On Allocating Goods to Maximize Fairness

Deeparnab Chakrabarty* Julia Chuzhoy† Sanjeev Khanna‡

January 1, 2009

Abstract

Given a set $\mathcal{A}$ of m agents and a set I of n items, where agent $A \in \mathcal{A}$ has utility $u_{A,i}$ for item $i \in I$, our goal is to allocate items to agents to maximize fairness. Specifically, the utility of an agent is the sum of the utilities for 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 approximation algorithm that achieves an $\tilde{O}(n^{\epsilon})$ approximation for any $\epsilon = \Omega(\log\log n/\log n)$ 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 $\tilde{O}(n^{\epsilon})$-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 a non-zero utility for at most two agents. We show that even in this restricted setting the problem is hard to approximate upto any factor better than 2, and show a factor $(2+\epsilon)$-approximation algorithm running in time $\text{poly}(n, 1/\epsilon)$ for any $\epsilon > 0$. This special case can be cast as a graph edge orientation problem, and our algorithm can be viewed as a generalization of Eulerian orientations to weighted graphs.

1 Introduction

In this paper we consider the problem of allocating indivisible goods to a set of agents with the objective to maximize the minimum utility among all agents. In particular, we are given a set $\mathcal{A}$ of m agents, a set 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 $S \subseteq I$ of items is $u_A(S) := \sum_{i \in S} u_{A,i}$, that is, the utility

*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.

‡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.

function is additive. An allocation of items is a function $\Phi : \mathcal{A} \to 2^I$ such that an item is allocated to at most one agent, that is, $\Phi(A) \cap \Phi(A') = \varnothing$ whenever $A \neq A'$. The MAX-MIN ALLOCATION problem is to find an allocation Φ of items which maximizes $\min_{A \in \mathcal{A}} \{u_A(\Phi(A))\}$.

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. This is in contrast to allocation problems where the goal is to maximize the *total* utility of agents, a problem that admits a trivial solution for any additive utility function: allocate each item to a person that derives the highest utility from it. The MAX-MIN ALLOCATION problem may be viewed as a 'dual' problem to the well studied min-max (also known as makespan) scheduling problem where the goal is to find an allocation minimizing the utility (load) of the maximum utility agent (machine).

The MAX-MIN ALLOCATION problem was indeed first studied as a machine scheduling problem where the minimum completion time has to be maximized. Woeginger [16] and Epstein and Sgall [9] gave polynomial time approximation schemes (PTAS) for the cases when all agents (machines) have identical utilities for the items. Woeginger [17] also gave an FPTAS for the case when 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 [5] who gave a $(n - m + 1)$-approximation algorithm. They also showed the problem is NP-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 property that it has the same utility for a subset of agents and 0 for the rest. In other words, for each agent A and item i, either $u_{A,i} = u_i$, depending only on the item i, or $u_{A,i} = 0$. 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 [10] showed a constant upper bound on the integrality gap of configuration LP for the Santa Claus problem. However his proof is non-constructive and does not translate into an approximation algorithm. Subsequently, Asadpour, Feige and Saberi [1] provided an alternative non-constructive proof of a factor-5 upper bound on the integrality gap of the configuration LP.

As for the general MAX-MIN ALLOCATION problem, Bansal and Sviridenko [3] showed that the configuration LP has an integrality gap of $\Omega(\sqrt{n})$ in this setting. Recently, Asadpour and Saberi [2] gave an $O(\sqrt{m} \log^3 m)$ approximation 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 [5]. 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.

Additionally, we investigate a special case of MAX-MIN ALLOCATION when 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 the instance a *uniform* 2-restricted instance. The (uniform) 2-restricted MAX-MIN ALLOCATION reduces to an *orientation* problem in (uniformly) non-uniformly weighted graphs where one has to orient the edges so as to maximize the minimum weighted in-degree of a vertex (a non-uniformly weighted graph has two

weights per edge - one for each end point). This orientation problem is called the *graph balancing* problem and is motivated by the min-max analogue studied recently by Ebenlendr et.al. [8]. 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 show that even the uniform 2-restricted Max-Min Allocation is hard to approximate up to any factor better than 2. Moreover, we give a polynomial time $(2+\epsilon)$-approximation algorithm for the non-uniform 2-restriced Max-Min Allocation, for any $\epsilon > 0$. In fact, we show that the integrality gap of the configuration LP is exactly 2 in this special case – the extra ϵ in the approximation factor comes from the fact that the configuration LP can only be solved approximately.

Remark: We have recently learned that independently of our work, Bateni, Charikar, and Guruswami [4] showed an approximation algorithm for a special case of the Max-Min Allocation problem, where the configuration LP is known to have a large integrality gap. Their algorithm achieves an $O(m^{\epsilon})$-approximation in $m^{O(1/\epsilon)}$ time and a polylogarithmic approximation in quasi-polynomial time for the special case. They also obtain a factor 4 approximation for the 2-restricted Max-Min Allocation.

1.1 Overview of Results and Techniques

Our main result is as follows:

Theorem 1 *For any $\epsilon \geq 8 \log\log n / \log n$, there is an $n^{O(1/\epsilon)}$-time algorithm to compute an $\tilde{O}(n^{\epsilon})$-approximate solution for the* Max-Min Allocation *problem.*

We now give an overview of the proof of Theorem 1 and highlight the main ideas involved. We begin by guessing the value M of the optimal solution. This can be done since, as we show later, we can assume that all utilities are polynomially bounded, losing a constant factor in the approximation ratio. We then show that we can convert an instance of the general Max-Min Allocation to what we call *canonical instances*, while losing an $O(\log n)$ factor in the approximation ratio. In a canonical instance, there are only two types of agents – heavy agents whose utilities for items are 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 is a large integer (larger than the desired approximation factor). The items with a utility of M/t for a light agent are referred to as the *light items*.

Our second idea is that of transforming the problem of assigning items to agents into a network flow problem by the means of *private items*. Each agent is assigned at most one distinct private item, for which it has utility M. Note that necessarily the private item of a light agent will be its heavy item. Nevertheless there could be many ways of assigning private items to heavy agents. Of course if we find an assignment of private items for every agent we are trivially done, since this assignment induces a near-optimal solution. Thus the interesting case is when for a *maximal* assignment of private items we have some agents who are not assigned any private items. Such agents will be called *terminals*. Suppose for the time being that all light agents are assigned private items. The key observation which we use is that if the optimum value is M, then, given any

assignment of private items, there must exist 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, and then these private items can 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 in-flow of t). We then design a linear programming relaxation to obtain a fractional flow solution in 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 $\Omega(\sqrt{m})$ – that is, there is an instance and allocation of private items such that in the resulting network a flow of M can be sent to the terminals fractionally, while an integral flow is not possible even if the "constraints" on the flow are "relaxed" by a factor of $O(\sqrt{m})$. Thus, rounding the LP-solution directly cannot give a better than $O(\sqrt{m})$ approximation factor.

This brings us to another key technical contribution. Although the LP has a large 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*, that is, 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 much smaller number of terminals than the stating instance. We thus show that in poly$(1/\epsilon)$ phases, either one of these instances will certify infeasibility of the integral optimum, or we will get an allocation of items which is an $\tilde{O}(n^{\epsilon})$-approximation. This ends the high-level idea of Theorem 1.

We note that Theorem 1 can also be used to obtain an approximation in terms of number of agents. In particular, we show that we can obtain for any fixed $\epsilon > 0$, a quasi-polynomial time m^{ϵ}-approximation algorithm.

Our second result in the paper is about 2-restricted Max-Min Allocation instances. Recall that a 2-restricted instance is one in which every item $i \in I$ has positive utility for at most two agents A_i and B_i. A 2-restricted Max-Min Allocation instance is called *uniform* if for every item i, $u_{A_i,i} = u_{B_i,i}$. We prove the following theorem which pins down the approximability of 2-restricted instances.

Theorem 2 *For any $\epsilon > 0$, there exists a $(2+\epsilon)$-approximation algorithm to the non-uniform 2-restricted* Max-Min Allocation *problem which runs in time* poly$(n, 1/\epsilon)$. *Furthermore, for any $\delta > 0$, it is NP-hard to obtain a $(2-\delta)$-approximation algorithm even for the uniform 2-restricted* Max-Min Allocation *problem.*

In fact, we show that the integrality gap of the configuration LP of [3] for 2-restricted Max-Min Allocation is bounded from above by 2 (recall that for general Max-Min Allocation, the integrality gap is at least $\Omega\sqrt{n}$). As mentioned above, the 2-restricted Max-Min Allocation can be cast as an orientation problem on (non-uniformly) weighted graphs. The main technical lemma which we use for proving the our result above 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 of 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 solve the configuration LP and this gives for each item i a ratio x_{A_i} and $x_{B_i} = 1 - x_{A_i}$ in which it is allocated to the two agents wanting it. Call an item fractionally divided if both x_{A_i} and x_{B_i} are strictly positive. We show that for any agent the total utility of the set of items it is allocated fractionally equals OPT plus the maximum utility element in that set. Using the lemma described in the above paragraph, we can get an allocation of value at least $OPT/2$ for every agent.

1.2 Related Work

The Max-Min Allocation problem falls in the broad class of resource allocation problems – allocating limited resources subject to constraints – 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* have been extensively studied by economists and political scientists with entire books (for example, [6]) written on the subject. However, the *indivisible* case has not received as much attention. There can be many notions of fairness and apart from the notion we study, another measure of fairness studied from an algorithmic point of view has been that of *envy-freeness*. An allocation is *envy-free* if every agent prefers the set of items allocated to it to a set allocated to any other agent. Lipton et.al. [14] studied the notion of *envy* of an allocation and gave polynomial time algorithms to find an allocation with an absolute bound on the envy. We should remark here that the two notions of fairness are not related – an envy free allocation could be far from being a max-min allocation, and vice-versa.

The complexity of resource allocation problems also depends on the complexity of the utility functions of agents. As we mention above, 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 $u_A(S) := \sum_{i \in S} u_{A,i}$. However, such an assumption on utilities is too restrictive and more general utility functions have been studied in the literature. Two such utilities are *submodular utilities* – 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* – $u_A(S) + u_A(T) \geq u_A(S \cup T)$. Note that submodular utilities are a special case of the sub-additive utilities. Khot and Ponnuswami [12] gave a $(2m-1)$-approximate algorithm for Max-Min Allocation with sub-additive utilities. Recently, Goemans et.al. [11] using the Asadpour-Saberi [2] $\tilde{O}(\sqrt{m})$-algorithm 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 [11] 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.

As we have already mentioned, Max-Min Allocation may be viewed as a dual problem to the minimum makespan machine scheduling. Lenstra, Shmoys and Tardos [13] 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. [8] studied a very restricted setting where each job can only go to two machines and moreover takes the same time on both. For this

case they gave a 7/4 approximation algorithm and also showed 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 [8] and our hardness result is similar. However, the ideas behind our approximation algorithm are quite different.

2 Preliminaries

Our goal is to produce an $\tilde{O}(n^\epsilon)$-approximate solution in time $n^{O(1/\epsilon)}$. We assume that $n^\epsilon \geq \Omega(\log^8 n)$, and thus $\epsilon \geq \Omega(\log\log n/\log n)$. We also assume that n is larger than any constant and throughout when we say "n large enough" we imply larger than a suitable constant. We use M to denote the (guessed) value of the optimal solution. If $M \leq \mathsf{OPT}$ then our algorithm produces an $\tilde{O}(n^\epsilon)$-approximation, otherwise it returns a certificate that $M > \mathsf{OPT}$. For an agent A and 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 an item i is *wanted* by agent A iff $u_{A,i} > 0$.

2.1 Polynomially Bounded Utilities

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 w.l.o.g. 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 S of items assigned to A by OPT. The total utility of 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 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. Let $M' \leq 2n$ be the maximum utility in the transformed instance. From here on, we assume that our starting instance is the transformed instance with polynomially bounded utilities, and will denote M' by M and the new utilities $u'_{A,i}$ by $u_{A,i}$.

2.2 Canonical Instances

It will be convenient to work with a structured class of instances that we refer to as *canonical instances*. The notion of a canonical instance depends on the approximation ratio that we desire. Given any $\epsilon \geq \Omega(\log\log n/\log n)$, we say an instance $\mathcal{I}$ of Max-Min Allocation is ϵ-*canonical*, or simply, *canonical* iff:

- All agents can be partitioned into two sets, namely, a set L of light agents and a set H of heavy agents.

- Each heavy agent $A \in H$ is associated with a subset $\Gamma(A)$ of items such that each item in $\Gamma(A)$ has a utility of M for A.
- Each light agent $A \in L$ is associated with
  - a *distinct* item $h(A)$ that has utility M for A and is referred to as the *heavy item* for A. Note that if $A \neq A'$ then $h(A) \neq h(A')$,
  - a parameter $N_A \geq n^\epsilon$, and
  - a set $S(A)$ of items such that each item in $S(A)$ has a utility of M/N_A for A.

Given an assignment of items to agents in the canonical solution, we say that a heavy agent A is *satisfied* iff it is assigned one of the items in $\Gamma(A)$, 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 $S(A)$. 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. Given a canonical instance, our goal is to find an assignment of items to agents that 1-satisfies all agents.

Lemma 1 *Given 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, and 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 X be the set of items in $\mathcal{I}$. The set X' of items for the instance $\mathcal{I}'$ will contain items of X as well as $m(2s)$ 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 $\chi_0(B)$ and s light agents $\lambda_1(B), \ldots, \lambda_s(B)$ where each light agent $\lambda_j(B)$ is associated with value $N_{\lambda_j(B)} = M/(s \cdot 2^j) \geq M/(s \cdot 2^s) \geq n^\epsilon$.
- For each $j \in \{1, \ldots, s\}$, if the utility of item $i \in X$ for B is $2^{j-1} < u_{B,i} \leq 2^j$, then agent $\lambda_j(B)$ has utility $s \cdot 2^j = M/N_{\lambda_j(B)}$ for i. If $i \in X$ is an item for which $u_{B,i} > 2^s$, then $\chi_0(B)$ has utility M for item i.
- Additionally, for each light agent $\lambda_j(B)$ there is a heavy item $h(\lambda_j(B))$ and a heavy agent $\chi_j(B)$. Item $h(\lambda_j(B))$ has utility M for both $\lambda_j(B)$ and $\chi_j(B)$.
- 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 a utility of M for each of the agents in $\{\chi_0(B), \chi_1(B), \ldots, \chi_s(B)\}$, the set of heavy agents for B.

This completes the definition of the canonical instance $\mathcal{I}'$. Consider an optimal solution to $\mathcal{I}$. We show an assignment that 1-satisfies all agents. Consider some agent B in the transformed instance, and let $T(B)$ be the set of items assigned to B in the solution to $\mathcal{I}$. We can partition

$T(B)$ into $(s+1)$ sets as follows. The set $T_0 \subseteq T(B)$ contains all items i with $u_{B,i} > 2^s$. For each $j \in \{1, \ldots, s\}$, the set T_j contains all items i with $2^{j-1} < u_{B,i} \leq 2^j$. Assume first that $T_0 \neq \emptyset$, and let i be any item in T_0. Then we assign item i to heavy agent $\chi_0(B)$. The remaining s heavy agents corresponding to B now get assigned one item from Y_B each. The light agents $\lambda_j(B)$ are assigned their heavy items $h(\lambda_j(B))$. All agents corresponding to B are now 1-satisfied. Assume now that $T_0 = \emptyset$. Then there is $j \in \{1, \ldots, s\}$, such that the utility of items in T_j is at least M/s for B, so $|T_j| \geq M/(s \cdot 2^j) = N_{\lambda_j(B)}$. We assign all items in T_j to $\lambda_j(B)$, and $h(\lambda_j(B))$ is assigned to the heavy agent $\chi_j(B)$. Now the remaining s heavy agents are assigned one item of Y_B each. For each one of the remaining light agents, we assign $h(\lambda_{j'}(B))$ to $\lambda_{j'}(B)$. Therefore, the canonical instance has a solution that 1-satisfies all agents.

Conversely, consider now any α-approximate solution for the canonical instance $\mathcal{I}'$. Let B be some agent in the original instance. Consider the corresponding set of heavy agents $\chi_0(B), \ldots, \chi_s(B)$. 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 $\chi_0(B)$. Then it must be assigned some item $i \in X$ 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 $\chi_j(B)$, $j \neq 0$ that is not assigned an item from Y_B. Then $\chi_j(B)$ is assigned item $h(\lambda_j(B))$, and so $\lambda_j(B)$ must be assigned a set S' of at least $N_{\lambda_j(B)}/\alpha = M/(s \cdot 2^j \cdot \alpha)$ 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 $s \leq \log n)$, we obtain a $\max\{O(n^\epsilon \log n), O(\alpha \log n)\}$-approximate solution. □

From now on we focus on finding an approximate solution to the canonical instance. We assume that the optimal solution can 1-satisfy all agents. From the above claim, a solution that α-approximates all agents in the canonical instance will imply a $\max\{O(n^\epsilon \log n), O(\alpha \log n)\}$-approximate solution to the original instance.

2.3 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 an assignment P of private items to agents. Such an assignment is called *good* iff it satisfies the following properties:

- For every light agent $A \in L$, its private item is $P(A) = h(A)$ (that is, the distinct item of utility M associated with the light agent A).
- An item can be a private item for at most one agent. The set of items that do not serve as private items is denoted by S.
- The set of heavy agents that have private items are denoted by H'. The remaining heavy agents are called *terminals* and are denoted by T. Item i can be a private item of heavy agent $A \in H$ only if $i \in \Gamma(A)$ (recall that $\Gamma(A)$ is the set of items for which A has utility M).

The initial assignment P of private items to heavy agents is obtained as follows. We create a bipartite graph $G = (U, V, E)$, where $U = H$, 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 $i \in \Gamma(A)$. 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 $A \in T$, $P(A)$ is undefined and $\{P(A)\} \triangleq \emptyset$.

The Flow Network Given a canonical instance $\mathcal{I}$ and an assignment $P : L \cup H' \to I$ of private items, we define the corresponding **directed** flow network $N(\mathcal{I}, P)$ as follows. The set of vertices is $\mathcal{A} \cup I \cup \{s\}$. Source s connects to every vertex i where $i \in S$. If agent $A \in \mathcal{A}$ has a private item and $i = P(A)$, then vertex A connects to vertex i. If A is a heavy agent and $i \in \Gamma(A) \setminus \{P(A)\}$, then vertex i connects to vertex A. If A is a light agent and $i \in S(A)$ then vertex i connects to vertex A. Let $N(\mathcal{I}, P)$ denote the resulting network. A feasible integral flow in this is a flow obeying the following constraints.

C1. All flow originates at the source s.

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

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

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

C5. For each light agent $A \in L$, if the incoming flow is N_A then the outgoing flow is 1; otherwise both are 0.

An *integral flow* obeying the above conditions is called a *feasible flow.*

Lemma 2 *An optimal integral solution to the canonical instance $\mathcal{I}$ gives a feasible flow in $N(\mathcal{I}, P)$.*

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

We say that a flow is α-*feasible* iff constraints (C1)–(C4) hold for it, and Constraint (C5) is replaced by the following relaxed constraint:

C6. For each light agent $A \in L$, if the incoming flow is at least N_A/α then the outgoing flow is 1; otherwise both are 0.

Lemma 3 *An integral α-feasible flow in $N(\mathcal{I}, P)$ gives an α-approximate solution for the canonical instance $\mathcal{I}$.*

Proof: Consider an integral α-feasible flow. Consider any agent A that may be heavy or light. We simply assign it every item that sends flow to it if there is such an item. If there is no such item, we assign $P(A)$ 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 I^* be the set of items and H^* the set of heavy agents reachable from s by paths that do not contain light agents. A useful property of our initial assignment of private items is that H^* does not contain any terminals (otherwise we could have increased the matching). Throughout the algorithm, the assignment of private items to H^* does not change, and the above property is preserved. Given any pair v, v' of vertices and any path p that starts at v and ends at v', we say that p connects v to v' *directly* it does not contain any intermediate vertices representing light agents (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 flow is being sent via path p that does not contain light agents as intermediate vertices.

An Equivalent Formulation: A flow-path p is called a *simple path* iff it does not contain any light agents as intermediate vertices, and either a) starts and ends at light agents, or b) starts at a light agent and ends at a terminal, or c) starts at the source s and ends at a light agent. The problem of finding an integral flow satisfying the properties C1–C4 and C6, is *equivalent* to finding a collection of simple paths $\mathcal{P}$ 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 $A \in L$, 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.

2.4 Structured Near-Optimal Solutions for Canonical Instances

Given a canonical instance $\mathcal{I}$ and an assignment P of private items, an optimal solution OPT to $\mathcal{I}$ defines a feasible integral flow in $N(\mathcal{I}, P)$. Consider graph G obtained from $N(\mathcal{I}, P)$ as follows: we remove the source s and all its adjacent edges. We also remove all edges that do not carry flow in OPT. Graph G is then a collection of disjoint trees. Some of the trees in this collection are simply isolated vertices that correspond to agents and items with no flow passing through them. Such agents are assigned their private items. We ignore these trivial trees in this section and focus only on trees with 2 or more vertices.

Each (non-trivial) tree τ in the collection has a terminal $t \in T$ at its root. 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 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 and is bounded by h; we denote this number by $h(\tau)$. Note that $h(\tau) \geq 1$ must hold since there are no direct paths between s and the terminals, and thus $1 \leq h(\tau) \leq h$ for all 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 $h = 8/\epsilon$.

Lemma 4 *There is an $(h+1)$-approximate solution $\mathcal{S}$ to a canonical instance instance $\mathcal{I}$, that defines an h-layered forest, for $h = 8/\epsilon$.*

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 corresponding to OPT. We convert it into an h-layered tree in h iterations.

A light agent $A \in L$ 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 S. Let $L_1(\tau)$ be the set of all level-1 light agents of τ. Consider some agent $A \in L_1(\tau)$. For each child v of A in τ, if v is not on a simple path p connecting an item of 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 receives at least $N_A/(h+1)$ flow units directly from level-$(j-1)$ agents. In iteration j, we consider the set $L_j(\tau)$ of level-j agents. Let $A \in L_j(\tau)$ be any such agent, and let v be any child of A in τ. If v lies on a simple 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 $\mathcal{L} = \cup_{j=1}^{h} L_j(\tau)$. Thus we can make every τ h-layered implying a $(h+1)$-approximate h-layered solution.

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 $A \notin \mathcal{L}$. 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 τ, but it does not belong to $L_j(\tau)$ for $1 \leq j \leq h$. That is, it receives less than $N_A/(h+1)$ flow units directly from light agents in $L_j(\tau)$ for $1 \leq j \leq h-1$ and less than $N_A/(h+1)$ flow units directly from 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 L_h. Each one of these agents receives at least $n^{\epsilon}/(h+1)$ flow from agents in $L_{h-1}(\tau)$ 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 $L_j(\tau)$, and in particular it contains $(n^{\epsilon}/(h+1))^h$ agents from $L_1(\tau)$. We now show that by our choice of ϵ, h, $(n^{\epsilon}/(h+1))^h > m$ which would be a contradiction.

Recall that $n^{\epsilon} \geq \log^8 n$ and $1 > \epsilon \geq 8 \log\log n / \log n$. So $8 \leq (h = 8/\epsilon) \leq \log n / \log\log n$. Thus, for n large enough $(h+1) \leq 2h \leq 2\log n/(\log\log n) \leq \log n \leq n^{\epsilon/8}$ giving us $(n^{\epsilon}/(h+1))^h \geq (n^{7\epsilon/8})^h \geq n^7 \geq n > m$. □

From now on we focus on h-layered instances. For simplicity, we scale down all values N_A for $A \in \mathcal{A}$ by the factor of $(h+1)$, so that an optimal h-layered solution can 1-satisfy all agents. Note that this increases the approximation factor of our algorithm by the factor of $(h+1)$.

2.5 The [BS] Tree Decomposition

One of the tools we use in our construction 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 $N(\mathcal{I}, P)$ by a feasible solution. The setup for the theorem is the following. We are given an **undirected** bipartite graph $G = (\mathcal{A}, I, E)$ where $\mathcal{A}$ is a set of agents, I is a set of items and E contains an edge (A, i) iff i has utility M for A. 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:

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

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

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

The theorem of [3] shows that such an assignment can be decomposed into a more convenient structure. This structure is 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 $\mathcal{A}(\tau)$ is the set of agents of τ and $I(\tau)$ is the set of items of τ, then for each agent $A \in \mathcal{A}(\tau)$, it is possible to satisfy all agents in $\mathcal{A}(\tau) \setminus \{A\}$ by items in $I(\tau)$.

Theorem 3 *([3]) There exists a decomposition of $G = (\mathcal{A}, I, E)$ 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 $i \in I(T_j)$ is 2 and $\sum_{A \in \mathcal{A}(T_j)} x_A > 1/2$.*

Corollary 1 *For each tree T_j in the decomposition, for each agent $A \in \mathcal{A}(T_j)$, it is possible to satisfy all agents in $\mathcal{A}(T_j) \setminus \{A\}$ by items in $I(T_j)$.*

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 3) 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.

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. 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 $i \in I(\tau)$ 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 agent with $(A,i) \in E$. We set $y_{A,i} = 1$ and for all $i' \neq i$, $y_{A,i'} = 0$. We then add (A,i) to E^*, removing the edge and both its endpoints from G. Notice that constraints (1)–(3) continue to hold.

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

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

Step 3: Root tree τ at an arbitrary vertex in $I(\tau)$. Let $i \in I(\tau)$ 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 - let them be $A_1, A_2, \ldots, A_r$ 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, 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:

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

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

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

In this section, as a warm-up towards the eventual goal of computing optimal multi-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 I^* (items reachable directly from s) will only be used to send flow to the light agents, and items in $I' = I \setminus I^*$ will only be used to send flow directly to terminals. Similarly, heavy agents in H^* will send flow to light agents, while heavy agents in $H \setminus H^*$ will send flow directly to terminals. Therefore, if there are any edges from items in I' to agents in H^* we can remove them (no edges are possible between agents in $H \setminus H^*$ and items in I^*).

We now proceed with the design of an LP relaxation. For each light agent $A \in L$, we have a variable x_A showing whether or not A is satisfied as a light agent (by light items). We have a flow of type A, we call it f_A, and require that $N_A \cdot x_A$ units of this flow are sent to agent A, while at most x_A flow units of f_A go through any item in I^*. Finally, the total flow through any item of any type 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 I^*. We then have the following constraints:

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

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

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

Additionally, we need to send one flow unit to each terminal. For $A \in L$, $t \in T$, let $\mathcal{P}(A, t)$ be all paths directly connecting A to t that only use items in I'. We now have the standard flow constraints:

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

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

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

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 I'. The second step is randomized rounding which will create logarithmic congestion. In the last step we take care of the congestion.

Step 1: Tree decomposition We consider the graph induced by vertices corresponding to all agents in set $Z = L \cup (H \setminus H^*)$ and the set I' of items. Notice that agents in Z only have utilities M for items in I'. For each light agent $A \in L$ we have a value x_A (extent to which A is satisfied as light item). For all other agents A, let $x_A = 0$. Our fractional flow solution can be interpreted as a fractional assignment of items in I' to agents in Z, such that each agent $A \in Z$ is fractionally assigned $(1 - x_A)$-fraction of items in I', as follows. Let $A \in H \setminus H^*$ be any heavy agent. For each item $i \in I'$, 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 non-terminal agent, let z be the amount of flow sent on edge $(A \to P(A))$. We set $y_{A,P(A)} = 1 - z$. For a light agent A, we set $y_{A,P(A)} = 1 - x_A$. It is easy to see that this assignment satisfies Constraints (1)–(3). Therefore, we can apply Theorem 3 and obtain a collection $T_1, \ldots, T_s$ of trees together with a matching $\mathcal{M}$ of a subset of items $I'' \subseteq I'$ to a subset $Z' \subseteq Z$ of heavy agents, that do not participate in trees $T_1, \ldots, T_s$.

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 $x_A/X \leq 2x_A$. The selected light agent will eventually be satisfied as a light agent. 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 $L^* = \{A_1, \ldots, A_s\}$. We have obtained an assignment of an item from I' to every agent in $Z \setminus L^*$. This assignment in turn defines a collection of simple flow-paths $\mathcal{P}$, where every path $p \in \mathcal{P}$ connects an agent in $A_1, \ldots, A_s$ to a terminal, with exactly one path leaving each agent A_j and exactly one path entering each terminal, as follows: Consider any agent $A \in Z \setminus \{A_1, \ldots, A_s\}$ and assume that A is assigned some item $i \in I'$. If $i \neq P(A)$, then send one flow unit from i to A, and if $A \notin T$, send one flow unit from A to $P(A)$.

We now turn to finding a collection of simple paths to satisfy the agents in L^*. For each $A \in L^*$ we scale its flow f_A by the factor of $1/x_A$, so that A now receives N_A flow units. For agents not in 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 $O(\log n)$.

Step 3: Final solution In the final solution, we require that each agent $A \in L^*$ receives $\lfloor N_A / \log n \rfloor$ flow units *integrally* via internally disjoint paths from s. Since our original flow has congestion $O(\log n)$, we know that such a *fractional flow* exists. By integrality of flow we can get such an integral flow.

We now show that this algorithm is enough to get an $\tilde{O}(\sqrt{n})$ approximation for max-min allocation. 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 modification of Lemma 4 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 $O(\log n)$-approximation for resulting canonical instances and $\tilde{O}(\sqrt{n})$-approximation overall.

4 Almost Feasible Solutions for Multi-Layered Instances

We now generalize the algorithm from the previous section to arbitrary number of layers. From Lemma 4, it is enough to use $h = 8/\epsilon$ layers (recall that $1/\epsilon = O(\log n / \log\log n)$).

There is a natural generalization of the algorithm from the previous section, where we write a multi-layered flow LP, and then perform h rounds of randomized rounding, starting from the last layer. This is what the algorithm presented here does. However we are unable to obtain a feasible solution. Instead we obtain an almost-feasible solution in the following sense: all constraints ($\hat{\text{C}}1$)–($\hat{\text{C}}3$) hold, except that for some items and some heavy agents there will be two simple paths containing them: one terminating at some light agent and one at a terminal. Similarly, some light agents A will have two simple paths starting at A, one terminating at another light agent and one 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 B in the Appendix. Surprisingly we manage to bypass this obstacle in our final algorithm presented in Section 5, and obtain a better approximation guarantee while using the LP-rounding algorithm from this section as a subroutine.

More formally, in this section our goal is to prove the following theorem.

Theorem 4 *Let $\mathcal{I} = (\mathcal{A}, I)$ be any 1-satisfiable canonical instance and let P be a good assignment of private items to non-terminal agents. Let $N(\mathcal{I}, P)$ be the corresponding flow network and let $\alpha = O(h^4 \log n)$. Then we can find, in polynomial time, two collections $\mathcal{P}_1$ and $\mathcal{P}_2$ of simple paths 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 some 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 they may share endpoints. A non-terminal agent or an item may appear both in $\mathcal{P}_1$ and in $\mathcal{P}_2$.*
- *D3. For each light agent A, 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 $A \in L$, then there are at least N_A/α paths in $\mathcal{P}_2$ that terminate in A.*

Our LP itself is defined on a new graph $N_h(\mathcal{I}, P)$, that can be viewed as a "structured" or "layered" version of $N(\mathcal{I}, P)$. We start by describing the graph $N_h(\mathcal{I}, P)$. After that we define the LP itself, and finally the randomized rounding algorithm.

4.1 The Graph $N_h(\mathcal{I}, P)$

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 $G_{h'}$, $1 \leq h' \leq h$, is an h'-layered graph that will correspond to trees of height $h(\tau) = h'$.

We start with the description of $G_{h'}$. We create h' copies of the set of light agents: one copy for each layer. From now on we will treat these copies as distinct agents. Let $L_1, \ldots, L_{h'}$ denote the sets of agents corresponding to the layers. The graph consists of h' levels, where level j starts with light agents in L_{j-1} and ends with light agents in L_j (the first level starts with the source s and ends with L_1). We now proceed to define each level.

Level 1 contains the source s, a copy of each item in I^* and a copy of each heavy agent in H^* (recall that these are the items and the agents that can be reached directly from s). The assignment of private items is as before. There is an edge from s to every item in S, an edge from every heavy agent A to $P(A)$, and an edge from each item $i \in \Gamma(A) \setminus P(A)$ 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 L_1.

Level j is defined as follows. It contains a copy of each heavy non-terminal agent $A \in H \setminus T$ and a copy of each item $i \in I \setminus S$. Let H_j and I_j denote the set of the heavy agents and items at level j. Recall that each item $i \in I \setminus S$ 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 H_j to a copy of i in I_j. If it is a private item of light agent A, then we add an edge from a copy of A in L_{j-1} to a copy of i in 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 I_j to a copy of A in H_j. If i is a light item for a light agent A then we add an edge from a copy of i in I_j to a copy of A in L_j. This completes the description of $G_{h'}$. We will denote by $H_j^{h'}, I_j^{h'}$ 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 part to our graph that will be used to route flow directly to terminals.

This part consists of a set $\hat{H}$ containing a copy of every heavy agent and a set $\hat{I}$ containing a copy of every item $i \in I \setminus S$. Note that every item in $\hat{I}$ is a private item of some agent. If $i \in \hat{I}$ is a private item of a heavy agent A then we add an edge from a copy of A in $\hat{H}$ to a copy of i in $\hat{I}$. 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 $L_{h'}^{h'}$ to the copy of i in $\hat{I}$ (so there are only edges from the last layer of each $G_{h'}$ to items in $\hat{I}$). 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 $\hat{I}$ to the copy of A in $\hat{H}$.

This completes the description of the graph $N_h(\mathcal{I}, P)$. Notice that for each item $i \in I$ and each agent $A \in \mathcal{A}$ of $\mathcal{I}$, there are at most h^2 copies in $N_h(\mathcal{I}, P)$. We will be looking for an integral flow in this graph satisfying conditions (Ĉ1)–(Ĉ3). Notice that given an optimal solution to the original instance, it is easy to convert it into a feasible integral flow in the new instance. Moreover, in any feasible integral flow solution to the new instance, each tree τ is contained in some subgraph $G_{h'}$ (except for the final path that reaches the root directly), and if τ is contained in $G_{h'}$ then its height $h(\tau) = h'$. The edges connecting $G_{h'}$ to the rest of the graph only leave vertices in $L_{h'}$, and so any path from the source to any terminal that visits $G_{h'}$ has to visit a light agent in every layer $L_j^{h'}$ of $G_{h'}$.

4.2 The LP

We start with the 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 each light agent A in the last layer $L^{h'}_{h'}$ of $G_{h'}$, we have a variable x_A showing to which extent A is satisfied as a light agent (and equivalently, how much flow it sends to the terminals). We then write standard flow constraints on the part of the graph induced by $((\bigcup_{h' \leq h} L^{h'}_{h'}) \cup \hat{H} \cup \hat{I})$ requiring that each terminal receives one flow unit, and each agent $A \in L_h$ sends x_A flow units. We also have capacity constraints requiring that at most one flow unit traverses any vertex. We will eventually perform the same LP rounding as before on this part, using the BS procedure to obtain tree decomposition and randomized rounding to select layer-h' light agents in each subgraph $G_{h'}$ that will be satisfied as light agents. So this part is relatively straightforward, and similar to one from the preceding section.

After that we focus on each subgraph $G_{h'}$ separately. Our starting point is the set of randomly selected light agents to be satisfied in the last layer h' of $G_{h'}$. In the rounding algorithm, we will perform an iterative randomized rounding procedure to create a feasible integral solution that originates at the source s and satisfies these agents. In iteration j, we will select light agents in layer $(h-j)$ that need to be satisfied. Thus we will use light agent selection as an interface between successive layers, with each iteration moving the rounding process to the next layer. In the first iteration, for each selected agent A, 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 agents in layer $h'-1$ 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 layers makes the LP somewhat complex. The LP will consist of three parts. The first and the easiest part is sending the flow to the terminals directly. From this point onwards 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 $\mathcal{L} = \bigcup_{h'} L^{h'}_{h'}$). For each light agent $A \in \mathcal{L}$ we have a variable x_A signifying whether or not A is satisfied as a light agent, 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 $\hat{I}$ and heavy agents in $\hat{H}$ and there are no additional light agents on the path. We have the following constraints:

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

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

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

Notice that each heavy agent $A \in \hat{H} \setminus T$ 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 $L_0 = \{s\}$.

For each $j : 0 \leq j \leq h'$, we define a set $\mathcal{S}_j = L_{h'} \times L_{h'-1} \times \cdots \times L_j$. For each tuple $\lambda = (\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j) \in \mathcal{S}_j$ of light agents, where $\ell_k \in L_k$ for $1 \leq k \leq j$, 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 $\mathcal{S}_{h'} = L_{h'}$, and so we have the constraint:

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

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 $A \in L_{j-1}$, 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:

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

Additionally, for $j \geq 2$, each $A \in L_{j-1}$ 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):

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

Finally, we need to add the more complex capacity constraints that will ensure that the randomized rounding procedure will go through. Consider some tuple $\lambda(\ell_{h'}, \ell_{h'-1}, \ldots, \ell_j) \in \mathcal{S}_j$, and let $A \in L_k$ 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 $A \in L_k$ 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:

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

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:

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

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 $\ell_{j-1} \in L_{j-1}, \ell_j \in L_j$ 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:

$$\forall 0 \leq j \leq h', \forall \lambda = (\ell_{h'}, \ldots, \ell_j, \ell_{j-1}) \in \mathcal{S}_{j-1}, \quad \sum_{p \in \mathcal{P}(\ell_{j-1}, \ell_j)} f_\lambda(p) = y(\lambda) \quad \text{(routing flow of each type)} \tag{18}$$

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

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

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 I_k, where $k < j$. Then the total flow that i can send via all tuples whose prefix is λ is at most $y(\lambda)$. Given $\lambda \in \mathcal{S}_j$ and layer L_k with $k < j$, 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:

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

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, and has capacity constraints on vertices. 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 $n^{O(1/\epsilon)}$.

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 simple paths connecting light agents to terminals in the original graph $N(\mathcal{I}, P)$. 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 that are used to satisfy the light agents.

Part 1: Routing to Terminals We consider the sub-graph of $N_h(\mathcal{I}, P)$ induced by the set $Y = \hat{H} \cup \mathcal{L}$ of agents (where $\mathcal{L}$ contains the light agents in the last layer of every subgraph $G_{h'}$, $\mathcal{L} = \bigcup_{h' \leq h} L^{h'}_{h'}$) and the set $\hat{I}$ of items. Recall that the items in $\hat{I}$ are heavy items for all agents in Y. For each agent $A \in \mathcal{L}$ we have the value x_A defined by our LP-solution, while for each agent $A \in \hat{H}$ we set $x_A = 0$. Exactly like in the first part of the rounding algorithm for Section 3, we can produce values $y_{A,i}$ for each $A \in Y$, $i \in \hat{I}$ satisfying the constraints (1)–(3). We then again apply Theorem 3 to obtain a decomposition of the bipartite graph $G(Y, \hat{I})$ 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 $\frac{1}{2}$. 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 $2x_A$. 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. We therefore obtain an assignment of items in $\hat{I}$ to agents in $Y \setminus \mathcal{L}'$, where each agent is assigned one item. This assignment of items defines an integral flow in the sub-graph of $N_h(\mathcal{I}, P)$ induced by $Y \cup \hat{I}$, as follows. Every light agent in $\mathcal{L}'$ sends one flow unit to its private item. If agent $A \in Y \setminus \mathcal{L}'$ is assigned item $i \neq P(A)$, then i sends one flow unit to A and if $A \notin T$, it sends one flow unit to $P(A)$. This flow is a collection of disjoint simple paths connecting items in $\mathcal{L}'$ to the terminals. Let $\mathcal{P}_1$ denote the corresponding collection of paths in $N(\mathcal{I}, P)$, (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 $L^{h'}_{h'}$ for each graph $G_{h'}$ connect to $\hat{I}$, all these copies only connect to a single copy of $h(A)$ in $\hat{I}$ 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 $L'_{h'} \subseteq L_{h'}$ of light agents that have been selected in Part 1.

Given a tuple $\lambda(\ell_p, \ldots, \ell_j) \in \mathcal{S}_j$, we say that flow f_λ belongs to agent ℓ_p. In the first iteration, for each light agent $A \in L'_{h'}$, we scale all the flow belonging to A by the factor of $1/x_A$. For light agents $A \notin L'_{h'}$, we remove all their flow from the graph. We define a subset of tuples $\mathcal{S}'_{h'} \subseteq \mathcal{S}_{h'}$ to be $\mathcal{S}'_{h'} = L'_{h'}$. In this way we get a flow that is "integral" for layer h' and fractional for remaining layers. In general, in iteration $(h' - j)$ we will get a flow that is integral for layers $j, \ldots, h'$ and fractional for the remaining layers. We will claim that the solution obtained in each iteration satisfies constraints (13)–(16) and constraints (18),(20), while constraints (17) and (19) (general

capacity constraints for light agents and items) are satisfied approximately, with polylogarithmic congestion.

Since each non-terminal heavy agent has exactly one out-going edge (that connects it to its private item), it is enough to bound the congestion on items and light agents.

We now consider iteration $(h' - j)$. The input is a subset $\mathcal{S}'_j \subseteq \mathcal{S}_j$ of tuples, such that for every $\lambda \in \mathcal{S}'_j$, if $\lambda = \{\ell_{h'}, \ldots, \ell_j\}$, then our fractional solution routes N_A flow units of types $\lambda' \in \mathcal{S}_{j-1}$, where λ is a prefix of λ', to A, and for $\lambda \notin \mathcal{S}'_j$, no flow of any type λ' where λ is a prefix of λ' is present. Moreover, the constraints (13)–(16) and constraints (18),(20) hold, while constraints (17) and (19) (general capacity constraints for light agents and items) are satisfied approximately, with congestion of at most $(16h^2 \cdot \log n)(1 + \frac{1}{h})^j$ for each item and each light agent. Let $\lambda \in \mathcal{S}'_j$, and let $A \in L_j$ be the last agent in λ. Let $Z \subseteq \mathcal{S}_{j-1}$ be the collection of tuples whose prefix is λ and last agent is any light agent from L_{j-1}. Then due to Constraint (14), the summation of $y(\lambda')$ for $\lambda' \in Z$ is N_A. We randomly select each tuple $\lambda' \in Z$ with probability $y(\lambda')$. Notice that by the Chernoff bound, with high probability, for each $\lambda \in \mathcal{S}'_j$, at least $N_A/2$ tuples λ' have been selected. If λ' is selected, then we proceed as follows:

- Randomly sample a flow-path p carrying a flow of type λ' with probability $f_{\lambda'}(p)/y_{\lambda'}$.
- For all $j' < j$, scale all the values $y_{\lambda''}$ and flows of type λ'' for tuples $\lambda'' \in \mathcal{S}_{j'}$ such that λ' is a prefix of λ'' by a factor of $1/y(\lambda')$.

If λ' is not selected, then all flow corresponding to tuples λ'' where λ' is the prefix of λ'' is removed from the graph. It is easy to verify that constraints (13)–(16) and constraints (18),(20) continue to hold (since both sides of such constraints get scaled by the same factor). We now bound the congestion on items and light agents. Consider an item i at some level $k \leq j$. Due to constraint (20), the total flow corresponding to tuples λ'' whose prefix is λ' is at most $y(\lambda)$. Since before iteration $(h' - j)$, w.h.p. congestion on i was at most $(16h^2 \cdot \log n)(1 + \frac{1}{h})^j$, using Chernoff bound, with high probability the congestion goes up by at most a factor of $(1 + \frac{1}{h})^1$. A similar argument works for bounding congestion on light agents in the fractional part, and for rounding the congestion on items and heavy agents induced by the integral paths we have selected at level j.

At the end of the above procedure, we obtain a set $\mathcal{P}'$ of simple paths. If $A \in L^{h'}_{h'}$ or A has a simple path leaving it in $\mathcal{P}'$ then with high probability it has at least $N_A/2$ paths entering it in $\mathcal{P}'$. The total number of paths leaving a light agent is bounded by $O(h^2 \cdot \log n)$ and similarly the total number of paths to which a heavy agent or an item can belong is bounded by $O(h^2 \log n)$ with high probability.

Getting Almost-Feasible Solution In the last step of the algorithm we produce a set $\mathcal{P}_2$ of simple paths, such that properties (D1)–(D4) hold for $\mathcal{P}_1, \mathcal{P}_2$. Let $\mathcal{L}^* \subseteq L$ be the set of light agents in the original instance from which paths in $\mathcal{P}_1$ originate.

Consider the flow-paths in $\mathcal{P}'$, and let $\mathcal{P}''$ be the corresponding paths in the original graph $N(\mathcal{I}, P)$ (where we replace copies of agents and items by their original counterparts). These flow-paths have

[1]For any $0 < \delta \leq 2e - 1$, the probability that a random variable $Z = \sum_{i=1}^{N} Z_i$, where Z_i's are independent 0/1 random variables, deviates from its expectation by more than $(1 + \delta)$ is at most $e^{-(\delta^2 \mathrm{E}[Z])/4}$. So, if $\mathrm{E}[Z] \geq 16h^2 \log n$, this probability is at most $1/n^4$ for $\delta = 1/h$.

the following properties: (i) every vertex that does not correspond to a terminal may appear in at most $\alpha' = h^4 \log n$ flow-paths in $\mathcal{P}''$, and (ii) for any light agent A, there are at most α' paths in $\mathcal{P}''$ starting at A, and if there is at least one path in $\mathcal{P}'' \cup \mathcal{P}_1$ that originates at A, then there must be at least $N_A/2$ paths 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 $\mathcal{P}_1$, $\mathcal{P}_2$.

Lemma 5 *Let $\mathcal{P}$ be any collection of simple paths that all terminate at light agents, and let $\mathcal{L}'$ be the subset of light agents at which these paths terminate. Moreover, assume that for every $A \in \mathcal{L}'$ there are at least $N_A/2$ paths in $\mathcal{P}$ terminating at A, all paths in $\mathcal{P}$ start at vertices in $S \cup \mathcal{L}'$, and for any vertex v, there are at most β paths containing v as a first or an intermediate vertex. Then there is a collection $\mathcal{P}^*$ of paths, such that for each $A \in \mathcal{L}'$, there are $\lfloor N_A/2\beta \rfloor$ paths terminating at A, all paths in $\mathcal{P}^*$ start at vertices in $\mathcal{L}' \cup S$, and each vertex appears at most once as the first or an intermediate vertex of paths in $\mathcal{P}^*$.*

Proof: We call the light agents of $\mathcal{L}'$ *receivers*. A light agent $A \in \mathcal{L}'$ that has at least one path of $\mathcal{P}$ starting at A is called a *sender*.

We now build the following flow network. There is a set $\mathcal{S}$ of vertices corresponding to senders and set $\mathcal{R}$ of vertices corresponding to receivers (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 heavy agent and for each item, and there is a source s and a sink t. Source s connects to every item in S and to every sender with capacity-1 edges. Every receiver connects to t with $\lfloor N_A/2\beta \rfloor$ 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 $i \in \Gamma(A) \setminus \{P(A)\}$. There is an edge from item i to receiver A iff i is a light item for A. The goal is to route $\sum_{A \in \mathcal{R}} \lfloor N_A/2\beta \rfloor$ 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 β. Therefore, if we scale this flow down by the factor of β we will obtain a feasible flow of the desired value. From the integrality of flow, we can obtain integral flow of the same value. It is easy to see that this flow will define the desired set $\mathcal{P}^*$ of paths. □

This finishes the algorithm for getting an almost-feasible solution.

5 An $\tilde{O}(n^{\epsilon})$-Approximation Algorithm

We now show that the algorithm from Section 4 for obtaining an almost-feasible solution can be used as a building block to get an $\tilde{O}(n^{\epsilon})$-approximation. We use an iterative approach where each iteration j begins with a canonical instance $\mathcal{I}^j$, and a subset $\mathcal{L}^j$ of light agents such that each light agent $A \in \mathcal{L}^j$ is *satisfied* using (light) items from $S(A)$. We then generate an almost-feasible solution to the canonical instance $\mathcal{I}^j$ containing agents $\mathcal{A} \setminus \mathcal{L}^j$. The items that satisfied agents in $\mathcal{L}^j$, however, are made available for *re-use* in solving the iteration j canonical instance. The final step is to compose the previously computed assignment to $\mathcal{L}^j$ with an almost-feasible solution to the instance $\mathcal{I}^j$ to generate a new set $\mathcal{L}^{j+1}$ of light agents satisfied by light items, and a new canonical instance $\mathcal{I}^{j+1}$. The algorithm makes progress by reducing the number of terminals in each

successive iteration, until no more terminals remain and we have an $\tilde{O}(n^{\epsilon})$-approximate assignment for all agents.

5.1 The Algorithm

We start by defining the notion of partially satisfying a subset of light agents. Let $\mathcal{Q}$ be a collection of simple paths, such that all paths in $\mathcal{Q}$ terminate at light agents, and let $L' \subseteq L$ be any subset of light agents. We say that $\mathcal{Q}$ *α'-satisfies* L' iff

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

We now describe the algorithm in detail. The algorithm consists of h iterations (recall that $h = 8/\epsilon$). The input to iteration j is a subset $\mathcal{L}^j \subseteq L$ of light agents, a set $T^j \subseteq H$ of terminals and an assignment of private items $P^j : \mathcal{A} \setminus (\mathcal{L}^j \cup T^j) \to I$. Additionally, in the resulting flow network $N(\mathcal{I}^j, P^j)$, we have a collection $\mathcal{Q}^j$ of simple paths that α_j-satisfy agents in $\mathcal{L}^j$ where $\alpha_j = 2j\alpha$; here α is the approximation factor from Theorem 4. The output of iteration j is a valid input to iteration $(j+1)$, that is, sets $\mathcal{L}^{j+1}$,T^{j+1}, an assignment of private items $P^{j+1} : \mathcal{A}\setminus(\mathcal{L}^{j+1}\cup T^{j+1}) \to I$ and a collection $\mathcal{Q}^{j+1}$ of simple paths in $N(\mathcal{I}^{j+1}, P^{j+1})$ that α_{j+1}-satisfy $\mathcal{L}^{j+1}$.

The size of the set T^j decreases in each iteration by a factor of at least $n^{\epsilon}/(32h^2\alpha)$, so after h iterations, $|T^{h+1}| \leq |T|/(n^{\epsilon}/(32h^2\alpha))^h$. Since $h \leq \log n/\log\log n$, $n^{\epsilon} \geq \log^8 n$ and $\alpha = O(h^4 \log n)$, we have that $32h^2\alpha = O(h^6 \log n) = O(\log^7 n) = O(n^{7\epsilon/8})$. Thus, for large enough n, $(n^{\epsilon}/(32h^2\alpha))^h > (n^{\epsilon/8})^h = n$.

Therefore, $|T^{h+1}| < 1$, that is, T^{h+1} is empty and P^{h+1} assigns a private item to each agent in $\mathcal{A}\setminus\mathcal{L}$. Furthermore, $N(\mathcal{I}^{h+1}, P^{h+1})$ contains a collection $\mathcal{Q}^{h+1}$ of paths that α_{h+1}-satisfy the agents in $\mathcal{L}^{h+1}$ – the only agents without private items. The flow-paths in Q^{h+1} together with the assignment P^{h+1} of private items then define an $\alpha_{h+1} = 2(h+1)\alpha$-approximate solution.

In the input to the first iteration, $\mathcal{L}^1 = \emptyset$, $\mathcal{Q}^1 = \emptyset$. Each light agent $A \in L$ is assigned its heavy item as private item, $P^1(A) = h(A)$, and the assignment of private items to heavy agents is performed by calculating a maximum matching between the set of heavy agents and the remaining items. This is as in Section 2.3.

Iteration j (for $j = 1$ to h) 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 $\mathcal{L}^j$ from this instance. Let $\mathcal{N}_j = N(\mathcal{I}^j, P^j)$ be the corresponding flow network. Note that we do not remove any items from the instance. Thus, the integral optimum for this instance cannot decrease, and in particular the value of the optimal solution is at least $(h+1) \cdot N_A$ (recall that we have scaled the values N_A down by factor $(h+1)$). Therefore, there exists an h-layered solution of value N_A, implying the LP described in Section 4

must be feasible for this instance as well. Conversely, if the LP is not valid for $\mathcal{N}_j$, we say that the problem is infeasible.

We now apply the algorithm from Section 4 to $\mathcal{N}_j$, obtaining the two sets $\mathcal{P}_1$, $\mathcal{P}_2$ of simple paths, satisfying the properties D1–D4 as in Theorem 4.

Let $\mathcal{L}'$ 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 A, there are at least N_A/α paths in $\mathcal{P}_2$ terminating at A. Let $\mathcal{L}''$ be the set of light agents A such that either $A \in \mathcal{L}^j$, 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 $\mathcal{Q}^j$.

Our first step is to construct a set $\mathcal{Q}^*$ with the following property.

Claim 1 *There exists a set of internally-disjoint simple paths $\mathcal{Q}^*$ such that each agent A in $\mathcal{L}' \cup \mathcal{L}''$ has at least $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ paths terminating at A. Moreover, only light agents in $(\mathcal{L}' \cup \mathcal{L}'') \setminus \mathcal{L}^j$ have paths in $\mathcal{Q}^*$ originating from them, with at most one path originating from any agent.*

Proof: This is done similarly to Lemma 5, where light agents in $\mathcal{L}' \cup \mathcal{L}''$ serve as receivers and light agents in $(\mathcal{L}' \cup \mathcal{L}'') \setminus \mathcal{L}^j$ are the senders. Each receiver A is connected to the sink t with $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ edges of capacity 1. A source connects to all senders and items in S with edge of capacity 1. Consider the flows defined by paths in $\mathcal{P}_2$ and $\mathcal{Q}^j$. We send $\alpha/(\alpha_j + \alpha)$ flow units 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 $N_A/(\alpha_j + \alpha)$ flow units. From the integrality of flow, there is a collection $\mathcal{Q}^*$ of desired paths. □

Our next step is to resolve the conflicts between paths in $\mathcal{Q}^*$ and $\mathcal{P}_1$ when such paths share vertices. We will use Lemma 6 below to do so. The idea is to re-route the paths in $\mathcal{P}_1$ to get $\mathcal{P}_1'$, so that each such path only interferes with at most one path in $\mathcal{Q}^*$. We will then remove from $\mathcal{Q}^*$ the paths that share vertices with the re-routed paths and argue that we still make progress.

A Path Rerouting Lemma: For a directed path p starting at some vertex is v, we say that path p' is a *prefix* of p iff p' is a sub-path of p containing v. We will use the following lemma whose proof follows from the Spider Decomposition Theorem of [7] and appears in Appendix.

Lemma 6 *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 define, 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 $\gamma(q)$.*

Rerouting Paths in $\mathcal{P}_1$ and $\mathcal{Q}^*$: 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 $\mathcal{P}^*$ that originates from A. In order to apply Lemma 6 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 $\mathcal{P}_1^* = \{\overline{p} \mid p \in \mathcal{P}_1\}$. In the new set $\mathcal{P}_1^*$, the paths originate at the terminals and terminate at light agents. We now apply Lemma 6 to $\mathcal{P}_1^*$ and $\mathcal{Q}^{**}$ and obtain prefix $\gamma(p)$ for each $p \in \mathcal{P}_1^* \cup \mathcal{Q}^{**}$.

For each $p \in \mathcal{P}_1^*$, we construct a new path p' that will be used to re-route the flow to the terminal of p. 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 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, (notice that in this case since p and q are simple, $\gamma(p) = p$ and $\gamma(q) = q$ must hold), then we set $p' = \overline{p}$. Otherwise, we set p' to be the concatenation of $\gamma(q)$ and $\overline{\gamma(p)}$. In either case path q is removed from set $\mathcal{Q}^*$, and we say that the unique terminal t that lies on path p is *responsible* for the removal of q. We note here that a terminal will be responsible for the removal of *at most* one path q.

Now observe that the first vertex of q has to be a light agent A. For if it is an item in S, then the path $\gamma(q)$ followed by $\overline{\gamma(p)}$ is a simple path from an item in S to a terminal, which is impossible. Therefore, in both cases above, the new path p' has one end point a terminal and the other is a light agent A. This implies $\mathcal{Q}^*$ originally contained at least $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ paths terminating at A.

We denote by $\mathcal{P}_1'$ the resulting set containing paths p' for all $p \in \mathcal{P}_1$, and by $\mathcal{Q}_2$ the set of remaining paths in $\mathcal{Q}^*$. The set $(\mathcal{P}_1' \cup \mathcal{Q}_2)$ "almost" has all the desired properties of the final solution: all paths are internally vertex-disjoint; *each terminal* has exactly one path entering it and each light agent has at most one path leaving it. Moreover, if light agent A has a path leaving it, then *originally*, in $\mathcal{Q}^*$, there were at least $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ paths terminating at A. A potential problem is that it is possible that we have removed many of such paths on moving to $\mathcal{Q}_2$. We now take care of that.

Bad Light Agents: We call a light agent A is *bad* iff there is a path originating at A in $\mathcal{P}_1' \cup \mathcal{Q}_2$ but there are less than $N_A/(\alpha_j + 2\alpha) = N_A/\alpha_{j+1}$ paths terminating at A.

We start with $T^{j+1} = \emptyset$.While there exists a bad light agent A:

- Remove all paths entering A from $\mathcal{Q}_2$.
- If $A \notin \mathcal{L}^j$, then remove the unique path p leaving A from $\mathcal{P}_1'$ or $\mathcal{Q}_2$, and say that A is *responsible* for this path. If $p \in \mathcal{P}_1'$ and t is the terminal lying on p, then we add t to T^{j+1} and say that A is responsible for t.
- If $A \in \mathcal{L}^j$, consider the item $i = h(A)$. If there is a heavy agent A' for which i is a private item, we add A' to T^{j+1} (where it becomes a terminal). In either case, item i becomes the private item for A. We remove A from $\mathcal{L}^j$. If there is any path p containing i in $\mathcal{P}_1' \cup \mathcal{Q}_2$, then we remove p from $\mathcal{P}_1'$ or $\mathcal{Q}_2$ and say that A is responsible for p. If $p \in \mathcal{P}_1'$ and t is a terminal lying on p, then we add t to T^{j+1} and say that A is responsible for t.

It is easy to see that a bad light agent can only be responsible for at most one path in $\mathcal{P}_1' \cup \mathcal{Q}_2$, and at most two terminals in T^{j+1}. Notice that 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 show below that the size of T^{j+1} is small, but first we show how to produce the input to the next iteration (which is the output of the current iteration).

Input to Iteration $(j+1)$**:** We start with $\mathcal{L}^{j+1}$ containing all the remaining good agents in $\mathcal{L}^j$. Consider now the sets $\mathcal{P}'_1 \cup \mathcal{Q}_2$ of paths. Let $p \in \mathcal{P}'_1$, and let A be the first vertex and $t \in T^j$ be the last vertex on p. We then add A to $\mathcal{L}^{j+1}$ 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 of agents not lying in any path remains the same. Let P^{j+1} be the resulting assignment of private items. Note that the only agents with no private items assigned are agents of $\mathcal{L}^{j+1} \cup T^{j+1}$. We set $\mathcal{Q}^{j+1} = \mathcal{Q}_2$. Since no light agent in $\mathcal{L}^{j+1}$ is bad, set $\mathcal{Q}^{j+1}$ ensures that every agent in $\mathcal{L}^{j+1}$ is α_{j+1}-satisfied. Therefore we have produced a feasible input to iteration $(j+1)$.

Bounding the size of T^{j+1}**:** Finally we need to bound the size of T^{j+1}, the set of terminals in iteration $(j+1)$.

Lemma 7 $|T^{j+1}| \leq \left(\frac{32h^2\alpha}{n^\epsilon}\right)|T^j|$.

Proof: We may assume that $n^\epsilon \geq 16h^2\alpha$. Recall that each bad light agent is responsible for at most two terminals in T^{j+1}. Therefore, it is enough to prove that the number of bad light agents in iteration j is at most $\left(\frac{16h^2\alpha}{n^\epsilon}\right)|T^j|$. We build a graph G_B whose vertices are bad light agents and the terminals in 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 such a path, and A' is responsible for q then we add an edge from A to A' in graph G_B (observe that A' can be another bad light agent or a terminal).

Since any bad light agent or terminal 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 we see 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 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

$$n_B + |T_j| \geq \beta n_B$$

implying the number of bad light agents is bounded by $|T_j|/(\beta - 1) \leq 2|T_j|/\beta$ since $\beta \geq 2$. □

5.2 Approximation Factor and Running Time

Let α^* be the approximation factor that we achieve for the canonical instance. The final approximation factor is $\max\{O(n^\epsilon \log n), O(\alpha^* \log n)\}$. We now bound α^* in terms of $\alpha = O(h^4 \log n)$, the approximation factor from Theorem 4.

We lose a factor of $(h+1) \leq 2h$ when converting the optimal solution to an h-layered forest. The algorithm in the final section assigns $N_A/(2h\alpha)$ items to each light agent A. So overall $\alpha^* = O(h^2\alpha) = O(h^6 \log n))$. So we get a $\max\{O(n^\epsilon \log n), O(h^6 \log n)\}$-approximation. When ϵ is chosen to be $(8 \log\log n)/\log n$, we get an $O(\log^9 n)$-approximation algorithm.

The bottleneck in the running time of our algorithm is solving the linear program. As noted earlier, the time taken for solving the LP is $n^{O(1/\epsilon)}$. For our choice of ϵ above, we get an overall running time of $n^{O(\log n/\log\log n)}$.

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

In this section we show how to use the quasi-polynomial time $O(\log^9 n)$-factor algorithm to obtain an $O(m^\epsilon)$ algorithm for any fixed $\epsilon > 0$. Before that we make the following claim.

Claim 2 *There exists an $(\log n)^{O(m \log n)}$-time $O(1)$-approximation to* Max-Min Allocation.

Proof: From Section 2.1 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 there are $O(\log n)$ distinct values of utilities. We assume we are given an instance like this.

For every agent A, we let $v_j(A)$ be the number of items with $u_{A,i} = 2^j$, for $j = 1$ to $s = \lfloor \log 2n \rfloor$. Thus, the optimum solution corresponds to m vectors $v(A) := (v_1(A), \ldots, v_s(A))$. At the cost of losing another factor of 2, we can further assume that each of the $v_i(A)$ is a 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 $v(A)$ for every agent, we can check if there is a feasible assignment of the items respecting $v(A)$, that is, each agent A gets $v_j(A)$ items of utility $u_{A,i} = 2^j$. Construct a bipartite graph $G(U, V, E)$ where U contains s copies of each agent A: $A(1), \ldots, A(s)$, and the vertex set V corresponds to the set of items. An edge goes from $A(j)$ to item i iff $u_{A,i} = 2^j$. The problem of checking if whether the vector $v(A)$ for every 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 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 of all agents), we can get an $O(1)$ approximation to Max-Min Allocation. □

Using the above claim we can get the following.

Theorem 5 *For any constant $\epsilon > 0$, there exists a quasi-polynomial time algorithm which returns a $O(m^\epsilon)$-approximation to* Max-Min Allocation.

If $m < \log^{9/\epsilon} n$, then by the claim above we can get a $O(1)$ approximation in $(\log n)^{O(m \log n)}$ time which is quasi-polynomial if ϵ is a constant. If $m \geq \log^{9/\epsilon} n$, then our main result gives a quasi-polynomial time $O(\log^9 n) = O(m^\epsilon)$-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.

Definition: A Max-Min Allocation problem instance is 2-*restricted* if for each item i there exist at most two agents with $u_{A,i} > 0$. For item i, we denote these two agents as A_i and B_i. Note that $A_i = B_i$, if the item is wanted by only one agent. The 2-restricted Max-Min Allocation instance is *uniform* if for every item i, $u_{A_i,i} = u_{B_i,i}$.

Given a 2-restricted Max-Min Allocation instance $\mathcal{I}(\mathcal{A}, I)$, we construct an undirected graph $G(\mathcal{I})$ as follows. The set of vertices of $G(\mathcal{I})$ is the set $\mathcal{A}$ of agents, and for every item $i \in I$, we have an edge $i = (A_i, B_i)$. Note that $G(\mathcal{I})$ can have parallel edges and self-loops. An edge corresponding to an item i has two weights associated with it, one weight for each endpoint: $w_{A_i,i} = u_{A_i,i}$ and $w_{B_i,i} = u_{B_i,i}$. The interpretation of these weights is as follows: if the edge is oriented from B_i to A_i, then its weight is $w_{A_i,i}$, and if it is oriented towards B_i then its weight is $w_{B_i,i}$. Such a weighted graph will be called a *non-uniformly* weighted graph. If $w_{A_i,i} = w_{B_i,i}$ for all edges i, the graph is called *uniformly weighted*. The 2-restricted Max-Min Allocation problem on instance $\mathcal{I}(\mathcal{A}, I)$ is equivalent to the following orientation problem on $G(\mathcal{I})$.

Non-uniform Graph Balancing: Consider a graph $G(V, E)$ that can have self-loops and parallel edges, where for each edge $e = (u, v)$ we are given two weights $w_{u,e}$ and $w_{v,e}$. Given an orientation $\mathcal{O}$ of edges, for an edge (u, v) we denote $(u \overset{\mathcal{O}}{\rightarrow} v)$ if the edge is oriented towards v and $(v \overset{\mathcal{O}}{\rightarrow} u)$ when it is oriented towards u. The weighted in-degree of a vertex v is $\sum_{e=(u,v)\in E:\ (u \overset{\mathcal{O}}{\rightarrow} v)} w_{v,e}$. The goal is to find an orientation of the edges such that the minimum weighted in-degree of a vertex is maximized. In the uniform version of the graph balancing problem, $w_{u,e} = w_{v,e}$ for each edge $e = (u, v)$.

It is easy to see that the non-uniform (uniform) graph-balancing problem is equivalent to the non-uniform (uniform) 2-restricted Max-Min Allocation. In this section we give a 2-approximation for the 2-restricted non-uniform Max-Min Allocation problem. We also show that even the uniform version of the problem is NP-hard to approximate to within a factor better than 2.

6.1 Approximation Algorithm for Non-Uniform Graph Balancing

Let $\mathcal{I} = (\mathcal{A}, I)$ be the input instance of the 2-restricted Max-Min Allocation problem, and let $G = (\mathcal{A}, I)$ be the corresponding instance of the non-uniform graph balancing problem. We start by guessing the value OPT of the optimal solution via binary search.

For an agent $A \in \mathcal{A}$, let $\delta(A)$ denote the set of adjacent edges in G, that is, $\delta(A) := \{i : u_{A,i} > 0\}$ Given a parameter $M > 0$, let $\mathcal{C}(A)$ define a set of feasible configurations for A with respect to M, that is: $\mathcal{C}(A) := \{S \subseteq \delta(A) : \sum_{i\in S} u_{A,i} \geq M\}$. For any M, we define the following system $LP(M)$ of linear inequalities. We have a variable $z_{A,S}$ for every agent A and every $S \in \mathcal{C}(A)$, that indicates whether or not S is chosen for A. This system of inequalities is equivalent to the configuration LP of Bansal and Sviridenko [3].

$$
\begin{array}{ll}
LP(M): & \\
\quad \forall A \in \mathcal{A}: & \sum_{S \in \mathcal{C}(A)} z_{A,S} = 1 \\
\quad \forall i \in I: & \sum_{S \in \mathcal{C}(A_i): i \in S} z_{A_i,S} + \sum_{S \in \mathcal{C}(B_i): i \in S} z_{B_i,S} = 1 \\
\quad \forall A \in \mathcal{A}, \forall S \in \mathcal{C}(A): & z_{A,S} \geq 0
\end{array}
$$

Notice that $LP(M)$ has a feasible solution for $M = \mathsf{OPT}$, the optimal solution value for instance $\mathcal{I}$. It is well known (see for instance [3, 2]) that if $LP(M^*)$ is feasible, then a solution with value $(1-\epsilon)M^*$ can be found in polynomial time, for any $\epsilon > 0$. We will therefore assume that we are given a feasible solution z for $LP(M)$ with $M \geq (1-\epsilon)\mathsf{OPT}$.

Given agent A and item $i \in \delta(A)$, we say that item i is integrally allocated to A iff $\sum_{\substack{S \in \mathcal{C}(A): \\ i \in S}} z_{A,S} = 1$. If item i is not allocated integrally to A_i or B_i then we say that it is allocated fractionally. Let I' denote the set of items allocated fractionally, and for every agent $A \in \mathcal{A}$, let $I(A)$ denote the set of items integrally allocated to A. We set $M_A = M - \sum_{i \in I(A)} u_{A,i}$.

We now focus on the sub-graph H of $G(\mathcal{I})$ induced by the edges corresponding to items in I'. We remove from H all isolated vertices. Observe that H does not contain self-loops but may contain parallel edges. Moreover, for each agent A in H, $M_A > 0$. It now suffices to allocate the items of I' to the agents of H, such that every agent A gets a utility of at least $M_A/2$ – this will give a $(2+\epsilon)$-approximation algorithm. For each agent A, let $\delta'(A)$ denote the set of edges adjacent to A in H.

We begin with the following observation about H.

Claim 3 *Fix an agent A. Let i^* be an item in $\delta'(A)$ with maximum value $u_{A,i}$. Then*

$$\sum_{i \in \delta'(A) \setminus \{i^*\}} u_{A,i} \geq M_A$$

That is, the total utility of the fractional items having positive utility for agent A is at least $(M_A + u_{A,i^*})$.

Proof: Since item i^* is fractionally allocated, there is a configuration $S \in \mathcal{C}(A)$ with $z_{S,A} > 0$ such that $i^* \notin S$. Clearly, $\sum_{i \in S \cap I'} u_{A,i} \geq M_A$. □

The above claim implies that for every vertex $A \in V(H)$, the non-uniform weighted degree of A is at least $(M_A + \max_{i \in \delta'(A)} \{u_{A,i}\})$. We now prove a theorem about weighted graph orientations that will complete the proof.

Given an arbitrary graph $G = (V, E)$, for each vertex $v \in V$, we denote by $\Delta(v)$ the set of edges incident on v. Given an orientation $\mathcal{O}$ of edges, we denote by $\Delta^-_{\mathcal{O}}(v)$ and $\Delta^+_{\mathcal{O}}(v)$ the set of incoming and outgoing edges for v, respectively.

Theorem 6 *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:*

$$\sum_{e \in \Delta^-_{\mathcal{O}}(v)} w_{v,e} \geq \frac{\sum_{e \in \Delta(v)} w_{v,e} - \max_{e \in \Delta(v)} \{w_{v,e}\}}{2}$$

We apply Theorem 6 to graph H. The resulting orientation of edges implies an assignment of items in $\mathcal{I}'$ to agents of H. It is easy to see from Claim 3 that the total utility of items assigned to any agent A of H is at least $M_A/2$, and thus together with the assignment of the integral items we obtain a factor $(2+\epsilon)$-approximation for the 2-restricted Max-Min Allocation problem. We now turn to prove Theorem 6.

Proof: The proof is by induction on the number of edges of 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 now need the following claim.

Claim 4 *We can 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 4. 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, and assume that $v_{t+r} = v_r$. 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. 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. □

□

6.2 Hardness of Approximation for Uniform Graph Balancing

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 uniform 2-restricted Max-Min Allocation, since the two problems are equivalent.

Theorem 7 *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. [8]. 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 [15]

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. □

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-SIAM Symposium on the Theory of Computation (STOC)*, pages 114–121, 2007.

[3] N. Bansal and M. Sviridenko. The Santa Claus Problem. *Proceedings of ACM-SIAM 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. *Personal Communication*, December 2008.

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

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

[7] J. Chuzhoy and S. Khanna. Algorithms for single-source vertex connectivity. *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 105–114, 2008.

[8] 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.

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

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

[11] M. X. Goemans, N. J. A. Harvey, S. Iwata, and V. Mirokkni. Approximating submodular functions everywhere. *ACM-SIAM Annual Symposium on Discrete Algorithms (SODA)*, To appear, 2009.

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

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

[14] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. *ACM Conference on Electronic Commerce*, pp. 125–131, 2004.

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

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

[17] 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 Proof of Lemma 6

We use the following definitions from [7].

Definition: [Canonical Spider] Let $\mathcal{M}$ be any collection of simple paths, such that each path $p \in \mathcal{M}$ has a distinguished endpoint $t(p)$, and the other endpoint is denoted by $v(p)$. We say that paths in $\mathcal{M}$ form a *canonical spider* iff $|\mathcal{M}| > 1$ and there is a vertex v, such that for all $p \in \mathcal{M}$, $v(p) = v$. Moreover, the only vertex that appears on more than one path of $\mathcal{M}$ is v. We refer to v as the *head* of the spider, and the paths of $\mathcal{M}$ are called the *legs* of the spider.

Definition: [Canonical Cycle] Let $\mathcal{M} = \{g_1, \ldots, g_h\}$ be any collection of simple paths, where each path g_i has a distinguished endpoint $t(g_i)$ that does not appear on any other path in $\mathcal{M}$, and the other endpoint is denoted by $v(g_i)$. We say that paths of $\mathcal{M}$ form a *canonical cycle*, iff (a) h is an odd integer, (b) for every path g_i, $1 \leq i \leq h$, there is a vertex $v'(g_i)$ such that $v'(g_i) = v(g_{i-1})$ (here we use the convention that $g_0 = g_h$), and (c) no vertex of g_i appears on any other path of $\mathcal{M}$, except for $v'(g_i)$ that belongs to g_{i-1} only and $v(g_i)$ that belongs to g_{i+1} only.

Theorem 8 *(Theorem 4 in [7]) Given any collection $\mathcal{P}$ of paths, where every path $f \in \mathcal{P}$ has a distinguished endpoint $t(f)$ that does not appear on any other path of $\mathcal{P}$, we can find, in polynomial time, for each path $f \in \mathcal{P}$, a prefix $\gamma(f)$, such that in the graph induced by $\{\gamma(f) \mid f \in \mathcal{P}\}$, the prefixes appearing in each connected component either form a canonical spider, a canonical cycle, or the connected component contains exactly one prefix $\gamma(f)$, where $\gamma(f) = f$ for some $f \in \mathcal{P}$.*

We simply apply Theorem 8 to set $\mathcal{P}_1 \cup \mathcal{P}_2$. Consider now some connected component $\mathcal{C}$ in the graph induced by the prefixes. It is impossible that $\mathcal{C}$ is a canonical cycle since a canonical cycle contains an odd number of paths where every pair of consecutive paths intersect. Therefore, each component $\mathcal{C}$ either contains vertices of exactly one prefix $\gamma(p)$ and $p = \gamma(p)$ in this case, or it contains vertices of two prefixes $\gamma(p)$ and $\gamma(p')$ that form a canonical spider.

B 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 4. We then show how the algorithm described in Section 5 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 $S = \cup_j S(L_j)$, note that $|S| = M^2$. Items in 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 also 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 $h(L_1), \ldots, h(L_M)$.

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.

Thus the set of terminal are all the heavy agents. 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 $N(\mathcal{I}, P)$ for the case $M = 3$.

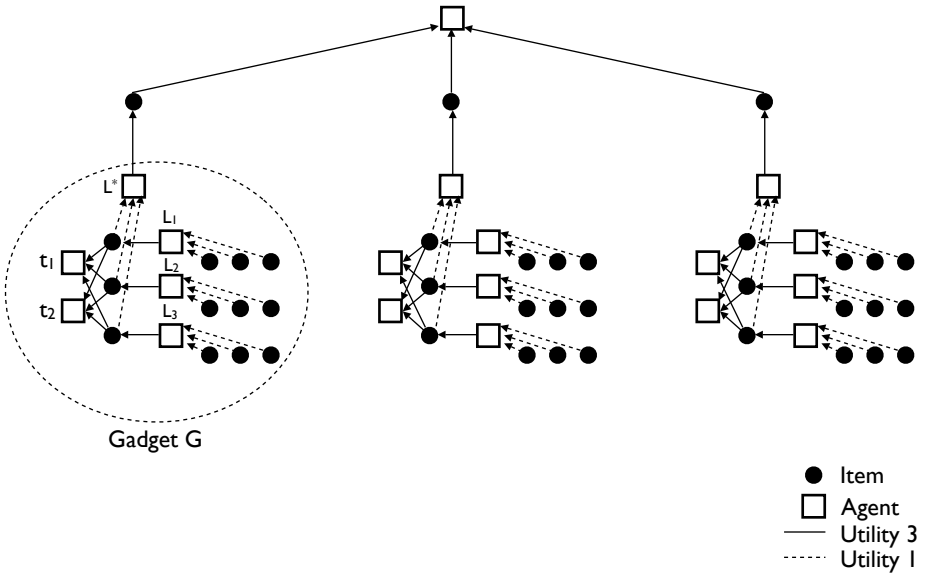

Figure 1: The flow network $N(\mathcal{I}, P)$ for the gap instance with $M = 3$.

We start by showing that in any integral solution, some agent receives a utility of at most 1. That is, any integral flow from 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 argue that there is a fractional flow which 1-satisfies all agents. 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 send $1/M$ flow to terminal t^*.

To be more precise, 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 flows are as described in the previous paragraph. One can check that this satisfies all the constraints of the LP described in Section 4. Thus this is a feasible fractional solution in which each agent is 1-satisfied, and the value of the solution for the Max-Min Allocation problem is M. 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 following assignment of private items – for the remaining light agents we still have $P'(L_j^*) = h(L_j^*)$ and $P'(L_M^j) = h(L_M^j)$ for $1 \leq j \leq M$, but now we assign a private item for every heavy agent t_i in each gadget, $P'(t_i) = h(L_i)$, 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 $N(\mathcal{I}', P')$ for $M = 3$ is shown in Figure 2.

Note that the set S of items which are not private items in each gadget G_j is still $\bigcup_j S(L_j)$. However, 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 4 is not feasible. In fact, even if $N_{L_j^*}$ for every gadget is reduced from M to 2, the LP is not feasible. Since t^* receives a flow of value 1, it must receive a flow of at least $1/M$ from one of the L_j^*. Thus $x(L_j^*) \geq 1/M$. 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 $(\mathcal{I}', P')$ described above. After the rounding algorithm described in Section 4, we get set of paths $\mathcal{P}_1, \mathcal{P}_2$ which are

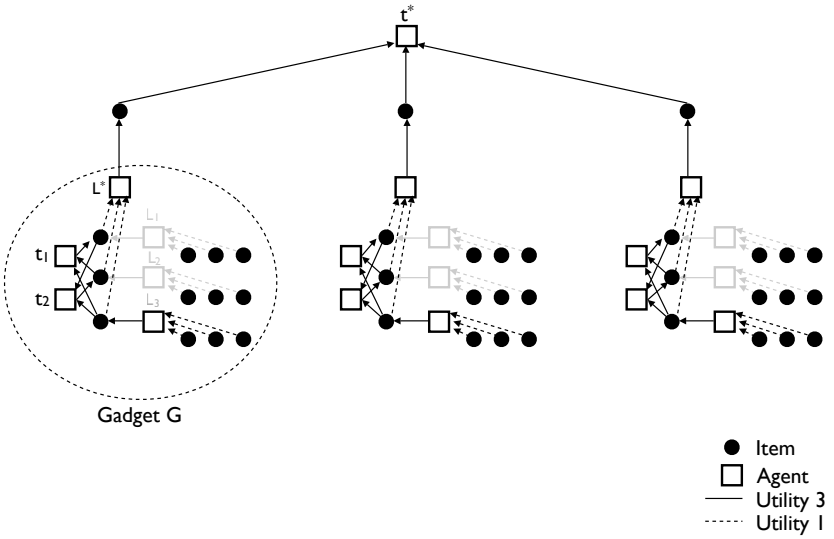

Figure 2: The flow network $N(\mathcal{I}', P')$. Note that the edges between t_i and $h(L_i)$ have flipped for all $1 \leq i \leq (M-1)$.

as follows:

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

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 $S(L_i)$ to L_i in all gadgets and L_i^1 to L_1^* for the gadget G_1. The path decomposition procedure of Section 5 returns one bad light agent (L_1^*) and the following sets of internally disjoint paths

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

Subsequently, the new set of terminals is $T_2 = \{t^*\}$ and the set of discarded light agents are $\mathcal{L}_2 = \{L_i^j : 1 \leq i \leq M-1,\ 1 \leq j \leq M\}$ and thus we get the instance $(\mathcal{I}', P')$. Hence, in the second iteration, the algorithm will return that the optimum M is infeasible for this Max-Min Allocation instance.