of edges with non-negative values $y_{A,j}$ forms a forest, while preserving constraints (1)–(3), as follows. Suppose there is a cycle $\mathcal{C}$ induced by edges of E. Since the cycle is even (the graph being bipartite), it can be decomposed into two matchings M_1 and M_2. Suppose the smallest value $y_{A,i}$ for any edge $(A, i) \in M_1 \cup M_2$ is δ, and assume w.l.o.g. that this edge lies in M_1. For each $(A, i) \in M_2$, we increase $y_{A,i}$ by δ and for each $(A, i) \in M_1$ we decrease $y_{A,i}$ by δ. It is easy to see that constraints (1)–(3) continue to hold, and at least one edge $(A, i) \in M_1 \cup M_2$ has value $y_{A,i} = 0$. All edges (A, i) with $y_{A,i} = 0$ are then removed from E, and all edges (A, i) with $y_{A,i} = 1$ are added to E^* with A and i removed from G. We can continue this process until no cycles remain in G.

We now fix some tree τ in G. While there exists an item $i \in \mathbf{I}(\tau)$ of degree 1, we perform Step 2.

Step 2: If there is an item i in τ with degree 1, then let A be the unique agent with $(A, i) \in E$. We set $y_{A,i} = 1 - x_A$ and $y_{A,i'} = 0$ for all $i \neq i'$. Notice that constraints (1)–(3) continue to hold. We then add (A, i) to E^*, removing the edge and both its endpoints from G giving a tree of type (i) in the theorem statement.

Assume now that the degree of every item $i \in \mathbf{I}(\tau)$ is 2. Clearly then $|\mathbf{I}(\tau)| = |\mathbf{A}(\tau)| - 1$. Then $\sum_{A \in \mathbf{A}(\tau)} \sum_{i \in \mathbf{I}(\tau)} y_{A,i} \leq |\mathbf{I}(\tau)| = |\mathbf{A}(\tau)| - 1$. On the other hand, $\sum_{A \in \mathbf{A}(\tau)} \sum_{i \in \mathbf{I}(\tau)} y_{A,i} = \sum_{A \in \mathbf{A}(\tau)} (1 - x_A) = |\mathbf{A}(\tau)| - \sum_{A \in \mathbf{A}(\tau)} x_A$. Therefore, $\sum_{A \in \mathbf{A}(\tau)} x_A \geq 1$. We add τ to the decomposition.

Otherwise, while there is an item in $\mathbf{I}(\tau)$ of degree greater than 2, we perform Step 3:

Step 3: Root tree τ at an arbitrary vertex in $\mathbf{I}(\tau)$. Let $i \in \mathbf{I}(\tau)$ be a vertex of degree at least 3, such that in the sub-tree of i all items have degree 2. Now consider the children of i, denoted by $A_1, A_2, \ldots, A_r$, with $r \geq 2$. Note that there is $j : 1 \leq j \leq r$ with $y_{A_j,i} < 1/2$. Remove the edge (A_j, i) from G, and add the sub-tree τ' rooted at A_j to the decomposition. Note that all item vertices in this sub-tree has degree exactly 2. Also note that:

$\sum_{A' \in \mathbf{A}(\tau')} \sum_{i' \in \mathbf{I}(\tau')} y_{A',i'} \leq |\mathbf{I}(\tau')| = |\mathbf{A}(\tau')| - 1$, while on the other hand $\sum_{A' \in \mathbf{A}(\tau')} \sum_{i' \in \mathbf{I}(\tau')} y_{A',i'} = \sum_{A' \in \mathbf{A}(\tau')} (1 - x_{A'}) - y_{A_j,i} = |\mathbf{A}(\tau')| - \sum_{A' \in \mathbf{A}(\tau')} x_{A'} - y_{A_j,i}$.

Therefore, $\sum_{A' \in \mathbf{A}(\tau')} x_{A'} \geq 1 - y_{A_j,i} \geq \frac{1}{2}$. □

3.3 An $\tilde{O}(\sqrt{n})$-Approximation via 1-Layered Instances

In this section, as a warm-up towards the eventual goal of computing approximate h-layered solutions, we consider the special case of finding an optimal 1-layered solution, that is, we restrict solutions to trees τ with $h(\tau)$=1. We design an LP relaxation and a rounding scheme for the relaxation, to obtain an $O(\log n)$-approximation to the problem of finding the optimal 1-layered solution. We then describe how this suffices to obtain an $\tilde{O}(\sqrt{n})$-approximation overall.

As we restrict our solution to trees τ with $h(\tau) = 1$, we know that the items in $\mathbf{I}^*$ (items reachable directly from s) will only be used to send flow to the light agents, and items in $\mathbf{I}' = \mathbf{I} \setminus \mathbf{I}^*$ will only be used to send flow directly to terminals. Similarly, heavy agents in $\mathbf{H}^*$ (agents directly reachable from s) will send flow to light agents, while heavy agents in $\mathbf{H}_{\mathsf{pvt}} \setminus \mathbf{H}^*$ will send flow directly to terminals. Therefore, if there are any edges from items in $\mathbf{I}'$ to agents in $\mathbf{H}^*$ we can remove them (no edges are possible between agents in $\mathbf{H} \setminus \mathbf{H}^*$ and items in $\mathbf{I}^*$, since each item has only one incoming edge). Similarly, if there are any edges from items in $\mathbf{I}'$ to light agents in $\mathbf{L}$, we can remove them.

We now proceed with the design of an LP relaxation. For each light agent $A \in \mathbf{L}$, we have a variable x_A showing whether or not A is satisfied by light items. We have a flow of type A, denoted by f_A, from the source s to the agent A, and we require that A receives at least $N_A \cdot x_A$ flow units of this type. Furthermore, for any item in $\mathbf{I}^*$, at most x_A flow units of this type should go through it. Finally, the total flow through any item is bounded by 1. Let $\mathcal{P}(A)$ be the set of paths originating in s and ending at A, that only use items in $\mathbf{I}^*$. We then have the following constraints:

$$\forall A \in \mathbf{L} \quad \sum_{p \in \mathcal{P}(A)} f_A(p) = N_A \cdot x_A \qquad (N_A \cdot x_A \text{ flow units sent to } A) \tag{4}$$

$$\forall A \in \mathbf{L}, \forall i \in \mathbf{I}^* \quad \sum_{\substack{p \in \mathcal{P}(A): \\ i \in p}} f_A(p) \leq x_A \qquad (\text{capacity constraint w.r.t. flow of type } A) \tag{5}$$

$$\forall i \in \mathbf{I}^* \quad \sum_{A \in \mathbf{L}} \sum_{p: i \in p} f_A(p) \leq 1 \qquad (\text{general capacity constraint}) \tag{6}$$

Additionally, we need to send one flow unit to each terminal. For $A \in \mathbf{L}$, $t \in \mathbf{T}$, let $\mathcal{P}(A, t)$ be the set of all paths directly connecting A to t. We now have the standard flow constraints:

$$\forall t \in \mathbf{T} \quad \sum_{A \in \mathbf{L}} \sum_{p \in \mathcal{P}(A,t)} f(p) = 1 \qquad (\text{Each terminal receives 1 flow unit}) \tag{7}$$

$$\forall A \in \mathbf{L} \quad \sum_{t \in \mathbf{T}} \sum_{p \in \mathcal{P}(A,t)} f(p) = x_A \qquad (\text{Light agent } A \text{ sends } x_A \text{ flow units}) \tag{8}$$

$$\forall i \in \mathbf{I}' \quad \sum_{p: i \in p} f(p) \leq 1 \qquad (\text{Capacity constraints}) \tag{9}$$

It is easy to see if there is an 1-layered solution which 1-satisfies all agents, the above LP is feasible. We now describe a rounding procedure to give a $O(\log n)$ approximation for finding the optimal 1-layered solution.

The rounding algorithm consists of three steps. In the first step we perform the BS tree decomposition on the part of the graph induced by $\mathbf{L} \cup (\mathbf{H} \setminus \mathbf{H}^*)$ and $\mathbf{I}'$. The second step is randomized rounding which will create logarithmic congestion. In the last step we take care of the congestion to get an approximate feasible solution.

Step 1: Tree decomposition We consider the graph induced by vertices corresponding to all agents in set $\mathbf{Z} = \mathbf{L} \cup (\mathbf{H} \setminus \mathbf{H}^*)$ and the set $\mathbf{I}'$ of items. Notice that agents in $\mathbf{Z}$ have utility M (or

0) for items in $\mathbf{I}'$ (since we have removed all edges from $\mathbf{I}'$ to $\mathbf{L}$). For each light agent $A \in \mathbf{L}$ we have a value x_A (extent to which A is satisfied as light item). Our fractional flow solution can be interpreted as a fractional assignment of items in $\mathbf{I}'$ to agents in $\mathbf{Z}$, as in Section 3.2, such that each agent $A \in \mathbf{Z}$ is fractionally assigned $(1 - x_A)$-fraction of items in $\mathbf{I}'$, as follows. Let $A \in \mathbf{H} \setminus \mathbf{H}^*$ be any heavy agent. For each item $i \in \mathbf{I}'$, we set $y_{A,i}$ to be the amount of flow sent on edge $(i \to A)$ if such an edge exists. If A is a heavy non-terminal agent, let z be the amount of flow sent on edge $(A \to \pi(A))$. We set $y_{A,\pi(A)} = 1 - z$. For a light agent A, we set $y_{A,\pi(A)} = 1 - x_A$. It is easy to see that this assignment satisfies Constraints (1)–(3). Therefore, we can apply Theorem 5 and obtain a collection $T_1, \ldots, T_s$ of trees together with a matching $\mathcal{M}$ of a subset of items $\mathbf{I}'' \subseteq \mathbf{I}'$ to a subset $\mathbf{Z}' \subseteq \mathbf{Z}$ of agents, that do not participate in trees $T_1, \ldots, T_s$. Since the sum of values x_A for agents in any tree is at least a $1/2$, each tree has at least one light agent.

Step 2: Randomized rounding Consider some tree T_j computed above. We select one of its light agents A with probability x_A/X, where X is the summation of values $x_{A'}$ for $A' \in T_j$. Notice that $x_A/X \leq 2x_A$, since $X \geq \frac{1}{2}$. The selected light agent will eventually be satisfied by light items. Once we select one light agent A_j for each tree T_j, we can satisfy the remaining agents of the tree with items of the tree. Let $\mathbf{L}' = \{A_1, \ldots, A_s\}$. We have thus obtained an assignment of an item from $\mathbf{I}'$ to every agent in $\mathbf{Z} \setminus \mathbf{L}'$. This assignment in turn defines a collection of simple paths $\mathcal{P}_1$, where every path $p \in \mathcal{P}_1$ connects an agent in $A_1, \ldots, A_s$ to a terminal, with at most one path leaving each agent A_j and exactly one path entering each terminal, as follows: Consider any terminal T. Suppose it is assigned item i. Since item i is not the private item of T, there must be a agent A such that $\pi(A) = i$. This agent either belongs to $\mathbf{L}'$ and then we are done, or it is a heavy agent. In the latter case we continue to the item that is assigned to it and so on. In the end, we obtain a path connecting a light agent in $\mathbf{L}'$ to the terminal T. It is clear that each light agent has at most one path originating at it since it has a unique private item, and all paths that we obtain are completely disjoint. In fact, the paths of $\mathcal{P}_1$ of Theorem 4 will be obtained in a similar fashion.

We now turn to finding a collection of elementary paths to satisfy the agents in $\mathbf{L}'$. For each $A \in \mathbf{L}'$ we multiply its flow f_A by the factor of $1/x_A$, so that A now receives N_A flow units. For agents not in $\mathbf{L}'$, we reduce their flows to 0. Notice that due to constraint (5), at most 1 unit of flow f_A goes through any item. Since each agent A is selected with probability at most $2x_A$, using the standard Chernoff bound, we get that w.h.p. the congestion (total flow) on any vertex is $O(\log n)$[1].

Step 3: Final solution In the final solution, we require that each agent $A \in \mathbf{L}'$ receives $\lfloor N_A / \log n \rfloor$ flow units *integrally* via internally disjoint paths from s. Recall that we have an integral flow with congestion $O(\log n)$. So by scaling this flow by a factor of $1/O(\log n)$, we obtain a feasible fractional $1/O(\log n)$-relaxed flow. By the integrality of flow we can get such an integral flow; equivalently a collection of internally-disjoint paths, $\mathcal{P}_2$, with $N_A/O(\log n)$ paths terminating at each agent $A \in \mathbf{L}'$. Combining $\mathcal{P}_1$ and $\mathcal{P}_2$ (note that the union is a set of pairwise internally-disjoint set of paths) we get a $O(\log n)$ approximation to 1-layered instances.

We now show that this algorithm is enough to get an $\tilde{O}(\sqrt{n})$ approximation for Max-Min Allo-

[1]Let $f_A(i)$ and $F_A(i)$ be the total flow of type A through item i in the LP and the final solution respectively. We have $\mathrm{E}[F_A(i)] \leq 2x_A f_A(i)/x_A = 2f_A(i)$. Note that for $A \in \{A_1, \ldots, A_s\}$, $F_A(i)$'s are independent random variables. By constraint (5), $F_A(i) \leq 1$. By constraint (6), $\sum_A f_A(i) \leq 1$ implying total expected flow through i is at most 2. Therefore, by a standard application of the Chernoff bound, we get that for each item i, $\Pr[\sum_A F_A(i) \geq O(\log n)] \leq 1/n^2$ and thus by a union bound, w.h.p. every item has total flow of $O(\log n)$ through it.

CATION. To obtain such an approximation, it is enough to consider instances where $M \geq 4\sqrt{n}$. It is easy to see by using a straightforward modification of Lemma 3 that in this case, by losing a constant factor, we can assume that the optimal solution consists of trees τ with $h(\tau) = 1$. Therefore the algorithm presented here will provide an $\tilde{O}(\sqrt{n})$-approximation for MAX-MIN ALLOCATION.

3.4 Almost Feasible Solutions for Multi-Layered Instances

In this section we prove Theorem 4, by generalizing the algorithm from the previous section to $h = 9/\epsilon$ layers (recall that $1/\epsilon = O(\log n / \log\log n)$). Recall that from Lemma 3, we can assume that the optimal solution to our instance $\mathcal{I}$ induces a collection of h-layered trees. However, we do not know what layer each agent and item appears at in the optimal solution. Therefore, we define a new flow network $N_h(\mathcal{I}, \pi)$, that can be viewed as a "structured" or "layered" version of $N(\mathcal{I}, \pi)$. We simulate trees of height h', for $1 \leq h' \leq h$, by a graph $G_{h'}$, that contains h' levels. Each level j, roughly speaking, contains a copy of the original graph $N(\mathcal{I}, \pi)$, and it is used to simulate the flow-paths that belong to level j in the optimal solution (we say that a vertex belongs to level j in the optimal solution iff there are j light agents on the path connecting it to the source s). The graphs $G_{h'}$, for $1 \leq h' \leq h$ are completely disjoint, except that they share the source s. Additionally, our final graph $N_h(\mathcal{I}, \pi)$ contains another sub-graph $\hat{G}$, that is used to route flow from the last level of each graph $G_{h'}$ to the terminals.

We solve the LP and perform randomized rounding on the new graph $N_h(\mathcal{I}, \pi)$. Since this graph contains a large number of copies of each item and agent from the original instance, we will need in the end to transform our solution into an almost-feasible flow in the original graph $N(\mathcal{I}, \pi)$. We now proceed to formally define the graph $N_h(\mathcal{I}, \pi)$.

3.4.1 The Graph $N_h(\mathcal{I}, \pi)$

Recall that any integral solution induces a collection of disjoint trees τ with various heights $h(\tau) \leq h$. To simplify the description of the LP (and at the cost of losing another factor h in the approximation ratio), our graph will consist of h subgraphs, where subgraph $G_{h'}$, for $1 \leq h' \leq h$, is an h'-layered graph that will correspond to trees of height $h(\tau) = h'$ in the optimal solution. Graphs $G_{h'}$ are completely disjoint except that they share the source vertex s. Fix some $h' : 1 \leq h' \leq h$. Graph $G_{h'}$ is partitioned into h' *levels*. It contains h' copies $\mathbf{L}_1, \ldots, \mathbf{L}_{h'}$ of the set $\mathbf{L}$ of the light agents. Copy $\mathbf{L}_j$ belongs to level j of $G_{h'}$, for $1 \leq j \leq h'$.

Level 1 additionally contains the source s, a copy of each item in $\mathbf{I}^*$ and a copy of each heavy agent in $\mathbf{H}^*$ (recall that these are the items and the agents that can be reached directly from s). There is an edge from s to every item in $\mathbf{S}$, an edge from every heavy agent A to its private item $\pi(A)$, and an edge from each item $i \in \Gamma_{\mathrm{H}}(A) \setminus \pi(A)$ to A. Additionally, if item i is a light item for a light agent A, we put an edge between i and a copy of A in $\mathbf{L}_1$.

Level j is defined as follows. Apart from $\mathbf{L}_j$, it contains a copy of each heavy non-terminal agent $A \in \mathbf{H} \setminus \mathbf{T}$ and a copy of each item $i \in \mathbf{I} \setminus \mathbf{S}$. Let $\mathbf{H}_j$ and $\mathbf{I}_j$ denote the set of copies of the heavy agents and items at level j. Recall that each item $i \in \mathbf{I} \setminus \mathbf{S}$ is a private item of some agent. Consider some such item i. If it is a private item of a heavy agent A, then we add an edge from a copy of A in $\mathbf{H}_j$ to a copy of i in $\mathbf{I}_j$. If it is a private item of light agent A, then we add an edge from a copy of A in $\mathbf{L}_{j-1}$ to a copy of i in $\mathbf{I}_j$. If i is a non-private item admissible for heavy agent A, then we add

an edge from a copy of i in $\mathbf{I}_j$ to a copy of A in $\mathbf{H}_j$. If i is a light item for a light agent A then we add an edge from a copy of i in $\mathbf{I}_j$ to a copy of A in $\mathbf{L}_j$. This completes the description of $G_{h'}$. We will denote by $\mathbf{H}_j^{h'}, \mathbf{I}_j^{h'}$ the copies of the heavy agents and items at level j of $G_{h'}$, and we will omit the superscript h' when clear from context. Our final graph consists of the union of graph $G_{h'}$, for $1 \leq h' \leq h$. Finally, we add an additional sub-graph $\hat{G}$ to $N_h(\mathcal{I}, \pi)$, that will be used to route flow directly to the terminals.

Graph $\hat{G}$ consists of a set $\hat{\mathbf{H}}$ of vertices, containing a vertex for every heavy agent and a set $\hat{\mathbf{I}}$ containing a vertex for every item $i \in \mathbf{I} \setminus \mathbf{S}$. Note that every item in $\hat{\mathbf{I}}$ is a private item of some agent. If $i \in \hat{\mathbf{I}}$ is a private item of a heavy agent A then we add an edge from a copy of A in $\hat{\mathbf{H}}$ to a copy of i in $\hat{\mathbf{I}}$. If it is a private item of a light agent A, then for all $h' : 1 \leq h' \leq h$, we add an edge from the copy of A in $\mathbf{L}_{h'}^{h'}$ to the copy of i in $\hat{\mathbf{I}}$ (so there are only edges from the last layer of each $G_{h'}$ to items in $\hat{\mathbf{I}}$). If i is an admissible but non-private item for some heavy agent A then we add an edge from the copy of i in $\hat{\mathbf{I}}$ to the copy of A in $\hat{\mathbf{H}}$.

This completes the description of the graph $N_h(\mathcal{I}, \pi)$. Notice that for each item $i \in \mathbf{I}$ and each agent $A \in \mathbf{A}$ of $\mathcal{I}$, there are at most h^2 copies in $N_h(\mathcal{I}, \pi)$. Also notice that there is a single copy of the terminals. We will be looking for an integral flow in this graph satisfying conditions ($\hat{\text{C}}1$)–($\hat{\text{C}}3$). Notice that given an optimal h-layered solution to the original instance, it is easy to convert it into a feasible integral flow in the new instance.

3.4.2 The Linear Programming Relaxation

We start with a high-level overview and intuition. The LP is a natural generalization of the 1-level LP from the previous section, but the size of the LP now grows to $n^{O(1/\epsilon)}$. Consider some sub-graph $G_{h'}$, for $1 \leq h' \leq h$. For each light agent A in the last layer $\mathbf{L}_{h'}^{h'}$ of $G_{h'}$, we have a variable x_A showing the extent to which A is satisfied as a light agent (and equivalently, how much flow it sends to the terminals). Let $\mathcal{L} = \bigcup_{h'} \mathbf{L}_{h'}^{h'}$ denote the set of all light agents in the last layers of graphs $G_{h'}$. We then write standard flow constraints on the part of the graph induced by $(\mathcal{L} \cup \hat{\mathbf{H}} \cup \hat{\mathbf{I}})$ requiring that each terminal receives one flow unit, and each agent $A \in \mathcal{L}$ sends x_A flow units. We also have capacity constraints requiring that at most one flow unit traverses any vertex.

It is instructive to first try to generalize the LP-rounding algorithm from the previous section to the multi-layered setting. We can again perform the BS decomposition on the LP-solution in graph $\hat{G}$, to obtain the tree decomposition, and then randomly choose one vertex in each such tree that will be satisfied by light items. Therefore, for each graph $G_{h'}$, we obtain a subset $\mathbf{L}'_{h'} \subseteq \mathbf{L}_{h'}^{h'}$ of light agents, that need to be satisfied by light items. From this point onwards we focus on each subgraph $G_{h'}$ separately. We perform an iterative randomized rounding procedure to create a feasible integral solution that originates at the source s and satisfies these agents. In the jth iteration we will select light agents from layer $(h' - j)$ which need to be satisfied by light items. Given these agents we will choose the light agents in the successive layer, again via randomized rounding. So after j iterations, we will have an integral flow in levels $h' - j, \ldots, h'$ and fractional flow in the remaining levels.

In the first iteration, for each selected agent $A \in \mathbf{L}'_{h'}$, we scale all its *incoming* flow from the source s by a factor of $1/x_A$. In order to ensure that congestion does not grow, our LP needs to ensure that for every item i in $G_{h'}$ (no matter what level it is in), the amount of flow traversing item i that eventually reaches the agent A is at most x_A. Once A is chosen, we need to choose N_A light agents in $\mathbf{L}_{h'-1}$ that will send their flow to A. This again is done by randomized rounding. For each A' that

we choose, we will scale all the relevant flow by $1/x_{A'}$, and we again need to ensure that no vertex in the graph sends more than $x_{A'}$ flow to A'. The need to ensure this type of capacity constraints over all levels makes the LP somewhat complex. In particular, for the flow-paths in level j, $1 \leq j \leq h'$, we need to keep track of the $h' - j$ light agents that this flow-path traverses in sets $\mathbf{L}_{h'-j}, \ldots, \mathbf{L}_{h'}$. In fact for each such j-tuple of light vertices, we need to introduce an LP-variable, which leads to the large size of the LP. We now describe the LP in detail.

The LP consists of three parts. The first and the easiest part deals with routing the flow directly to the terminals. After this part, we can focus on each subgraph $G_{h'}$ separately. In the second part, we will coordinate the flow that light agents at one level of $G_{h'}$ send to light agents in the next level, ignoring the actual routing of the flow inside each level. In the third part we will perform the routing inside each level, imposing the capacity constraints on the vertices.

Part 1: Flow arriving directly to terminals Let $\mathcal{L}$ be the set of light agents appearing in the last layer of each graph $G_{h'}$ (so $\mathcal{L} = \bigcup_{h'} \mathbf{L}_{h'}^{h'}$). For each light agent $A \in \mathcal{L}$ we have a variable x_A signifying whether or not A is satisfied by light items, or equivalently whether or not it sends flow directly to terminals. For each terminal $t \in T$, for each light agent $A \in \mathcal{L}$, let $\mathcal{P}(A, t)$ be the set of all paths connecting A to t (notice that each such path only uses items in $\hat{\mathbf{I}}$ and heavy agents in $\hat{\mathbf{H}}$ and there are no additional light agents on the path). We have the following constraints:

$$\forall t \in \mathbf{T} \quad \sum_{A \in \mathcal{L}} \sum_{p \in \mathcal{P}(A,t)} f(p) = 1 \quad \text{(Each terminal receives one flow unit)} \tag{10}$$

$$\forall A \in \mathcal{L} \quad \sum_{t \in \mathbf{T}} \sum_{p \in \mathcal{P}(A,t)} f(p) = x_A \quad \text{(Light agent } A \text{ sends } x_A \text{ flow units)} \tag{11}$$

$$\forall i \in \hat{\mathbf{I}} \quad \sum_{p : i \in p} f(p) \leq 1 \quad \text{(Capacity constraints)} \tag{12}$$

Notice that each heavy agent $A \in \hat{\mathbf{H}} \setminus \mathbf{T}$ has only one outgoing edge connecting it to its private item, and so it is enough to enforce the capacity constraints on the items.

This is the only part that is common to the whole graph. From now on we fix a subgraph $G_{h'}$ and describe the constraints relevant to it. For simplicity we will omit the superscript h'.

Part 2: Routing among the light agents inside $G_{h'}$ This part will specify the *amount* of flow to be routed between different light agents, while we ignore the routing itself, which will be taken care of in the next part. For clarity of exposition, we will add a layer 0 to our graph, where $\mathbf{L}_0 = \{s\}$.

For each $j : 0 \leq j \leq h'$, we let $\mathcal{S}_j = \mathbf{L}_{h'} \times \mathbf{L}_{h'-1} \times \cdots \times \mathbf{L}_j$. For each tuple $\lambda = (\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j) \in \mathcal{S}_j$ of light agents, where $\ell_k \in \mathbf{L}_k$ for $j \leq k \leq h'$, we have a variable $y(\lambda)$. The meaning of this variable in the integral solution is whether or not ℓ_j sends flow to $\ell_{h'}$ via a path whose only light agents are $\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j$. Notice that $\mathcal{S}_{h'} = \mathbf{L}_{h'}$, and so we have the constraint:

$$\forall A \in \mathbf{L}_{h'} \quad y(A) = x_A \quad \text{(Total flow of } x_A \text{ for each light agent } A \in \mathbf{L}_{h'}) \tag{13}$$

Consider now some tuple $\lambda = (\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j) \in \mathcal{S}_j$. If $y(\lambda) = 1$, and flow is being sent from ℓ_j to $\ell_{h'}$ via paths corresponding to the tuple λ, then ℓ_j has to receive N_{ℓ_j} flow units from layer $(j-1)$. For $\lambda \in \mathcal{S}_j$ and $A \in \mathbf{L}_{j-1}$, let $\lambda \circ A \in \mathcal{S}_{j-1}$ be the tuple obtained by concatenating A at the end of λ. So we have the constraint:

$$\forall 1 \leq j \leq h', \forall \lambda = (\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j) \in \mathcal{S}_j \quad \sum_{A \in \mathbf{L}_{j-1}} y(\lambda \circ A) = N_{\ell_j} \cdot y(\lambda) \tag{14}$$

Additionally, for $j \geq 2$, each $A \in \mathbf{L}_{j-1}$ is only allowed to send at most $y(\lambda)$ flow units via the tuple λ (this is similar to the capacity constraint (5) from the previous LP):

$$\begin{array}{l} \forall 2 \leq j \leq h', \forall A \in \mathbf{L}_{j-1} \\ \forall \lambda = (\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j) \in \mathcal{S}_j \end{array}, \quad y(\lambda \circ A) \leq y(\lambda) \quad (\lambda\text{-flow capacity constraints for one level}) \tag{15}$$

Finally, we need to add the more complex capacity constraints that will ensure that the iterative randomized rounding procedure will go through without incurring large congestion. Consider some tuple $\lambda = (\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j) \in \mathcal{S}_j$, and let $A \in \mathbf{L}_k$ be any light agent in some layer k, where $k < j$. Then the total amount of flow that A can send via all tuples whose *prefix* is λ is at most $y(\lambda)$. Given $\lambda \in \mathcal{S}_j$ and $A \in \mathbf{L}_k$ with $k < j$, let $Z(\lambda, A) \subseteq \mathcal{S}_k$ be the set of all tuples whose prefix is λ and whose last agent is A. Then we have the following capacity constraints:

$$\begin{array}{l} \forall 1 \leq k < j \leq h' \\ \forall A \in \mathbf{L}_k, \forall \lambda \in \mathcal{S}_j \end{array}, \quad \sum_{\lambda' \in Z(\lambda, A)} y(\lambda') \leq y(\lambda) \quad (\lambda\text{-flow capacity constraints for multiple levels}) \tag{16}$$

Actually the set (16) of constraints contains the constraints in (15) as a special case, and we only added (15) for motivating these more general constraints. Finally to complete this part we require that each light agent sends at most one flow unit in total:

$$\forall 1 \leq j \leq h, \forall A \in \mathbf{L}_j \quad \sum_{\substack{\lambda \in \mathcal{S}_j: \\ A \in \lambda}} y(\lambda) \leq 1 \quad (\text{General capacity constraints for light agents}) \tag{17}$$

Part 3: Routing the flow We focus on level j of graph $G_{h'}$. Consider some tuple $\lambda = (\ell_{h'}, \ldots, \ell_j, \ell_{j-1}) \in \mathcal{S}_{j-1}$. We will have flow f_λ of type λ, and we need to send $y(\lambda)$ flow units of this type from ℓ_{j-1} to ℓ_j. For any pair $\ell_{j-1} \in \mathbf{L}_{j-1}, \ell_j \in \mathbf{L}_j$ of agents, let $\mathcal{P}(\ell_{j-1}, \ell_j)$ be the set of all paths connecting them (note that these paths are completely contained inside level j). Then:

$$\forall 1 \leq j \leq h', \forall \lambda = (\ell_{h'}, \ldots, \ell_j, \ell_{j-1}) \in \mathcal{S}_{j-1}, \quad \sum_{p \in \mathcal{P}(\ell_{j-1}, \ell_j)} f_\lambda(p) = y(\lambda) \quad (\text{routing flow of each type}) \tag{18}$$

We need to add the simple capacity constraints that the flow via any item is at most 1:

$$\forall 1 \leq j \leq h', \forall i \in \mathbf{I}_j \quad \sum_{p: i \in p} \sum_{\lambda \in \mathcal{S}_{j-1}} f_\lambda(p) \leq 1 \quad (\text{General capacity constraints}) \tag{19}$$

Note that since each non-terminal heavy agent has exactly one out-going edge (that connects it to its private item), the constraint above also implicitly bounds the flow through a non-terminal heavy agent to be 1.

And finally, we need to add capacity constraints, which are very similar to (16). For a tuple $\lambda = (\ell_{h'}, \ldots, \ell_j) \in \mathcal{S}_j$, for each $j \leq q \leq h'$, we denote $\lambda_q = \ell_q$. Consider some tuple $\lambda = (\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j) \in \mathcal{S}_j$, and let i be any item in any layer $\mathbf{I}_k$, where $k \leq j$. Then the total flow of type λ through i via all tuples whose prefix is λ is at most $y(\lambda)$. Given $\lambda \in \mathcal{S}_j$ and layer $\mathbf{L}_k$ with $k \leq j$, let $Z'(\lambda, k) \subseteq \mathcal{S}_k$ be the set of all tuples whose prefix is λ. Then we have the following capacity constraints:

$$\begin{array}{l} \forall 1 \leq k \leq j \leq h' \\ \forall \lambda \in \mathcal{S}_j, \forall i \in \mathbf{I}_k \end{array} , \quad \sum_{\lambda' \in Z'(\lambda, k-1)} \sum_{\substack{p \in \mathcal{P}(\lambda'_{k-1}, \lambda'_k): \\ i \in p}} f_{\lambda'}(p) \leq y(\lambda) \quad \text{(multi-leveled capacity constraints)} \tag{20}$$

Solving the LP: The total number of different flow types in the LP is $O(|\mathcal{S}_1|) = O(m^h) = n^{O(1/\epsilon)}$. As written, the LP has exponential number of variables representing the flow-paths. For computing a solution, we can replace it with the standard compact formulation for multi-commodity flows that specifies flow-conservation constraints. We can then use the standard decomposition into flow-paths to obtain a feasible solution for our LP. Therefore the overall complexity of computing the LP solution is $n^{O(1/\epsilon)}$.

3.4.3 The Rounding Algorithm

The algorithm has three parts. The first part uses the BS decomposition to take care of the direct routing to the terminals. The output of the first part is the set $\mathcal{P}_1$ of vertex-disjoint elementary paths connecting light agents to terminals in the original graph $N(\mathcal{I}, \pi)$. The second part is randomized rounding in each sub-graph. The third part is the "clean-up" phase where we get rid of almost all the congestion and create the set $\mathcal{P}_2$ of paths in $N(\mathcal{I}, \pi)$ that are used to satisfy the light agents.

Part 1: Routing to the Terminals We consider the sub-graph of $N_h(\mathcal{I}, \pi)$ induced by the set $\mathbf{Z} = \hat{\mathbf{H}} \cup \mathcal{L}$ of agents (where $\mathcal{L}$ contains the light agents in the last layer of every subgraph $G_{h'}$, $\mathcal{L} = \bigcup_{h' \leq h} \mathbf{L}^{h'}_{h'}$) and the set $\hat{\mathbf{I}}$ of items. Recall that the items in $\hat{\mathbf{I}}$ give utility M or 0 to all agents in $\mathbf{Z}$. For each agent $A \in \mathcal{L}$ we have the value x_A defined by our LP-solution, while for each agent $A \in \hat{\mathbf{H}}$ we set $x_A = 0$. Exactly like in the first part of the rounding algorithm for Section 3.3, we can produce values $y_{A,i}$ for each $A \in \mathbf{Z}$, $i \in \hat{\mathbf{I}}$ satisfying the constraints (1)–(3). We then again apply Theorem 5 to obtain a decomposition of the bipartite graph $G(\mathbf{Z}, \hat{\mathbf{I}})$ into trees $T_1, \ldots, T_s$. Let T_j, $1 \leq j \leq s$ be any tree in the decomposition containing more than one edge. Recall that the summation of values x_A for agents A in tree T_j is at least $\frac{1}{2}$ and thus it contains at least one light agent. We select one of the agents A of T_j with probability x_A/X, where X is the summation of values $x_{A'}$ for all agents A' in T_j. Notice that this probability is at most $2x_A$, since $X \geq \frac{1}{2}$. Once A is selected, we can assign the items of tree T_j to the remaining agents. Let $\mathcal{L}' \subseteq \mathcal{L}$ be the set of light agents we have thus selected. Let $\mathcal{L}' := \bigcup_{h' \leq h} \mathbf{L}'_h$ be the partition of $\mathcal{L}'$ across the h subgraphs. We obtain an assignment of items in $\hat{\mathbf{I}}$ to agents in $\mathbf{Z} \setminus \mathcal{L}'$, where each agent is assigned one item. This

assignment of items defines an integral flow in the sub-graph of $N_h(\mathcal{I},\pi)$ induced by $\mathbf{Z}\cup\hat{\mathbf{I}}$, as follows. Every light agent in $\mathcal{L}'$ sends one flow unit to its private item. If agent $A\in\mathbf{Z}\setminus\mathcal{L}'$ is assigned item $i\neq\pi(A)$, then i sends one flow unit to A and if $A\notin\mathbf{T}$, it sends one flow unit to $\pi(A)$. This flow is a collection of disjoint elementary paths connecting items in $\mathcal{L}'$ to the terminals. Let $\mathcal{P}_1$ denote the corresponding collection of paths in $N(\mathcal{I},\pi)$, (where we replace copies of agents and items back by the corresponding agents and items themselves). The set $\mathcal{P}_1$ of paths is completely vertex disjoint: it is clear that paths in $\mathcal{P}_1$ cannot share heavy agents or items. It is also impossible that a light agent A has more than one path in $\mathcal{P}_1$ starting at A: even though many copies of A, appearing in the last layers $\mathbf{L}^{h'}_{h'}$ for each graph $G_{h'}$ connect to $\hat{\mathbf{I}}$, all these copies only connect to a single copy of $h(A)$ in $\hat{\mathbf{I}}$ and so all paths starting at copies of A have to go through this vertex.

The set $\mathcal{P}_1$ of paths will not change for the rest of the algorithm and will be part of the output. We now focus on finding the set $\mathcal{P}_2$ of paths that supply flow to the light agents in $\mathcal{L}'$.

Part 2: Randomized Rounding We focus on one subgraph $G_{h'}$, where we have a subset $\mathbf{L}'_{h'}\subseteq\mathbf{L}_{h'}$ of light agents that have been selected in Part 1.

Our randomized rounding procedure has $h'+1$ iterations. Intuitively, iteration j produces an LP-solution that is "integral" for levels $(h'-j+1,\ldots h')$, and is fractional for the remaining levels. Formally, the input to iteration $(h'-j'+1)$, for $h'\geq j'\geq 0$, is an LP-solution that has the following properties:

P1. For each $j: j'\leq j\leq h'$, for each $\lambda\in\mathcal{S}_j$, we have an *integral* value $y(\lambda)\in\{0,1\}$, and for each flow-path p, the value $f_\lambda(p)\in\{0,1\}$ is also integral. The remaining variables (representing flow inside levels $1,\ldots,j'$) may be fractional.

P2. Constraints (14)–(16) and constraint (20) hold for all $j\leq j'$, while constraint (18) holds for all $j: 1\leq j\leq h'$.

P3. For for $\lambda\in S_j$, where $j'<j\leq h'$, constraint (14) is satisfied approximately, that is, we replace it with the following constraint (notice that $y(\lambda), y(\lambda\circ A)\in\{0,1\}$):

$$\forall j'<j\leq h', \forall\lambda=(\ell_{h'},\ell_{h'-1},\ldots,\ell_j)\in\mathcal{S}_j \quad \sum_{A\in\mathbf{L}_{j-1}} y(\lambda\circ A)\geq N_{\ell_j}\cdot y(\lambda)/2 \tag{21}$$

P4. Constraints (17) and (19) (general capacity constraints for light agents and items) are satisfied approximately, with congestion at most $(16h^2\cdot\log n)(1+\frac{1}{h})^{h'-j'}$ for each item and each light agent, respectively. That is, we replace constraints (17) and (19) with the following constraints:

$$\forall 1\leq j\leq h, \forall A\in\mathbf{L}_j \quad \sum_{\substack{\lambda\in\mathcal{S}_j:\\ A\in\lambda}} y(\lambda)\leq(16h^2\cdot\log n)(1+\frac{1}{h})^{h'-j'} \quad \text{(General capacity constraints for light agents)} \tag{22}$$

$$\forall 1\leq j\leq h', \forall i\in\mathbf{I}_j \quad \sum_{p:i\in p}\sum_{\lambda\in\mathcal{S}_{j-1}} f_\lambda(p)\leq(16h^2\cdot\log n)(1+\frac{1}{h})^{h'-j'} \quad \text{(General capacity constraints)} \tag{23}$$

The input to the first iteration (corresponding to $j' = h'$) is produced as follows. Recall that $\mathcal{S}_{h'} = \mathbf{L}_{h'}$, and so for each $A \in \mathbf{L}_{h'}$, $y(A) = x_A$. For each $A \in \mathbf{L}'_{h'}$, set $y(A) = 1$ and for each $A \notin \mathbf{L}'_h$, set $y(A) = 0$. For each $j : 1 \leq j \leq h'$, for each $\lambda = (\ell_{h'}, \ldots, \ell_j) \in \mathcal{S}_j$, we proceed as follows. If $A = \ell_{h'}$ belongs to $\mathbf{L}'_{h'}$, then we multiply $y(\lambda)$ by factor $1/x(A)$, and for each path p, we multiply the flow $f_\lambda(p)$ by the same factor. Otherwise, if $\ell_{h'} \notin \mathbf{L}'_{h'}$, then we set $y(\lambda) = 0$, and for all paths p, $f_\lambda(p)$ is also set to 0. Note that property P1 holds. Furthermore, all constraints (14)–(16) and constraints (18),(20) hold for all j, since we multiply both sides of each such constraint by the same value, so we have properties P2 and P3. We will prove below that P4 holds as well (Lemma 5).

We now consider iteration $(h' - j')$, and assume that properties P1–P4 hold for the input to this iteration. Let $\mathcal{S}'_{j'} \subseteq \mathcal{S}_{j'}$ be the subset of tuples λ with $y(\lambda) = 1$, and let $\lambda = (\ell_{h'}, \ldots, \ell_{j'})$ be some such tuple. Let $Z(\lambda, j' - 1)$ denote the set of of tuples in $\mathcal{S}_{j'-1}$ whose predecessor is λ, that is, $Z(\lambda, j' - 1) := \{\lambda' = \lambda \circ A : A \in \mathbf{L}_{j'-1}\}$. Then due to constraint (14), $\sum_{\lambda' \in Z(\lambda, j'-1)} y(\lambda') = N_{\ell_{j'}}$. We randomly select each tuple $\lambda' \in Z(\lambda, j' - 1)$ with probability $y(\lambda')$. We do this for every $\lambda \in \mathcal{S}_{j'}$ with $y(\lambda) = 1$.

If λ' is selected, then we proceed as follows:

- Set $y(\lambda') = 1$.
- Randomly sample a flow-path p carrying positive flow of type λ' with probability $f_{\lambda'}(p)/y_{\lambda'}$. Set $f_{\lambda'}(p) = 1$ if p is selected, and set it to be 0 if p is not selected.
- For all $j < j' - 1$, for all tuples $\lambda'' \in \mathcal{S}_j$, such that λ' is a prefix of λ'', multiply the value $y(\lambda'')$ by factor $1/y(\lambda')$, and for each path p', multiply the value $f_{\lambda''}(p)$ by the same factor.

If λ' is not selected, then for all $j < j'$, for all tuples $\lambda'' \in \mathcal{S}_j$, such that λ' is a prefix of λ'', we set $y(\lambda'') = 0$ and $f_{\lambda''}(p') = 0$ for all paths p'.

This completes the description of an iteration. We show that w.h.p. properties P1–P4 remain true after the current iteration.

First observe that for all $\lambda' \in \mathcal{S}_{j'-1}$, the values $y(\lambda)$ become integral, and the same is true for $f_{\lambda'}(p)$ for all paths p. Since $y(\lambda)$ and $f_\lambda(p)$ haven't been modified for $\lambda \in \mathcal{S}_j;\ j \geq j'$, we get property P1.

We now consider the properties one-by-one. First, fix some $j \leq j' - 1$, and consider constraints (14)–(16) and constraint (20). For each such constraint, we have scaled both its sides by the same value, so these constraints continue to hold. Consider now constraint (18). For $j < j'$, we have again scaled both sides of such constraints by the same factor, and so they continue to hold. For $j > j'$, the constraints continue to hold because they held at the beginning of the current iteration. Finally, for $j = j'$, for each $\lambda' \in \mathcal{S}_{j'-1}$, if $y(\lambda') = 1$, then we have selected exactly one of the corresponding paths p and set $f_{\lambda'}(p) = 1$, setting $f_{\lambda'}(p') = 0$ for all $p \neq p'$. If $y(\lambda') = 0$, then we have set $f_{\lambda'}(p) = 0$ for all p. Therefore, constraint (18) holds for all $j : 1 \leq j \leq h'$. This establishes property P2. Finally, in the next two lemmas we establish the remaining properties.

Lemma 4 *If properties P1–P4 hold at the beginning of iteration $(h' - j')$, then property P3 holds at the beginning of iteration $(h' - j' + 1)$ w.h.p.*

Proof: Recall that constraint (14) holds for $j = j' - 1$. Let $\lambda = (\ell_{h'}, \ldots, \ell_{j'}) \in \mathcal{S}'_{j'}$, and denote $A = \ell_{j'}$. As before, let $Z(\lambda, j' - 1) \subseteq \mathcal{S}_{j'-1}$ be the set of all tuples $\{\lambda' = \lambda \circ A : A \in \mathbf{L}_{j'-1}\}$.

Then due to constraint (14), $\sum_{\lambda' \in Z(\lambda, j'-1)} y(\lambda') = N_{l_{j'}}$. Each such tuple $\lambda' = \lambda \circ A$ is selected with probability $y(\lambda')$. So the expected number of tuples in $Z(\lambda, j'-1)$ that are selected is exactly $N_{l_{j'}}$. It is enough to prove that at least $N_{l_{j'}}/2$ tuples are selected w.h.p. This easily follows from Chernoff bound: If $Y_{\lambda'}$ is the indicator for λ' being selected, $Pr[\sum Y_{\lambda'} < N_{l_{j'}}/2] \leq e^{-N_{l_{j'}}/16} \leq O(1/n^8)$ since $N_{l_{j'}} \geq n^{\epsilon}/h \geq \Omega(\log^8 n)$. □

Lemma 5 *Property P4 holds at the beginning of the first iteration. Moreover, for each $0 \leq j' < h'$, if properties P1–P4 hold at the beginning of iteration $(h' - j')$, then property P4 holds at the beginning of iteration $(h' - j' + 1)$ w.h.p.*

Proof: We start with the input to the first iteration. Let i be any item in any level $i \in \mathbf{I}_k$. Consider any tuple $\lambda \in \mathcal{S}_{h'}$ (so $\lambda \in \mathbf{L}_{h'}$)). As before, let $Z'(\lambda, k-1) \subseteq \mathcal{S}_{k-1}$ be the set of all tuples whose prefix is λ. Let $F_\lambda(i)$ denote the total flow of types $\lambda' \in Z'(\lambda, k-1)$ that goes through i, i.e., $F_\lambda(i) = \sum_{\lambda' \in Z'(\lambda,k-1)} \sum_{\substack{p \in \mathcal{P}(\lambda'_{k-1}, \lambda'_k): \\ i \in p}} f_{\lambda'}(p)$. Constraint (20) ensures that for each $\lambda \in \mathcal{S}_{h'}$, $F_\lambda(i) \leq y(\lambda)$. The total flow through item i is $\sum_{\lambda \in \mathcal{S}_{h'}} F_\lambda(i) \leq 1$ due to constraint (19). Recall that our algorithm in Part 1 selected each agent $A \in \mathbf{L}_{h'}$ with probability at most $2x_A = 2y(A)$. If A is selected, then the flow $f_{\lambda'}(p)$ for all $\lambda' \in Z'(\lambda, k-1)$ were $\lambda = A$ is multiplied by factor $1/y(A)$. Therefore, the expected amount of flow through item i in the input to the first iteration is bounded by $\sum_{\lambda \in \mathcal{S}_{h'-1}} 2F_\lambda(i) \leq 2$. Using the standard Chernoff bound, it is easy to see that w.h.p. the total flow through item i is at most $O(\log n)$.

We now consider iteration $(h' - j')$, and we assume that properties P1–P4 hold for the input to this iteration. Fix some item $i \in \mathbf{I}_k$ for some $k \leq j' - 1$ and consider some $\lambda \in \mathcal{S}_{j'-1}$. As before, let $Z'(\lambda, k-1) \subseteq \mathcal{S}_{k-1}$ be the set of all tuples whose prefix is λ. Again, let $F_\lambda(i)$ denote the total flow of types $\lambda' \in Z'(\lambda, k-1)$ that goes through i, i.e., $F_\lambda(i) = \sum_{\lambda' \in Z'(\lambda,k-1)} \sum_{\substack{p \in \mathcal{P}(\lambda'_{k-1}, \lambda'_k): \\ i \in p}} f_{\lambda'}(p)$. Constraint (20) ensures that for each $\lambda \in \mathcal{S}_{j'-1}$, $F_\lambda(i) \leq y(\lambda)$. The total flow through item i is $\sum_{\lambda \in \mathcal{S}_{j'-1}} F_\lambda(i) \leq (16h^2 \cdot \log n)(1 + \frac{1}{h})^{h'-j'}$ due to property P4.

Recall that our selected each tuple $\lambda \in \mathcal{S}_{j'-1}$ with probability at most $y(\lambda)$, and if λ is selected, then the flow $f_{\lambda'}(p)$ for all $\lambda' \in Z'(\lambda, k-1)$ is multiplied by factor $1/y(\lambda)$. Therefore, the expected amount of flow through item i after iteration $(h'-j')$ is bounded by $\sum_{\lambda \in \mathcal{S}_{j'-1}} F_\lambda(i) \leq (16h^2 \cdot \log n)(1+\frac{1}{h})^{h'-j'}$.

For each $\lambda \in \mathcal{S}_{j'-1}$, let $C(\lambda)$ be the random variable whose value is $F_\lambda(i)/y(\lambda)$ if λ is selected, and it is 0 otherwise. Then $\mu := \mathbf{E}\left[\sum_{\lambda \in \mathcal{S}_{j'-1}} C(\lambda)\right] = \sum_{\lambda \in \mathcal{S}_{j'-1}} F_\lambda(i)$. We need to show that w.h.p. this summation is at most $(16h^2 \cdot \log n)(1 + \frac{1}{h})^{h'-j'+1}$. We again apply the standard Chernoff bound[2]: If $\mu = o(h^2 \log n)$, we get

$$\mathbf{Pr}\left[\sum_{\lambda \in \mathcal{S}_{j'-1}} C(\lambda) \geq 16h^2 \log n\right] \leq \exp(-h^2 \log n) \leq 1/\operatorname{poly}(n)$$

since $h \geq 1$. If $\mu = \Omega(h^2 \log n)$, we get

[2] $\mathbf{Pr}[X > (1+\delta)\mu] \leq 2^{-\delta\mu}$, if $\delta > 2e - 1$ and $\mathbf{Pr}[X > (1+\delta)\mu] \leq e^{-\mu\delta^2/4}$ otherwise.

$$\mathbf{Pr}\left[\sum_{\lambda\in\mathcal{S}_{j'-1}} C(\lambda) \geq \mu(1+1/h)\right] \leq \exp(-\mu/4h^2) \leq 1/\operatorname{poly}(n)$$

since $\mu \geq \Omega(h^2 \log n)$.

So far we have bounded congestion on items. The proof for bounding congestion on light agents is identical, except that now we use constraints (16) and (17).

□

At the end of the above procedure, we obtain a set $\mathcal{P}'$ of elementary paths. Constraint (21) ensures that if $A \in \mathbf{L}_{h'}^{h'}$ or A has a elementary path leaving it in $\mathcal{P}'$, then with high probability it has at least $N_A/2$ paths entering it in $\mathcal{P}'$. From constraint (22), the total number of paths leaving a light agent is bounded by $O(h^2 \cdot \log n)$, and from constraint (23), the total number of paths to which an item can belong is bounded by $O(h^2 \log n)$ with high probability. Since each heavy agent has at most one out-going edge, that connects it to an item, the number of paths in $\mathcal{P}'$ containing a specific heavy agent is also at most $(h^2 \log n)$ w.h.p.

Getting an Almost-Feasible Solution In the last step of the algorithm we produce a set $\mathcal{P}_2$ of elementary paths, such that properties (D1)–(D4) hold for $\mathcal{P}_1$, $\mathcal{P}_2$.

Consider the flow-paths in $\mathcal{P}'$, and let $\mathcal{P}''$ be the corresponding paths in the original graph $N(\mathcal{I}, \pi)$ (where we replace copies of agents and items by their original counterparts). These flow-paths have the following properties: (i) every vertex that does not correspond to a terminal may appear in at most $\alpha' = O(h^4 \log n)$ flow-paths in $\mathcal{P}''$ (the additional factor of h^2 is due to the at most h^2 copies of every vertex in $N_h(\mathcal{I}, \pi)$), and (ii) for any light agent A, there are at most α' paths in $\mathcal{P}''$ starting at A, and if there is a path in $\mathcal{P}'' \cup \mathcal{P}_1$ originating at A, then there must be at least $N_A/2$ paths in $\mathcal{P}''$ terminating at A.

We now show how to convert the set $\mathcal{P}''$ of paths into set $\mathcal{P}_2$, such that properties D1–D4 hold for $\mathcal{P}_1$, $\mathcal{P}_2$. The next lemma will complete the proof of Theorem 4.

Lemma 6 *Given the sets $\mathcal{P}_1, \mathcal{P}''$, we can find in polynomial time a set $\mathcal{P}_2$ of paths, such that properties D1–D4 hold for $\mathcal{P}_1$, $\mathcal{P}_2$.*

Proof: Let $\mathcal{S}$ be the set of agents from which paths in $\mathcal{P}''$ originate, and let $\mathcal{R}$ be the set of agents in which paths in $\mathcal{P}''$ terminate. The agents in $\mathcal{S}$ are called *senders*, and agents in $\mathcal{R}$ are called *receivers*. Note that by construction of $\mathcal{P}''$, each sender must also be a receiver.

We now build the following flow network. The set of vertices consists of vertices representing agents in $\mathcal{S}$ and $\mathcal{R}$ (so if an agent serves both as sender and receiver, it will appear in both $\mathcal{S}$ and $\mathcal{R}$, and there will be two vertices representing it). Additionally, there is a vertex for each non-terminal heavy agent and for each item, and there is a source s and a sink t. Source s connects to every item in $\mathbf{S}$ and to every sender with capacity-1 edges. Every receiver connects to t with $\lfloor N_A/2\alpha' \rfloor$ parallel edges of capacity 1 each. There is an edge from every sender to its private item, and from every heavy agent to its private item. There is an edge from item i to heavy agent A iff $i \in \Gamma_{\mathrm{H}}(A) \setminus \{\pi(A)\}$. There is an edge from item i to receiver A iff i is a light item for A. Each of these edges is of unit capacity. The goal is to route $\sum_{A\in\mathcal{R}} \lfloor N_A/2\alpha' \rfloor$ flow units from s to t.

Observe that the flow-paths in $\mathcal{P}''$ induce flow of value at least $\sum_{A\in\mathcal{R}} N_A/2$, and violate the edge capacities by at most factor α' (since there are at most α' paths containing a vertex as start or intermediate node). Therefore, if we scale this flow down by the factor of α', we will obtain a feasible flow of the desired value. Note that this flow saturates all the edges from receivers to t. From the integrality of flow, we can obtain integral flow of the same value. In particular, all receiver-to-t edges will remain saturated.

The desired set $\mathcal{P}_2$ is obtained by the edge disjoint paths from the senders (light agents) or the vertices in $\mathbf{S}$ to the receivers (light agents). Note that these paths are elementary paths and terminate at light agents. Furthermore, these edge disjoint paths are internally-vertex disjoint as well since every intermediate vertex in a path, a heavy agent or an item, has out-degree equal to 1 or in-degree equal to 1. Finally, if a path in $\mathcal{P}_2$ originates from an agent A, a sender, then since every sender is a receiver, there are $\lfloor N_A/2\alpha' \rfloor$ paths terminating at it as well. □

4 An $\tilde{O}\left(n^{\epsilon}\right)$-Approximation Algorithm: Proof of Theorem 3

In this section we prove Theorem 3; we first give a high level sketch. The algorithm has h (recall $h = 9/\epsilon$) iterations; we start with the LP solution to the original instance and the almost feasible solution obtained from Theorem 4. Using the latter, we identify a set of light agents and in the next iteration we consider the instance with these light agents removed. The crucial observation is that on removing a set of agents the optimum value doesn't decrease and thus the instance obtained remains 1-satisfiable, and we can, after a suitable re-definition of private items, apply Theorem 4 again. We show that either the LP proves infeasibility in one of the iterations, or at the end of h iterations we obtain our collection of elementary paths.

We introduce the following notation. Let $\mathcal{Q}$ be a collection of elementary paths, such that all paths in $\mathcal{Q}$ terminate at light agents. We say that $\mathcal{Q}$ *β-satisfies* a subset $\mathbf{L}' \subseteq \mathbf{L}$ of light agents iff

- the paths in $\mathcal{Q}$ do not share intermediate vertices,
- each light agent $A \in \mathbf{L}'$ has at least N_A/β paths in $\mathcal{Q}$ that terminate at A and no path in $\mathcal{Q}$ originates from A, and
- each light agent $A \in \mathbf{L} \setminus \mathbf{L}'$ has at most one path in $\mathcal{Q}$ originating from it, and if such a path exists, then there are at least N_A/β paths in $\mathcal{Q}$ that terminate at A.

The input to iteration j, $1 \leq j \leq h$, of the algorithm is a subset $\mathbf{L}^j \subseteq \mathbf{L}$ of light agents, a set $\mathbf{T}^j \subseteq \mathbf{H}$ of terminals and an assignment of private items $\pi^j : \mathbf{A} \setminus (\mathbf{L}^j \cup \mathbf{T}^j) \to \mathbf{I}$. We maintain the property that every light agent $A \notin \mathbf{L}^j$ is assigned its heavy item, that is, $\pi^j(A) = h(A)$. Additionally, we have a collection $\mathcal{Q}^j$ of elementary paths in the resulting flow network $N(\mathcal{I}, \pi^j)$, that α_j-satisfy agents in $\mathbf{L}^j$, for $\alpha_j = 2j\alpha$ (here α is the parameter from Theorem 4). The output of iteration j is a valid input to iteration $(j+1)$, that is, sets $\mathbf{L}^{j+1}$,$\mathbf{T}^{j+1}$, an assignment $\pi^{j+1} : \mathbf{A} \setminus (\mathbf{L}^{j+1} \cup \mathbf{T}^{j+1}) \to \mathbf{I}$ of private items, and a collection $\mathcal{Q}^{j+1}$ of elementary paths in $N(\mathcal{I}, \pi^{j+1})$ that α_{j+1}-satisfy $\mathbf{L}^{j+1}$. The size of the set $\mathbf{T}^j$ of terminas decreases in each iteration and after h iterations $\mathbf{T}^{h+1}$ becomes empty, with π^{h+1} assigning a private item to each agent in $\mathbf{A} \setminus \mathbf{L}^{h+1}$. We then use the set $\mathcal{Q}^{h+1}$ of paths in network $N(\mathcal{I}, \pi^{h+1})$ to complete the proof of Theorem 3.

Initialization and Invariants: In the input to the first iteration, $\mathbf{L}^1 = \emptyset$, $\mathcal{Q}^1 = \emptyset$. Each light agent $A \in \mathbf{L}$ is assigned its heavy item as a private item, $\pi^1(A) = h(A)$. The assignment of private

items to heavy agents is performed by computing a maximum matching between the set of heavy agents and the remaining items. Recall that $\mathbf{H}^*$ is the set of heavy agents and $\mathbf{I}^*$ is the set of items directly reachable from the source s in network $N(\mathcal{I}, \pi^1)$, while $\mathbf{S}$ is the set of items that do not serve as private items in π^1. Clearly, each agent in $\mathbf{H}^*$ is assigned an item in $\mathbf{I}^*$ and there are no direct paths from s to any terminal in $\mathbf{T}^1$. Throughout the algorithm, we ensure that for all j, there are no direct paths from s to any terminal $t \in \mathbf{T}^j$ in $N(\mathcal{I}, \pi^j)$.

Iteration j: Iteration j is performed as follows. We construct a canonical instance $\mathcal{I}^j$ that is identical to $\mathcal{I}$ except that we remove the light agents in $\mathbf{L}^j$ from this instance. Let $\mathcal{N}_j = N(\mathcal{I}^j, \pi^j)$ be the corresponding flow network. Note that π^j is a good assignment for this instance since we ensure $\pi^j(A) = h(A)$ for all light agents in $\mathcal{I}_j$. Since we do not remove any items from the instance, if instance $\mathcal{I}$ is 1-satisfiable, so is $\mathcal{I}^j$. Combined with the fact that there are no direct paths from s to terminals in $\mathbf{T}^j$, we can apply Theorem 4 to obtain two sets $\mathcal{P}_1$, $\mathcal{P}_2$ of elementary paths in $\mathcal{N}_j$, satisfying properties D1–D4.

Let $\mathbf{L}'$ be the subset of light agents A for which there is a path in either $\mathcal{P}_1$ or $\mathcal{P}_2$ originating from A. Recall that for any such agent A, there are at least N_A/α paths in $\mathcal{P}_2$ terminating at A. Let $\mathbf{L}''$ be the set of light agents A such that either $A \in \mathbf{L}^j$, or there is a path in $\mathcal{Q}^j$ originating from A. Recall that there are at least N_A/α_j paths terminating at A in $\mathcal{Q}^j$. The remainder of this iteration consists of three steps. In Step 1, we use paths in $\mathcal{P}_2 \cup \mathcal{Q}^j$ to produce a new set $\mathcal{Q}^*$ of paths, such that every agent in $\mathbf{L}' \cup \mathbf{L}''$ has at least $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ paths in $\mathcal{Q}^*$ terminating at it, and at most one path leaving it. Moreover, the only light agents from which such paths originate lie in $(\mathbf{L}' \cup \mathbf{L}'') \setminus \mathbf{L}^j$. In Step 2 we re-route paths in $\mathcal{P}_1$, so that each new path intersects at most one path in $\mathcal{Q}^*$. Let $\mathcal{P}'$ be the resulting set. Next we obtain $\mathcal{Q}' \subseteq \mathcal{Q}^*$ by removing from $\mathcal{Q}^*$ all paths intersecting paths in $\mathcal{P}'$. In Step 3 we use sets $\mathcal{P}'$ and $\mathcal{Q}'$ to produce input to iteration $(j+1)$.

Step 1: Combining $\mathcal{Q}^j$ and $\mathcal{P}_2$. This step is summarized in the next lemma.

Lemma 7 *We can find, in polynomial time, a set of internally-disjoint elementary paths $\mathcal{Q}^*$ in $N(\mathcal{I}, \pi^j)$ such that each agent $A \in \mathbf{L}' \cup \mathbf{L}''$ has at least $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ paths terminating at A. Moreover, only light agents in $(\mathbf{L}' \cup \mathbf{L}'') \setminus \mathbf{L}^j$ have paths in $\mathcal{Q}^*$ originating from them, with at most one path originating from any agent.*

Proof: This is done similarly to Lemma 6 – we form a network where light agents in $\mathbf{L}' \cup \mathbf{L}''$ serve as receivers and light agents in $(\mathbf{L}' \cup \mathbf{L}'') \setminus \mathbf{L}^j$ are the senders. Note that we have two copies of agents in $\mathbf{L}' \cup \mathbf{L}'' \setminus \mathbf{L}^j$. Additionally, there is a vertex for every non-terminal heavy agent and item, and a source s and sink t. Each receiver A is connected to a sink t with $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ parallel edges of capacity 1. The source s connects to all senders and items in $\mathbf{S}$ with edges of capacity 1. There is an edge from each sender to its private item (note that the agents in $\mathbf{L}^j$ are not senders) and from every heavy agent to its private item. There is an edge from item i to heavy agent A iff $i \in \Gamma_H(A)$, and there is an edge from item i to a receiver A iff i is a light item for A. Each such edge has capacity 1.

Note that all the paths in $\mathcal{P}_2$ and $\mathcal{Q}^j$ are also paths in this network. Consider the flows defined by paths in $\mathcal{P}_2$ and $\mathcal{Q}^j$. We send $\alpha/(\alpha_j+\alpha)$ flow units from s to t along each path in $\mathcal{P}_2$ and $\alpha_j/(\alpha_j+\alpha)$ flow units along each path in $\mathcal{Q}^j$. The resulting flow causes congestion of at most 1 on the edges, and each receiver A gets at least $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ flow units. From the integrality of flow, and as in the proof of Lemma 6, there is a collection $\mathcal{Q}^*$ of desired paths. □

Step 2: Re-Routing paths in $\mathcal{P}_1$. We say that elementary paths p, q intersect iff they either share intermediate vertices, or they start from the same vertex. Notice that each path in $\mathcal{P}_1$ may

intersect many paths in $\mathcal{Q}^*$. In our next step, we re-route paths in $\mathcal{P}_1$ to get a set $\mathcal{P}'$ of paths, so that, roughly speaking, each new path intersects at most one path in $\mathcal{Q}^*$. We formalize this as the following lemma.

Lemma 8 *We can find, in polynomial time, a set $\mathcal{P}'$ of disjoint paths, and a partition $\mathcal{Q}^* = \mathcal{Q}' \cup \mathcal{Q}''$, such that each path in $\mathcal{P}'$ starts at a distinct light agent in $\mathbf{L}' \cup \mathbf{L}'' \setminus \mathbf{L}^j$ and ends at a distinct terminal in $\mathbf{T}^j$, and each terminal in $\mathbf{T}^j$ has one path terminating at it. Moreover, $|\mathcal{Q}''| \leq |\mathcal{P}'|$, and paths in $\mathcal{P}'$ do not intersect paths in $\mathcal{Q}'$.*

Proof: We start with a path re-routing lemma. For a directed path p starting at some vertex v, we say that path p' is a *prefix* of p iff p' is a sub-path of p containing v.

Lemma 9 *Let $\mathcal{P}, \mathcal{Q}$ be two collections of directed paths, such that all paths in $\mathcal{P}$ are completely vertex disjoint and so are all paths in $\mathcal{Q}$. We can find, in polynomial time, for each path $p \in \mathcal{P} \cup \mathcal{Q}$, a prefix $\gamma(p)$, such that if $\mathcal{C}$ is a connected component in the graph G_γ defined by the union of prefixes $\{\gamma(p) \mid p \in \mathcal{P} \cup \mathcal{Q}\}$, then*

- *either $\mathcal{C}$ only contains vertices belonging to a single prefix $\gamma(p)$ and $\gamma(p) = p$, or*
- *$\mathcal{C}$ contains vertices of exactly two prefixes $\gamma(p)$ and $\gamma(q)$, where $p \in \mathcal{P}$, $q \in \mathcal{Q}$, and the two prefixes have exactly one vertex in common, which is the last vertex of both $\gamma(p)$ and $\gamma(q)$.*

We provide the proof of Lemma 9 below, and first complete the proof of Lemma 8. Recall that the paths in $\mathcal{P}_1$ are completely vertex-disjoint. Paths in $\mathcal{Q}^*$ may share endpoints, but for each agent A there is at most one path in $\mathcal{Q}^*$ that originates from A. In order to apply Lemma 9 however we need to ensure that paths in $\mathcal{Q}^*$ are completely vertex-disjoint. For each path $q \in \mathcal{Q}^*$, if A is the last vertex on q, we introduce a new dummy vertex $v(q, A)$ that replaces A on path q. Let $\mathcal{Q}^{**}$ be the resulting set of paths. We also transform set $\mathcal{P}_1$ as follows. Given a directed path p, we denote by $\overline{p}$ the path obtained by reversing the direction of all edges of p. We define $\overline{\mathcal{P}}_1 = \{\overline{p} \mid p \in \mathcal{P}_1\}$. In the new set $\overline{\mathcal{P}}_1$, the paths originate from the terminals and terminate at light agents. We now apply Lemma 9 to $\overline{\mathcal{P}}_1$ and $\mathcal{Q}^{**}$ and obtain prefix $\gamma(p)$ for each $p \in \overline{\mathcal{P}}_1 \cup \mathcal{Q}^{**}$.

We start with $\mathcal{Q}'' = \emptyset$. For each $p \in \overline{\mathcal{P}}_1$, we construct a new path p' in $\mathcal{P}'$ as follows. Consider the connected component $\mathcal{C}$ in the graph G_γ induced by the prefixes, to which $\gamma(p)$ belongs. If $\mathcal{C}$ only contains vertices of $\gamma(p)$, then $p = \gamma(p)$ and we set $p' = \overline{p}$. Otherwise, $\mathcal{C}$ contains vertices of p and of another path $q \in \mathcal{Q}^{**}$. Consider the vertex v that is common to $\gamma(p)$ and $\gamma(q)$. If v is a light agent, then since p and q are elementary paths, v must be the last vertex in p implying $\gamma(p) = p$. We set $p' = \overline{p}$. Otherwise, we set p' to be the concatenation of $\gamma(q)$ and $\overline{\gamma(p)}$. In either case we add q to $\mathcal{Q}''$.

Now observe that the first vertex of q has to be some light agent: otherwise, it is the source s, and then the path $\gamma(q)$ followed by $\overline{\gamma(p)}$ is an elementary path from s to a terminal, which is impossible by the invariant of our algorithm. Therefore, in both cases above, the new path p' starts from a light agent, A and ends at a terminal, T. Note that A lies in $\mathbf{L}' \cup \mathbf{L}'' \setminus \mathbf{L}^j$; in the first case there is a path in $\mathcal{P}_1$ starting from A implying $A \in \mathbf{L}'$ which is disjoint from $\mathbf{L}^j$; in the second case there is a path in $\mathcal{Q}^*$ originating from A implying $A \in \mathbf{L}' \cup \mathbf{L}'' \setminus \mathbf{L}^j$. Furthermore, since connected components of G_γ are vertex disjoint, no two paths in $\mathcal{P}'$ share the originating light agent. Finally, since we obtain a path in $\mathcal{P}'$ for each path in $\mathcal{P}_1$ and the terminating vertex remains the same, for each terminal we have a path in $\mathcal{P}'$ ending at it.

Each path in $\mathcal{P}'$ is responsible for adding at most one path to $\mathcal{Q}''$, therefore $|\mathcal{Q}''| \leq |\mathcal{P}'|$. Define $\mathcal{Q}' := \mathcal{Q}^* \setminus \mathcal{Q}''$. Note that for every $q \in \mathcal{Q}'$, we have $\gamma(q) = q$. This is because all paths q, for which $\gamma(q)$ intersects some $\gamma(p)$ have been added to $\mathcal{Q}''$. Thus, paths in $\mathcal{Q}'$ do not intersect paths in $\mathcal{P}'$. This completes the proof. □

Since $|\mathcal{Q}''| \leq |\mathcal{P}'| = |\mathbf{T}^j|$, we can find a 1-1 mapping $f : \mathcal{Q}'' \to \mathbf{T}^j$. If $f(q) = t$ for $q \in \mathcal{Q}''$ and $t \in \mathbf{T}^j$, we say that t is *responsible* for q. Note that we now have a set $\mathcal{Q}' \cup \mathcal{P}'$ of elementary paths that almost has the desired properties of Theorem 3. All paths in $\mathcal{P}'$ connect light agents to terminals and are vertex disjoint; all paths in $\mathcal{Q}'$ are internally vertex disjoint. Each light agent A has at most one path in $\mathcal{Q}' \cup \mathcal{P}'$ leaving it, and if such a path is present, then $\mathcal{Q}^*$ contains at least $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ paths terminating at it, while all light agents in $\mathbf{L}^j$ are $(\alpha_j + \alpha)$-satisfied by $\mathcal{Q}^*$. The problem is that set $\mathcal{Q}''$ may contain many paths from $\mathcal{Q}^*$, and so a light agent may have a path in $\mathcal{P}' \cup \mathcal{Q}'$ leaving it, but not enough paths in $\mathcal{Q}'$ entering it. We take care of this problem and define the input to the next iteration in the next step.

Before going to the next step, we give a proof of the path re-routing lemma.

Proof: (of Lemma 9) The proof of this lemma is similar to proof of Theorem 3.1 in [9] which the authors use to reconstruct edge-disjoint paths between pairs of vertices in an undirected graph.

As in [9], we prove the lemma via the stable matching theorem. Given the sets $\mathcal{P}$ and $\mathcal{Q}$ of paths, construct a bipartite multigraph $G(\mathcal{P}, \mathcal{Q}, E)$. If paths $p \in \mathcal{P}$ and $q \in \mathcal{Q}$ share k vertices, then we add k parallel edges between p and q. Each such edge is labeled with the corresponding vertex common to p and q.

Given a vertex $p \in \mathcal{P}$, its preference order over the edges incident on it is as follows. If two edges (possibly parallel) with labels u and v are incident on p, then p prefers edge with label u to edge with label v iff u appears before v on p (notice that there can be several edges with identical labels incident on p; the preference order for such edges is arbitrary). The preference order of edges incident to a vertex $q \in \mathcal{Q}$ is determined similarly.

Consider any matching M in graph $G(\mathcal{P}, \mathcal{Q}, E)$. An edge (p, q) with label v is called *rogue* iff:

- p does not participate in M, or an edge incident on p with some label u belongs to M, while p prefers v to u, and
- q does not participate in M, or an edge incident on q with some label u' belongs to M, while q prefers v to u'.

Matching M is *stable* iff there are no rogue edges in $G(\mathcal{P}, \mathcal{Q}, E)$. [9] gave a polynomial time algorithm to compute a stable matching in a bipartite multigraph; this is a standard generalization of the Gale-Shapley algorithm to find a stable marriage.

Given a stable matching M, we construct the prefixes of the paths in $\mathcal{P}$ and $\mathcal{Q}$ as follows. If some path $p \in \mathcal{P} \cup \mathcal{Q}$ does not participate in M, then we set $\gamma(p) = p$. Otherwise, for every pair (p, q) of paths, such that an edge connecting p and q with label v belongs to M, we set $\gamma(p)$ to be the prefix of p ending at v, and $\gamma(q)$ to be the prefix of q ending at v. Note that $\gamma(p)$ and $\gamma(q)$ are internally disjoint; if they intersected at a vertex u other than v, then the edge (p, q) labeled u would be a rogue edge.

Pick any path $p \in \mathcal{P} \cup \mathcal{Q}$ and consider $\gamma(p)$. We now show that if for any q, $\gamma(q)$ intersects $\gamma(p)$, then the common point must be the end point of both $\gamma(p)$ and $\gamma(q)$, and for every other $q' \neq p, q$,

$\gamma(p)$ and $\gamma(q')$ are disjoint. This completes the proof of the lemma.

Suppose $p \in \mathcal{P}$. If $\gamma(q)$ intersects $\gamma(p)$, $q \in \mathcal{Q}$ since $\mathcal{P}$ was a set of vertex disjoint paths. We now claim q is the *unique* path matched to p in the stable matching M, and the point of intersection is the last vertex,v, on both $\gamma(p)$ and $\gamma(q)$. Suppose not. If p and q are both unmatched, then clearly the edge (p, q) is rogue. If q is matched to $p' \in \mathcal{P}$, then $\gamma(p)$ and $\gamma(q)$ intersect at u and u is *not* the last vertex of $\gamma(q)$. This is because the last vertex of $\gamma(q)$ is common to p' and thus cannot be on p, since they are vertex disjoint. Thus, q prefers (p, q) to (p', q). Similarly, if p is matched to q', p prefers (q, p) to (q', p); or p is unmatched. In any case, (p, q) forms a rogue edge. Therefore, (p, q) is in the matching M. By definition of $\gamma(p)$ and $\gamma(q)$, these intersect at their last vertex v. □

Step 3: Producing Input to Iteration $(j + 1)$**.** We call a light agent A *bad* iff there is path in $\mathcal{P}' \cup \mathcal{Q}'$ originating at A and there are less than $N_A/(\alpha_j + 2\alpha) = N_A/\alpha_{j+1}$ paths in $\mathcal{Q}'$ terminating at A, or $A \in \mathbf{L}^j$ and there are less than $N_A/(\alpha_j + 2\alpha) = N_A/\alpha_{j+1}$ paths in $\mathcal{Q}'$ terminating at A. We now perform the following procedure that will define the new set $\mathbf{T}^{j+1}$ of terminals. We initialize $\mathbf{T}^{j+1} = \emptyset$. While there exists a bad light agent A:

- Remove all paths entering A from $\mathcal{Q}'$. Note that this doesn't make any new light agent bad.
- If $A \notin \mathbf{L}^j$, then remove from $\mathcal{P}'$ or $\mathcal{Q}'$ the unique path p leaving A. and say that A is *responsible* for this path. If $p \in \mathcal{P}'$ and $t \in \mathbf{T}^j$ is the terminal lying on p, then we add t to $\mathbf{T}^{j+1}$ and say that A is responsible for t. Note that removal of p could lead to new bad agents.
- If $A \in \mathbf{L}^j$, consider the item $i = h(A)$. If there is a heavy agent A' for which i is a private item, we add A' to $\mathbf{T}^{j+1}$ (where it becomes a terminal). In either case, item i becomes the private item for A. We remove A from $\mathbf{L}^j$ and say it is responsible for A'.

  If there is any path p containing i in $\mathcal{P}' \cup \mathcal{Q}'$, then we remove p from $\mathcal{P}'$ or $\mathcal{Q}'$ and say that A is responsible for p. If $p \in \mathcal{P}'$ and $t \in \mathbf{T}^j$ is a terminal lying on p, then we add t to $\mathbf{T}^{j+1}$ and say that A is responsible for t. The removal of p could lead to new bad agents.

Note that a bad light agent can only be responsible for at most two terminals in $\mathbf{T}^{j+1}$, and for the removal of at most one path from $\mathcal{P}' \cup \mathcal{Q}'$. Once we take care of a bad light agent A, this could result in another agent A' becoming a bad light agent. We repeat this process until no bad light agents remain. We now show how to produce the input to the next iteration.

Input to Iteration $(j + 1)$**:** We start with set $\mathbf{L}^{j+1}$ containing all the remaining good agents in $\mathbf{L}^j$. Consider now the residual sets $\mathcal{P}' \cup \mathcal{Q}'$ of paths. Let $p \in \mathcal{P}'$, and let A be the first vertex and $t \in \mathbf{T}^j$ be the last vertex on p. We then add A to $\mathbf{L}^{j+1}$ and re-assign private items that lie on path p as follows. If A' is the agent lying immediately after item i on path p then i becomes a private item for A'. The assignment of private items to agents not lying on any path in $\mathcal{P}'$ remains the same. Let π^{j+1} be the resulting assignment of private items after we process all paths in $\mathcal{P}'$. Note that the only agents with no private items assigned are agents in the set $\mathbf{L}^{j+1} \cup \mathbf{T}^{j+1}$. We set $\mathcal{Q}^{j+1} = \mathcal{Q}'$. Since no light agent in $\mathbf{L}^{j+1}$ is bad, set $\mathcal{Q}^{j+1}$ ensures that every agent in $\mathbf{L}^{j+1}$ is α_{j+1}-satisfied.

The next lemma shows that in $N(\mathcal{I}^{j+1}, \pi^{j+1})$ the terminals $\mathbf{T}^{j+1}$ are not directly connected to the items not assigned as private items by π^{j+1}.

Lemma 10 *Let* $\mathbf{S}$ *be the set of items that are not assigned to any agent by* π^1*, and let* $\mathbf{H}^*$ *be the set of heavy agents reachable by direct paths from* s *in* $N(\mathcal{I}, \pi^1)$*. Then for each iteration* j*, the set of items that are not assigned to any agent by* π^j *is* $\mathbf{S}$*, and for each* $A \in \mathbf{H}^*$*,* $\pi^j(A) = \pi^1(A)$*.*

Proof: Recall that there are no direct paths from s to any terminal in $\mathbf{T}^1$ in $N(\mathcal{I}^1, \pi^1)$. Since the set of items not assigned to any agent by π^j remains $\mathbf{S}$, and since for every $A \in \mathbf{H}^*$, $\pi^j(A) = \pi^1(A)$, there are no direct paths from s to any terminal in $\mathbf{T}^j$ in $N(\mathcal{I}^j, \pi^j)$.

The proof is by induction. Assume that the lemma holds for iterations $1, \ldots, j$, and consider iteration $(j+1)$. Recall that we only change π^j while constructing π^{j+1} in the following cases:

- For each path $p \in \mathcal{P}'$, we re-assign the items along the path.
- For bad agents $A \in \mathbf{L}^j$, if $h(A) = i$ and $\pi^j(A') = i$ for some A', then we set $\pi^{j+1}(A) = i$ and A' becomes a terminal in $\mathbf{T}^{j+1}$.

Consider first some agent $A \in \mathbf{H}^*$ and assume that $\pi^j(A) \neq \pi^{j+1}(A)$. Then either A lies on some path in $\mathcal{P}'$, and then A can reach a terminal $t \in \mathbf{T}^j$ directly, so there is a direct path from s to $t \in \mathbf{T}^j$ in $N(\mathcal{I}^j, \pi^j)$, a contradiction. Or $\pi^j(A) = i$ where $i = h(A')$ for some light agent $A' \in \mathbf{L}^j$. But we know that $\pi^1(A) \neq i$ since π^1 assigns i to A', and since we assumed that $\pi^j(A) = \pi^1(A)$, this is impossible.

It is also easy to see that an item $i \in \mathbf{S}$ cannot be assigned to any agent by π^{j+1}, since we only re-assign items along the paths in $\mathcal{P}'$, or items $i = h(A)$ for $A \in \mathbf{L}^j$. Clearly, an item on a path in $\mathcal{P}'$ cannot belong to $\mathbf{S}$, as such an item has a direct path connecting it to a terminal in T^j in graph $N(\mathcal{I}^j, \pi^j)$. Otherwise, if $i = h(A)$ for $A \in \mathbf{L}^j$, item i cannot belong to $\mathbf{S}$, since it is assigned to A by π_1 and we assume that $\mathbf{S}$ has not changed throughout iterations $1, \ldots, j$. Finally we need to argue that no new items are added to $\mathbf{S}$. This is also easy to see since we only perform re-assignment of items that have already been assigned by π^j, and each item that was assigned to some agent in π^j is assigned to some agent by π^{j+1}. □

Therefore, π^{j+1} is a good assignment of private items for instance $\mathcal{I}^j$ maintaining all the invariants; and thus we have produced a feasible input to iteration $(j+1)$. The lemma below bounds the size of $\mathbf{T}^{j+1}$.

Lemma 11 $|\mathbf{T}^{j+1}| \leq \left(\frac{32h^2\alpha}{n^\epsilon}\right) |\mathbf{T}^j|$.

Proof: Since $\epsilon \geq 9 \log\log n / \log n$, $h = 9/\epsilon$ and $\alpha = O(h^5 \log n)$, we have that $n^\epsilon \geq \log^9 n$ and so $n^\epsilon \geq 16h^2\alpha$. Recall that each bad light agent is responsible for at most two terminals in $\mathbf{T}^{j+1}$. Therefore, it is enough to prove that the number of bad light agents in iteration j is at most $\left(\frac{16h^2\alpha}{n^\epsilon}\right) |\mathbf{T}^j|$. We build a graph G_B whose vertices are bad light agents and the terminals in $\mathbf{T}^j$. Consider now some bad light agent A. Originally there were at least $\left\lfloor \frac{N_A}{(\alpha_j+\alpha)} \right\rfloor \geq \frac{n^\epsilon}{(2j+1)\alpha} - 1$ paths entering A in $\mathcal{Q}^*$. Since A is a bad light agent, eventually less than $n^\epsilon/((2j+2)\alpha)$ paths remained. Therefore, at least $\frac{n^\epsilon}{(2j+1)(2j+2)\alpha} - 1 \geq \frac{n^\epsilon}{8h^2\alpha}$ paths have been removed from $\mathcal{Q}^*$. If q is a path entering agent A that was removed, and an agent A' is responsible for the removal of q, then we add an edge from A to A' in graph G_B. Notice that if q is a path in $\mathcal{Q}''$, then there is some agent $A' \in \mathbf{T}^j$ that is responsible for its removal. Otherwise, q is a path that was removed during the processing of bad light agents, and we had identified a unique bad light agent A' to be responsible for its removal.

Since any bad light agent or terminal in $\mathbf{T}^j$ is responsible for the removal of at most one path, the in-degree of every vertex is at most 1. Moreover by the discussion above, the out-degree of every bad light agent is at least $\beta = \frac{n^\epsilon}{8h^2\alpha} \geq 2$. Note that the out-degree of terminals in $\mathbf{T}^j$ is 0. Let n_B be

the number of bad light agents in G_B. Since the sum of in-degrees equal the sum of out-degrees, we have $n_B + |\mathbf{T}^j| \geq \beta n_B$ implying the number of bad light agents $n_B \leq |\mathbf{T}^j|/(\beta - 1) \leq 2|\mathbf{T}^j|/\beta$ since $\beta \geq 2$. □

Thus, after h iterations, $|\mathbf{T}^{h+1}| \leq |\mathbf{T}^1|/(n^\epsilon/(32h^2\alpha))^h$. Since $h \leq \log n/\log\log n$, $n^\epsilon \geq \log^9 n$ and $\alpha = O(h^5 \log n)$, we have that $32h^2\alpha = O(h^7 \log n) = O(\log^8 n) = O(n^{8\epsilon/9})$. Thus, for large enough n, $(n^\epsilon/(32h^2\alpha))^h > (n^{\epsilon/9})^h = n$. Therefore, after h iterations $\mathbf{T}^{h+1} = \emptyset$.

Finishing proof of Theorem 3: If in any iteration the LP of Theorem 4 is infeasible, then we conclude that $\mathcal{I}$ is not 1-satisfiable. Otherwise, we can run $(h+1)$ iterations of the previous algorithm.

At the end of iteration $(h+1)$, we have an assignment π^{h+1} of private items to all agents, except for those in $\mathbf{L}^{h+1}$, and a set $\mathcal{Q}^{h+1}$ of paths in the network $N(\mathcal{I}, \pi^{h+1})$, which α_{h+1} satisfy agents in $\mathbf{L}^{h+1}$. We convert it into a good assignment π as follows. This will lead to a terminal set $\mathbf{T}$, and we construct the collection of elementary paths $\mathcal{P}$ satisfying the statement of the theorem.

We initialize $\pi = \pi^{h+1}$, $\mathbf{T} = \emptyset$, and $\mathcal{P} = \mathcal{Q}^{h+1}$. For every $A \in \mathbf{L}^{h+1}$, consider the item $i = h(A)$. If there is a heavy agent A' for which i is a private item, we add A' to $\mathbf{T}$ and modify $\pi(A) = i$ and $\pi(A') = \emptyset$.

Two cases arise. If there is no path $p \in \mathcal{P}$ containing i, then we add the path $(A \to i \to A')$ to $\mathcal{P}$. If there is any path p containing i in $\mathcal{P}$, then we do the following. Let p' be the sub-path from the start of p to agent A'. Let p'' be the sub-path from i to the end of path p. We remove p from $\mathcal{P}$. We add p' to $\mathcal{P}$. Thus there is a unique path ending at this terminal. We add the path $p''' := A \to p''$ to $\mathcal{P}$.

The final private assignment is π, the final set of terminals is $\mathbf{T}$ and the set of elementary paths is $\mathcal{P}$. Note that paths in $\mathcal{P}$ are internally vertex disjoint since paths in $\mathcal{Q}^{h+1}$ were, and the new paths added in $\mathcal{P}$ do not destroy this property. Thus ($\hat{\text{C}}1$) holds. Once we create a terminal in $\mathbf{T}$, we also create a path in $\mathcal{P}$ which terminates at it, ensuring ($\hat{\text{C}}2$) holds. Finally, if a path originates from a light agent, then it has N_A/α_{h+1} paths terminating at it. This is true for paths in $\mathcal{Q}^{h+1}$. It is also true for $\mathcal{P}$ since the new paths originate from $\mathbf{L}^{h+1}$ whiche were α_{h+1}-satisfied. Since $\alpha_{h+1} = O(h\alpha) = O(h^6 \log n)$ and $h = O(1/\epsilon)$, the proof of Theorem 3 follows.

The algorithm in Theorem 4 runs in $n^{O(1/\epsilon)}$ time, and the algorithm presented in this section performs $O(1/\epsilon)$ number of calls to Theorem 4. Therefore, the overall running time of the algorithm is $n^{O(1/\epsilon)}$. This completes the $O(n^\epsilon \log n)$ algorithm for the Max-Min Allocation problem. □

5 A Quasi-Polynomial Time $O(m^\epsilon)$-Approximation Algorithm

Theorem 1 implies a $O(\log^{10} n)$-approximation algorithm in quasi-polynomial ($n^{O(\log n)}$) time. Using that, we show that it is possible to obtain an $O(m^\epsilon)$-approximation in quasi-polynomial time for any fixed $\epsilon > 0$. We start with the following easy lemma.

Lemma 12 *There exists an* $(\log n)^{O(m \log n)}$*-time* $O(1)$*-approximation algorithm for* Max-Min Allocation.

Proof: From Section 2 we can assume all the utilities $u_{A,i}$ to be between 1 and $2n$. By losing another constant factor in the approximation, we round down all the utilities to the nearest power of 2. Thus we can assume from here on that there are $O(\log n)$ distinct values of utilities.

Fix an optimal solution OPT. For every agent A, we let $v_j(A)$ be the number of items assigned to A in OPT whose utility for A is 2^j, for $j = 1 \ldots s = \lfloor \log 2n \rfloor$. Thus there exists an $O(1)$-approximate solution that can be described a collection of s-dimensional vectors, one for each agent. For an agent A, let $v(A) := (v_1(A), \ldots, v_s(A))$ denote the s-dimensional vector associated with it. By losing another factor of 2, we can further assume that each $v_j(A)$ has been rounded down to the nearest power of 2. Therefore, for every agent there are at most $(\log n)^s$ possible vectors $v(A)$, and one of them corresponds to the optimal solution.

We now show how, given vector $v(A)$ for every agent A, we can check if there is a feasible assignment of the items respecting these vectors, so that each agent A is assigned $v_j(A)$ items with utility $u_{A,i} = 2^j$. Construct a bipartite graph $G(U, V, E)$ where U contains s copies of each agent A, say $A(1), \ldots, A(s)$, and the vertex set V corresponds to the set of items. There is an edge from $A(j)$ to an item i iff $u_{A,i} = 2^j$. The problem of checking whether a given collection of vectors $v(A)$ can be realized is equivalent to testing if there exists a matching from U to V such that every vertex $A(j)$ in U has exactly $v_j(A)$ edges incident on it, and every vertex in V has at most one edge incident on it. This can be done in polynomial time.

Thus in time $(\log n)^{O(m \log n)}$ (over all the choices of vectors for all agents), we can get an $O(1)$ approximation to Max-Min Allocation. □

Using the above claim we can get the following.

Theorem 6 (Corollary 1) *For any constant $\epsilon > 0$, there exists a quasi-polynomial time algorithm which gives a $O(m^\epsilon)$-approximation to* Max-Min Allocation.

Proof: If $m < \log^{10/\epsilon} n$, then by the claim above we can get a $O(1)$-approximation in $(\log n)^{O(m \log n)}$ time which is quasi-polynomial for any fixed $\epsilon > 0$. If $m \geq \log^{10/\epsilon} n$, then our main result gives a quasi-polynomial time $O(\log^{10} n) = O(m^\epsilon)$-factor algorithm. □

6 The 2-Restricted Max-Min Allocation problem

In this section we focus on the restricted version of Max-Min Allocation, where each item i is wanted by at most 2 agents A_i and B_i (not necessarily distinct). We give a 2-approximation for the above problem. We also show in Appendix B that even the Santa Claus version of the problem, where each item has the same utility for each agent who wants it, is NP-hard to approximate to within a factor better than 2. Such a theorem was also proved by [4] and follows from the reduction of [10]. We include the proof for completeness.

Given a 2-restricted Max-Min Allocation instance $\mathcal{I} = (\mathbf{A}, \mathbf{I})$, construct a graph $G(\mathcal{I}) = (\mathbf{A}, \mathbf{I})$ where an item i is an edge between agents A_i and B_i. Such a graph can have self-loops and parallel edges. An allocation of items can be represented by an orientation of the edges of this graph. The weights on the edges are non-uniform, that is, an edge has different weights for its two end points.

For agent $A \in \mathbf{A}$, let $\delta(A)$ denote the set of adjacent edges in $G(\mathcal{I})$, that is, $\delta(A) := \{i : u_{A,i} > 0\}$. Given a parameter $M > 0$, we define the following system $LP(M)$ of linear inequalities.

$$\forall i \in \mathbf{I} \quad x_{A_i,i} + x_{B_i,i} \leq 1 \tag{24}$$

$$\forall A \in \mathbf{A}, \forall S \subseteq \delta(A) \quad \sum_{i \notin S} \hat{u}_{A,i} x_{A,i} \geq M - u_A(S) \tag{25}$$

where $u_A(S) := \sum_{i \in S} u_{A,i}$ and $\hat{u}_{A,i} := \min(u_{A,i}, M - u_A(S))$. The first inequality states that each item is allocated to at most one agent. The second states that, given any set S of items, the total utility of items *not in* S allocated to agent A must be at least $(M - u_A(S))$. Furthermore, this should also be true if the utility of any individual item is capped to $\hat{u}_{A,i}$ which is the minimum of $u_{A,i}$ and $(M - u_A(S))$. This type of valid inequalities is called *knapsack cover* inequalities (see, for instance, [8]).

Let M^* be the largest value M for which the above LP is feasible, so $M^* \geq \mathsf{OPT}$. Note that this LP has exponentially many constraints of type (25). In general, it is sufficient to produce a separation oracle in order to solve this LP in polynomial time by the Ellipsoid method. We do not produce such an oracle. Rather, we show that if a solution x satisfies the inequality (25) for a single set S per agent, the set depending on the solution x, then we can obtain an integral allocation such that each agent gets utility at least $M^*/2$. Furthermore, we can check, given x, whether x satisfies these particular inequalities, in polynomial time. Therefore, in each iteration, we either get a desired x and thus a desired integral allocation; or we obtain a *separating inequality*. The ellipsoid method guarantees that in a polynomial number of steps either the former happens, or we prove that the LP is infeasible.

Given a solution x, we now describe the set of inequalities that we need to check. We say that an item i is *integrally allocated* to agent A, if $x_{A,i} = 1$. For every agent $A \in \mathbf{A}$, let $\mathbf{I}(A)$ denote the set of items integrally allocated to A. An item i is said to be *fractionally allocated* if it is not allocated integrally to any agent. Let $\mathbf{I}'$ be the set of items allocated fractionally. For every agent A, let $\delta'(A)$ be the set of items fractionally allocated to it, that is, $\delta'(A) = \{i \in \mathbf{I}' : x_{A,i} > 0\}$. Let i^*_A be the item in $\delta'(A)$ with maximum value $u_{A,i}$. The inequality that needs to be checked for agent A is (25) with $S = (\mathbf{I}(A) \cup \delta'(A)) \setminus \{i^*_A\}$. From the above discussion, we can find in polynomial time a solution x which satisfies the constraints (24) of LP(M^*) and the following constraints

$$\forall A \in \mathbf{A}, S = (\mathbf{I}(A) \cup \delta'(A)) \setminus \{i^*_A\}, \quad \sum_{i \notin S} \hat{u}_{A,i} x_{A,i} \geq M - u_A(S) \tag{26}$$

Given such an x, we allocate the items in $\mathbf{I}(A)$ to agent A and define $M_A := M - u_A(\mathbf{I}(A))$. We now focus on the sub-graph H of $G(\mathcal{I})$ induced by the edges corresponding to items in $\mathbf{I}'$. We remove from H all isolated vertices. Observe that H does not contain self-loops but may contain parallel edges. It now suffices to allocate items of $\mathbf{I}'$ to agents of H such that every agent A gets a utility of at least $M_A/2$. We start with the following observation about H.

Claim 2 *For every agent A, $u_A(\delta'(A) \setminus \{i^*_A\}) \geq M_A$.*

Proof: Assume otherwise, and let A be the violating agent. Then for $S = (\mathbf{I}(A) \cup \delta'(A)) \setminus \{i^*_A\}$, $u_A(S) < M$. Inequality (26) is then contradicted, since its LHS is precisely $\hat{u}_{A,i^*_A} x_{A,i^*_A}$, which is strictly less than $(M - u_A(S))$ if $M - u_A(S) > 0$, because by definition $x_{A,i^*_A} < 1$. □

The above claim implies that for every agent $A \in V(H)$, the (non-uniform) weighted degree of A is at least $(M_A + \max_{i \in \delta'(A)} \{u_{A,i}\})$. Given a graph $G = (V, E)$, for each vertex $v \in V$, we denote by $\Delta(v)$ the set of edges incident on v, and given an orientation $\mathcal{O}$ of its edges, we denote by $\Delta^-_{\mathcal{O}}(v)$ and $\Delta^+_{\mathcal{O}}(v)$ the set of incoming and outgoing edges for v, respectively. The following theorem about weighted graph orientations will complete the proof.

Theorem 7 *Given a non-uniformly weighted undirected graph $G(V,E)$ with weights $w_{u,e}$ and $w_{v,e}$ for every edge $e=(u,v)\in E$, there exists an orientation $\mathcal{O}$ such that in the resulting digraph, for every vertex v: $\sum_{e\in\Delta^-_{\mathcal{O}}(v)} w_{v,e} \geq \frac{1}{2}\left(\sum_{e\in\Delta(v)} w_{v,e} - \max_{e\in\Delta(v)}\{w_{v,e}\}\right)$. Moreover, such an orientation can be found efficiently.*

Proof: The proof is by induction on the number of edges in the graph. If the graph only contains one edge, the theorem is clearly true. Consider now the case that there is some vertex $u\in V$ with $|\Delta(u)|=1$. Let $e=(u,v)$ be the unique edge incident on u. We can direct e towards v, and by induction there is a good orientation of the remaining edges in the graph. Therefore we assume that every vertex in the graph has at least two edges incident on it. For each vertex $v\in V$, we denote by $e_1(v)\in\Delta(v)$ the edge e with maximum value of $w_{v,e}$ and by $e_2(v)$ the edge with second largest such value. Notice that $e_1(v)$ and $e_2(v)$ are both well defined, and it is possible that they are parallel edges. We need the following claim.

Claim 3 *We can efficiently find a directed cycle $C=(v_1,v_2,\ldots,v_k=v_1)$, where for each $j: 1\leq j\leq k-1$, $h_j=(v_j,v_{j+1})\in E$, and either:*

1. *$w_{v_j,h_{j-1}}\geq w_{v_j,h_j}$, or*
2. *$h_j=e_1(v)$ and $h_{j-1}=e_2(v)$*

We first show that the above claim finishes the proof of the theorem. Let C be the directed cycle from the above claim. We remove the edges of C from the graph and find the orientation of the remaining edges by induction. We then return the edges of C to the graph, with edge $h_j=(v_j,v_{j+1})$ oriented towards v_{j+1}, for all j. By rearranging the inequality that we need to prove for each vertex v we obtain the following expression:

$$\sum_{e\in\Delta^-_{\mathcal{O}}(v)} w_{v,e} + \max_{e\in\Delta(v)}\{w_{v,e}\} \geq \sum_{e\in\Delta^+_{\mathcal{O}}(v)} w_{v,e}$$

Consider some vertex $v\in V$. If v does not lie on the cycle C, then by induction hypothesis the inequality holds for v. Assume now that $v=v_j\in C$. In the orientation of edges of $E\setminus C$ the above inequality holds by induction hypothesis. We need to consider two cases. If $w_{v_j,h_{j-1}}\geq w_{v_j,h_j}$, then since h_{j-1} is added to $\Delta^-_{\mathcal{O}}(v)$ and h_j is added to $\Delta^+_{\mathcal{O}}(v)$, the inequality continues to hold.

Assume now that $h_j=e_1(v)$ and $h_{j-1}=e_2(v)$, and let $e_3(v)$ be the edge with third largest value of $w_{v,e}$. Then the RHS increases by w_{v,h_j}, while the LHS increases by $w_{v,h_{j-1}}+w_{v,h_j}-w_{v,e_3(v)}$. Since $w_{v,j_{h-1}}\geq w_{v,e_3(v)}$ the inequality continues to hold.

Proof of Claim 3. We start with an arbitrary vertex $v_1\in V$ and add v_1 to C. In iteration j we add one new vertex v_j to C, until we add a vertex u that already appears in C. Assume that the first appearance of u on C is $u=v_r$. We then remove vertices $v_1,\ldots,v_{r-1}$ from C and reverse the orientation of C to produce the final output (so vertices appear in reverse order to that in which they were added to C).

In the first iteration, $C=\{v_1\}$. Let $e_1(v_1)=(v_1,u)$. We then add u as v_2 to C. In general, in iteration j, consider vertex v_j and the edge $h_{j-1}=(v_{j-1},v_j)$. If $h_{j-1}\neq e_1(v_j)$, and $e_1(v_j)=(v_j,u)$, then we add u as v_{j+1} to C. Otherwise, let $e_2(v_j)=(v_j,u')$. We then add u' as v_{j+1} to C.

Let v_{t+r} be the last vertex we add to the cycle, so $v_{t+r} = v_r$. Consider the cycle $C' = (v_r, v_{r+1}, \ldots, v_{t+r} = v_r)$ (recall that we will reverse the ordering of vertices in C' in the final solution, the current ordering reflects the order in which vertices have been added to C). We denote $g_j = (v_j, v_{j+1})$. Consider now some vertex $v_j \in C$. Assume first that $j = r$. Then two cases are possible. If $g_r = e_1(v_r)$, then clearly $w_{v_r,g_r} \geq w_{v_r,g_{r+t-1}}$ and condition (1) will hold in the reversed cycle. Assume now that $g_r = e_2(v_r)$. Then the edge $e' = (v_{r-1}, v_r)$ that originally belonged to C is $e_1(r)$, and so $w_{v_r,g_r} \geq w_{v_r,g_{r+t-1}}$ still holds.

Assume now that $j \neq r$. If $g_j = e_1(v_j)$ then clearly $w_{v_j,g_j} \geq w_{v_j,g_{j-1}}$ and condition (1) holds (in the reversed cycle). Otherwise it must be the case that $g_{j-1} = e_1(v_j)$ and $g_j = e_2(v_j)$ and so condition (2) holds. □

□

References

[1] A. Asadpour, U. Feige, and A. Saberi. Santa Claus meets hypergraph matchings. *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 10–20, 2008.

[2] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. *Proceedings of ACM Symposium on the Theory of Computation (STOC)*, pages 114–121, 2007.

[3] N. Bansal and M. Sviridenko. The Santa Claus Problem. *Proceedings of ACM Symposium on the Theory of Computation (STOC)*, pages 31–40, 2006.

[4] M. H. Bateni, M. Charikar, and V. Guruswami. MaxMin Allocation via Degree Lower-Bounded Arborescences. *To Appear in Proceedings of ACM Symposium on the Theory of Computation (STOC)*, May 2009

[5] M. H. Bateni, M. Charikar, and V. Guruswami. New approximation algorithms for degree lower-bounded arborescences and max-min allocation. Technical Report TR-848-09, Computer Science Department, Princeton University, March 2009.

[6] I. Bezakova and V. Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005.

[7] S. J. Brams and A. D. Taylor. *Fair Division : From Cake-Cutting to Dispute Resolution.* Cambridge University Press, 1996.

[8] R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems *ACM-SIAM Annual Symposium on Discrete Algorithms (SODA)*, pages 106–115, 2000.

[9] M. Conforti, R. Hassin, and R. Ravi. Reconstructing edge-disjoint paths *Oper. Res. Letters.*, 31:273–276, 2003.

[10] T. Ebenlendr, M. Krcal, and J. Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. *ACM-SIAM Annual Symposium on Discrete Algorithms (SODA)*, pages 483–490, 2008.

[11] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57, 2004.

[12] U. Feige. On allocations that maximize fairness. *ACM-SIAM Annual Symposium on Discrete Algorithms (SODA)*, pages 287–293, 2008.

[13] M. X. Goemans, N. J. A. Harvey, S. Iwata, and V. Mirokkni. Approximating submodular functions everywhere. *Proceedings of ACM-SIAM Annual Symposium on Discrete Algorithms (SODA)*, pp. 535–544, 2009.

[14] S. Khot and A. K. Ponnuswami. Approximation Algorithms for the Max-Min Allocation Problem. *Proceedings of APPROX-RANDOM*, pp. 204-217, 2007.

[15] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming*, 46:259–271, 1990.

[16] C.H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theo. Comp. Sci.* 84:127–150, 1991

[17] G. J. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20:149–154, 1997.

[18] G. J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12:57–75, 2000.

A The Integrality Gap of the LP

In this section we show a lower bound of $\Omega(\sqrt{m})$ on the integrality gap of the LP from Section 3.4.2. We then show how the algorithm described in Section 4 overcomes this gap. The construction of the gap example is somewhat similar to the construction used by [3] to show a lower bound of $\Omega(\sqrt{n})$ on the integrality gap of the configuration LP.

We describe a canonical instance together with an assignment of private items. We start by describing a gadget G that is later used in our construction. Gadget G consists of M light agents $L_1, \ldots, L_M$. For each light agent L_j, there is a distinct collection $S(L_j)$ of M light items for which L_j has utility 1. Let $\mathbf{S} = \cup_j S(L_j)$, note that $|\mathbf{S}| = M^2$. Items in $\mathbf{S}$ will not be assigned as private items to any agent.

Additionally, for each $j : 1 \leq j \leq M$, agent L_j has one heavy item $h(L_j)$, for which L_j has utility M. This will be L_j's private item. The gadget also contains $M-1$ heavy agents $t_1, \ldots, t_{M-1}$. These heavy agents are not assigned private items and hence are terminals. Each terminal is a heavy agent that has utility M for each one of the items $h(L_1), \ldots, h(L_M)$. Finally, we have a light agent L^* that has utility 1 for each item $h(L_1), \ldots, h(L_M)$. The dotted circle in Figure 1 shows the gadget for $M = 3$.

We make M copies of the gadget, $G_1, \ldots, G_M$. We denote the vertex L^* in gadget G_j by L^*_j. We add a distinct heavy item $h(L^*_j)$ for each L^*_j. Item $h(L^*_j)$ is the private item for L^*_j, and gives utility M to it. Finally, we have a heavy agent t^* that has utility M for each $h(L^*_j)$, $1 \leq j \leq M$. This agent is also a terminal since it has no private item assigned. Our set of terminals thus consists of all the heavy agents in the instance. The total number of items is $n = O(M^3)$ and the total number of agents is $m = O(M^2)$. Figure 1 shows the flow network $N(\mathcal{I}, \pi)$ for the case $M = 3$.

We start by showing that in any integral solution, some agent receives a utility of at most 1. That is, any integral flow from $\mathbf{S}$ to the terminals will $1/M$-satisfy some light agent. This is because the terminal t^* must receive one unit of flow from L^*_j for some $1 \leq j \leq M$. Consider now the corresponding gadget G_j. We can assume w.l.o.g. that each light agent $L_i \in G_j$, $1 \leq i \leq M$ receives M flow units from its light items in $S(L_i)$. Each one of the $M-1$ terminals $t_1, \ldots, t_{M-1}$ has to receive one flow unit. This leaves only one flow unit to satisfy L^*_j, and so L^*_j is assigned at most one item for which it has utility 1.

We now show that the LP is feasible for the guess of M; this will show the $\Omega(\sqrt{m})$ gap. Consider some gadget G_i. Each light agent $L_j \in G_i$ receives M flow units from light agents in $S(L_j)$, and sends 1 flow unit to its private item $h(L_j)$, which in turn sends $1/M$ flow units to each one of the agents $t_1, \ldots, t_{M-1}, L^*_i$. Each one of the light agents $L^*_1, \ldots, L^*_M$ now receives 1 flow unit can thus

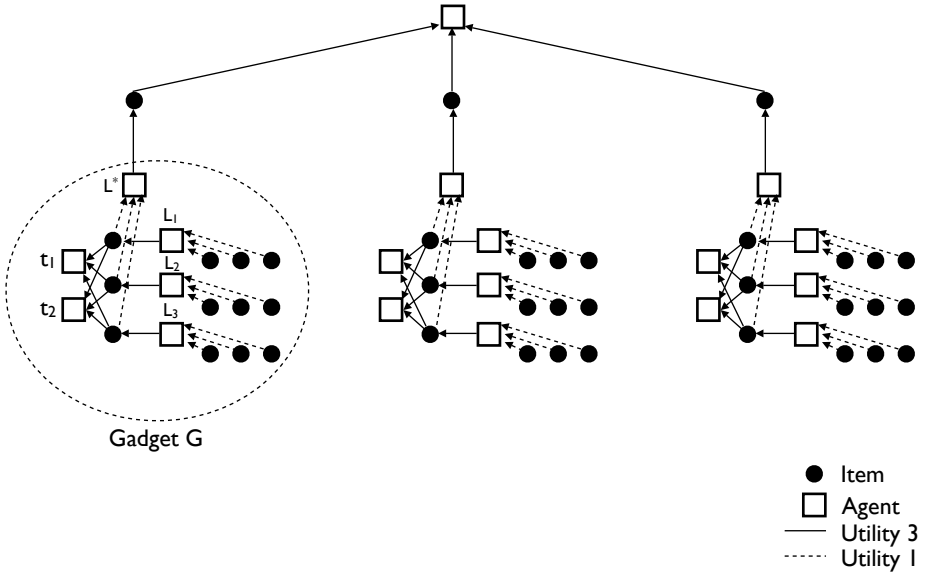

Figure 1: The flow network $N(\mathcal{I}, \pi)$ for the gap instance with $M = 3$.

send $1/M$ flow to terminal t^*.

The terminal t^* is sent a flow of value 1 by light agents in the last layer of the graph G_2 (the graph supposed to capture height 2 trees) while the terminals t_i^j (the ith terminal of the jth gadget) are sent flow by light agents in the last layer of G_1.

Let us consider the terminal t^* first. The tuple variables are as follows. We have $y(L_j^*) = x(L_j^*) = 1/M$ for all $j = 1, \ldots, M$ and for all $1 \leq j \leq M$, for all $1 \leq i \leq M$, we have $y(L_j^*, L_i^j) = 1/M$, where L_i^j is the ith light agent in G_j. The flow to the terminal t^* is via the paths $(L_i^* \to h(L_i^*) \to t^*)$ for all i, and each flow is of value $1/M$. The flow of $1/M$ for the tuple (L_i^*, L_i^j) is along the path $(L_i^j \to h(L_i^j) \to L_i^*)$ and is of value $1/M$. This takes care of terminal t^*.

Let us now consider the terminal t_i^j. The tuple variables for this are $x(L_{i'}^j) = y(L_{i'}^j) = 1 - 1/M$ for all $i' = 1, ..., M$. The flow to terminal t_i^j is a flow of $1/M$ via each of the paths of form $(L_{i'}^j \to h(L_{i'}^j) \to t_i^j)$, for all $i' = 1, ..., M$. Note that the total flow through a light agent L_i^j is $(M-1)/M = x(L_i^j)$. This completes the description of the gap example.

Before describing how our algorithm bypasses the integrality gap, we first show how in this example we can prove using the same LP that the integral optimum cannot be more than 1. Firstly, note that removing any agent cannot *decrease* the integral optimum. Therefore, if we remove the set of light agents $\{L_1, \ldots, L_{M-1}\}$ from every G_j, the integral optimum should still be at least M. However, consider now the assignment π' of private items defined as follows. For the remaining light agents we still have $\pi'(L_j^*) = h(L_j^*)$ and $\pi'(L_M^j) = h(L_M^j)$ for $1 \leq j \leq M$, but now we assign a private item for every heavy agent t_i in each gadget, $\pi'(t_i) = h(L_i)$, that is, the heavy item of agent L_i (who is not present in this instance). The only terminal in this instance is the heavy agent t^*. The resulting flow network $N(\mathcal{I}', \pi')$ for $M = 3$ is shown in Figure 2.

Note that the set $\mathbf{S}$ of items which are not private items in each gadget G_j is still $\bigcup_j S(L_j)$. However,

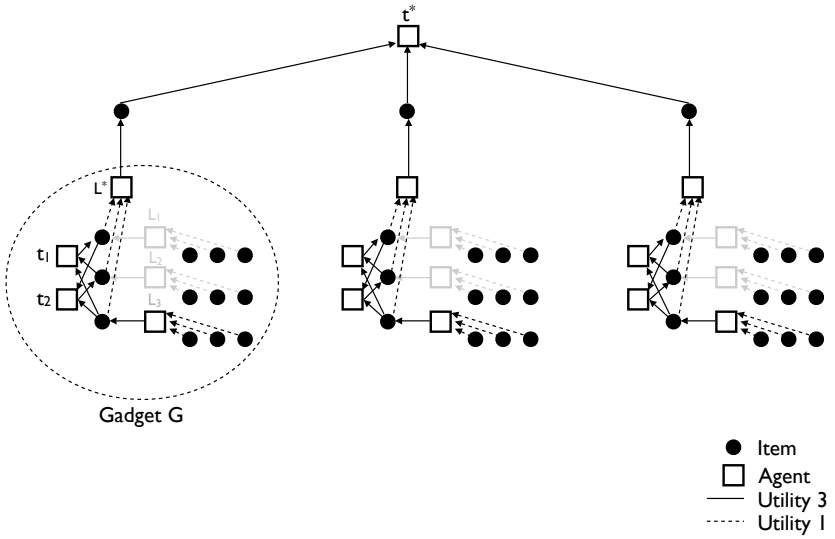

Figure 2: The flow network $N(\mathcal{I}', \pi')$. Note that the edges between t_i and $h(L_i)$ have flipped for all $1 \leq i \leq (M-1)$.

the items in $S(L_j)$ for $1 \leq j \leq (M-1)$ do not connect to any agent (since the light agents have been removed). Thus the flow to the terminal t^* must come from the M sets of items of the form $S(L_M^j)$.

We now argue that the LP of Section 3.4.2 is not feasible. In fact, even if the values $N_{L_j^*}$ are reduced from M to 2 for every agent L_j^*, for $1 \leq j \leq M$, the LP is still not feasible. Since t^* receives a flow of value 1, it must receive a flow of at least $1/M$ from one of the agents L_j^*, for $1 \leq j \leq M$. Thus for this j, $x(L_j^*) \geq 1/M$. This implies L_j^* must receive $N_{L_j^*} \cdot x(L_j^*)$ units of flow from the light agents in the lower level. However, there is only one light agent, namely L_M^j, in the lower level and from constraint (15) it can "feed" at most $x(L_j^*)$ units of flow to L_j^*. Thus, if $N_{L_j^*} > 1$, the LP will be infeasible.

Now we show how after one iteration of the algorithm we get to the instance $(\mathcal{I}', \pi')$ described above. After the rounding algorithm described in Section 3, we get set of paths $\mathcal{P}_1, \mathcal{P}_2$ which are as follows:

$$\mathcal{P}_1 = (L_1^* \to h(L_1^*) \to t^*) \cup \{(L_i^j \to h(L_i^j) \to t_i^j) : \forall i = 1 \ldots M-1, 1 \leq j \leq M\}$$
$$\mathcal{P}_2 = \{(v \to L_i^j) : v \in S(L_i^j), 1 \leq i \leq M-1, 1 \leq j \leq M\} \cup \{(v \to L_M^1) : v \in S(L_M^1)\}$$
$$\{(L_i^1 \to h(L_i^1) \to L_1^*) : 1 \leq i \leq M\}$$

that is, $\mathcal{P}_1$ is the set of paths from L_1^* to t^* and the paths from L_i^j to t_i^j in every gadget G_j; and $\mathcal{P}_2$ is the set of paths from $S(L_i^j)$ to L_i^j for $i = 1$ to $M-1$ for all gadgets j, except for the the gadget G_1, we have the paths from $S(L_M^1)$ to L_M^1 as well, and the path from L_i^1 to L_1^* for the gadget G_1.

The re-routing of paths (Step 2 of the algorithm described in Section 4) leads to the following set of

paths.

$$\mathcal{P}' = (L_1^* \to h(L_1^*) \to t^*) \cup \{(L_i^j \to h(L_i^j) \to t_i^j) : \forall i = 1 \dots M-1, 1 \le j \le M\}$$
$$\mathcal{Q}' = \{(v \to L_i^j) : v \in S(L_i^j), 1 \le i \le M-1, 1 \le j \le M\} \cup \{(v \to L_M^1) : v \in S(L_M^1)\}$$
$$\cup \{L_M^1 \to h(L_M^1) \to L_1^*\}$$

Thus there is only one bad agent, L_1^*, which has one path leaving it in $\mathcal{P}'$ and only one path (instead of M) entering it in $\mathcal{Q}'$. The input to iteration 2 (Step 3 of the algorithm) is then as follows. The new private assignment π^2 is the same as π', the light agent set $\mathbf{L}^2$ is the set $\{L_i^j : 1 \le i \le M-1,\ 1 \le j \le M\}$, the set of paths $\mathcal{Q}^2$ is the first set of $\mathcal{Q}'$ defined above, and the new set of terminal $\mathbf{T}^2$ is $\{t^*\}$. Note that the instance $\mathcal{I}^2$ is the same as $\mathcal{I}'$, and as we argued before, the LP returns infeasible for this instance and private assignments, and we conclude that the original instance was infeasible as well.

B Hardness of Approximation for Uniform Graph Balancing

Given a (nonuniform/uniform) weighted graph, the (nonuniform/uniform) graph balancing problem is to find the orientation that maximizes the minimum weighted (nonuniform/uniform) in-degree. We show that the uniform graph balancing is NP-hard to approximate up to a factor $2-\delta$ for any $\delta > 0$. This result implies the same hardness of approximation for the Santa Claus version of 2-restricted Max-Min Allocation, since the two problems are equivalent.

Theorem 8 *Uniform graph balancing is NP-hard to approximate to within a factor $2-\delta$, for any $\delta > 0$.*

Proof: This proof is similar to an NP-hardness of the min-max version of graph balancing due to Ebenlendr et.al. [10]. A similar proof has been independently shown by [4]. We reduce from the following variant of 3-SAT. The input is a 3CNF formula φ, where each clause has 3 variables, and each **literal** appears in at most 2 clauses and at least 1 clause. This version is NP-hard [16]

We construct a graph $G = (V, E)$ whose vertices correspond to the literals and clauses in the formula φ. We define edges of G and the weights associated with them as follows. For every variable x we have two vertices $x, \overline{x}$ representing the two corresponding literals, with an edge $(x, \overline{x})$ of weight 1. We refer to edges of this type as *variable edges*. Consider now some clause $C = (\ell_1 \vee \ell_2 \vee \ell_3)$. We have a vertex C and three *clause* edges (C, ℓ_1), (C, ℓ_2), (C, ℓ_3) associated with it. All clause edges have weight $\frac{1}{2}$. Additionally, we have a self-loop of weight $\frac{1}{2}$ for each clause C, and for each literal ℓ appearing in exactly one clause.

YES case: Assume that the formula is satisfiable. We show that the optimum value of the graph balancing instance is 1. Consider any variable x. If the optimal assignment gives value T for x, then we orient the corresponding variable edge towards x; otherwise it is oriented towards $\overline{x}$. For each literal ℓ set to F, orient all its adjacent clause edges towards it. Together with the self-loop, there are 2 edges oriented towards ℓ, each of which has weight $\frac{1}{2}$. Finally, consider clause C. Since this clause is satisfied by the assignment, at least one of its variable has value T, and we can orient the corresponding edge towards C. Together with the self-loop, C has 2 edges oriented towards it of weight $\frac{1}{2}$ each.

NO case: Suppose there is an allocation such that every vertex has weighted in-degree strictly more than 1/2. This implies that every vertex has weighted in-degree at least 1. Consider the orientation of the variable edges. This orientation defines an assignment to the variables: if $(x, \overline{x})$ is oriented towards x, the assignment is T, otherwise it is F. Consider a literal ℓ that does not have the variable edge oriented towards it. Then it must have the 2 remaining edges incident on it oriented towards it (since they have weight 1/2 each). Now consider a clause vertex C. Since its weighted in-degree is at least 1, it must have a clause edge oriented towards it. The corresponding literal then is assigned the value T and therefore C is satisfied. But we know at least one clause is not satisfied by the above assignment. This proves the theorem. □