2009 50th Annual IEEE Symposium on Foundations of Computer Science

On Allocating Goods to Maximize Fairness

Deeparnab Chakrabarty
Dept. of C&O
University of Waterloo
Waterloo, ON N2L 3G1
Email: deepc@math.uwaterloo.ca

Julia Chuzhoy*
Toyota Technological Institute
Chicago, IL 60637
Email: cjulia@tti-c.org

Sanjeev Khanna†
Dept. of Computer & Information Science
University of Pennsylvania
Philadelphia, PA 19104.
Email: sanjeev@cis.upenn.edu

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

Our main result is an algorithm that achieves an $\tilde{O}(n^{\epsilon})$-approximation for any $\epsilon = \Omega(\frac{\log\log n}{\log n})$ in time $n^{O(1/\epsilon)}$. In particular, we obtain poly-logarithmic approximation in quasi-polynomial time, and for every constant $\epsilon > 0$, we obtain an $\tilde{O}(n^{\epsilon})$-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. As a corollary of our main result, we also show that for any constant $\epsilon > 0$, an $O(m^{\epsilon})$-approximation can be achieved in quasi-polynomial time.

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

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

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

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

As for the general Max-Min Allocation problem, Bansal and Sviridenko [3] showed that the configuration LP has an integrality gap of $\Omega(\sqrt{m})$ in this setting. Asadpour and Saberi [2] gave an $O(\sqrt{m}\log^3 m)$ approximation algorithm for the problem using the same LP relaxation. This is the best approximation algorithm known for the problem prior to our work, while the best current hardness of approximation factor is 2 [6]. The main result of our paper is an $\tilde{O}(n^{\epsilon})$-approximation algorithm for any $\epsilon = \Omega(\log\log n/\log n)$ for the general Max-Min Allocation problem, whose running time is $n^{O(1/\epsilon)}$. This implies a

* Supported in part by NSF CAREER award CCF-0844872.
† Supported in part by a Guggenheim Fellowship, an IBM Faculty Award, and by NSF Award CCF-0635084.

quasi-polynomial time poly-logarithmic approximation for the general MAX-MIN ALLOCATION problem, and for any constant $\epsilon > 0$, it gives an n^ϵ-approximation in polynomial time. Interestingly, one of our main building blocks is an LP-relaxation whose integrality gap is $\Omega(\sqrt{m})$. We bypass this obstacle by iteratively using the LP solutions to construct a sequence of new instances with diminishing integrality gaps.

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

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

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

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

We now give a brief overview of our approach. Fix an ϵ such that the desired approximation factor is $\tilde{O}(n^\epsilon)$. Our algorithm starts by guessing the value M of the optimal solution. We then define a class of structured MAX-MIN ALLOCATION instances that we call *canonical instances*, and show that any input instance can be transformed into a canonical one, with an $O(\log n)$ loss in the approximation ratio. In a canonical instance, there are only two types of agents – *heavy agents* whose utility for any item is either 0 or M, and *light agents*. Each light agent has a distinct heavy item for which it has utility M, and for every other item, the utility is either M/t or 0, where $t \geq n^\epsilon$ is a large integer. The items with a utility of M/t for a light agent are referred to as the *light items*.

Next we transform the problem of assigning items to agents into a network flow problem, by means of *private items*. Each agent is assigned at most one distinct private item, for which it has utility M. The private item of a light agent is its unique heavy item, and we fix some maximal assignment of remaining items to heavy agents. Of course, if every agent is assigned a private item, we immediately obtain a near-optimal solution. So assume that some agents do not get assigned any private items; such agents are called *terminals*. A key observation is that if the optimal solution value is M, then, given any assignment of private items, there always exists a way of *re-assigning* private items such that every terminal is assigned a heavy item. Re-assignment means a heavy agent "frees" its private item if it gets another heavy item while a light agent frees its private item if it gets t light items. These freed-up private items can then be re-assigned. Thus, given any allocation of private items, we can construct a flow network with the property that there exists an *integral* flow satisfying certain constraints (for instance, out-flow of 1 for light agents implies an in-flow of t). We then design a linear programming relaxation to obtain a feasible fractional flow solution for the above network. Our LP relaxation has size $n^{O(1/\epsilon)}$ when the desired approximation ratio is $\tilde{O}(n^\epsilon)$. However, the integrality gap of the LP relaxation is $\Omega(\sqrt{m})$, and thus directly rounding the LP-solution cannot give a better than $O(\sqrt{m})$ approximation factor.

This brings us to another key technical idea. Although the LP has a large integrality gap, we show that we can obtain a better approximation algorithm by performing LP-rounding in phases. In each phase we solve the LP and run a rounding algorithm to obtain a solution which is *almost feasible*: all terminals get heavy items but some items might be allocated twice. From this almost-feasible solution, we recover a new assignment of private items and hence a new instance of the LP, one that has a much smaller number of terminals than the starting instance. We thus show that in poly$(1/\epsilon)$ phases, either one of these instances will certify that the optimal solution cost is smaller than the guessed value M, or we will get an $\tilde{O}(n^\epsilon)$-approximation.

We note that many previous results use m, the number of agents, as the parameter in the approximation factor, while our result is an $\tilde{O}(n^\epsilon)$ approximation. However, as an easy corollary, we can show that for any constant $\epsilon > 0$, an

$O(m^\epsilon)$-approximation can be achieved in quasi-polynomial time.

Corollary 1 *For any fixed $\epsilon > 0$, there is a quasi-polynomial time $O(m^\epsilon)$-approximation algorithm for* MAX-MIN ALLOCATION.

Our second result is about 2-restricted MAX-MIN ALLOCATION instances.

Theorem 2 *There is a 2-approximation algorithm for the general 2-restricted* MAX-MIN ALLOCATION *problem.*

The 2-restricted MAX-MIN ALLOCATION can be cast as an orientation problem on (non-uniformly) weighted graphs. Our main technical lemma is a generalization of Eulerian orientations to weighted graphs. At a high level, we show that the edges of any (non-uniformly) weighted graph can be oriented such that the total weight coming into any vertex w.r.t. the orientation is greater than half the total weight incident on the vertex in the undirected graph minus the maximum weight edge incident on the vertex. Note that in the case of unweighted graphs, these orientations correspond to Eulerian orientations.

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

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

MAX-MIN ALLOCATION may be viewed as a dual problem to the minimum makespan machine scheduling. Lenstra, Shmoys and Tardos [15] gave a factor 2-approximation algorithm for the problem, and also showed the problem is NP-hard to approximate to a factor better than $3/2$. Closing this gap has been one of the challenging problems in the field of approximation algorithms. Recently Ebenlendr et.al. [10] studied a very restricted setting where each job can only be assigned to two machines, and moreover its processing time for both machines is identical. For this case, [13] give a factor $(7/4)$-approximation algorithm, and show that even this special case is NP-hard to approximate better than a factor $3/2$. Our investigation of the 2-restricted MAX-MIN ALLOCATION is inspired by [10].

Organization. Section 2 presents transformations that allow us to focus on instances and solutions with a special structure. Section 3 shows how to get an *almost-feasible* $\tilde{O}(n^\epsilon)$-approximate solution for $\epsilon = \Omega(\log\log n / \log n)$. However due to lack of space, in this extended abstract we only state the theorem needed for the subsequent sections. In Section 4, we establish our main result, namely Theorem 1. Section 5 gives the proof of Corollary 1. We conclude with a proof of Theorem 2 in Section 6. All proofs missing from this extended abstract can be found in [8].

2. PRELIMINARIES

We are given a set $\mathbf{A}$ of m agents, a set $\mathbf{I}$ of n indivisible items, and non-negative utility $u_{A,i}$ for each agent A and item i. The utility of agent A for a subset $S \subseteq \mathbf{I}$ of items is $u_A(S) := \sum_{i \in S} u_{A,i}$. The MAX-MIN ALLOCATION problem is to find an allocation $\Phi : \mathbf{A} \to 2^{\mathbf{I}}$ of items to agents which maximizes $\min_{A \in \mathbf{A}} \{u_A(\Phi(A))\}$, while $\Phi(A) \cap \Phi(A') = \emptyset$ for $A \neq A'$. We assume that we are given $\epsilon \geq 9 \log\log n / \log n$, and so $n^\epsilon \geq \Omega(\log^9 n)$. Our goal is to produce an $\tilde{O}(n^\epsilon)$-approximate solution in $n^{O(1/\epsilon)}$ time. We use M to denote the (guessed) value of the optimal solution. Our algorithm either produces a solution of value at least $M/\tilde{O}(n^\epsilon)$, or returns a certificate that $M > \mathsf{OPT}$. For an agent A and an item i, we use interchangeably the phrases "A has utility γ for item i" and "item i has utility γ for A" to indicate that $u_{A,i} = \gamma$. We say that item i is *wanted* by agent A iff $u_{A,i} > 0$.

Polynomially Bounded Utilities. We first show that we can assume w.l.o.g. that all utilities are polynomially bounded. We give a simple transformation that ensures that each non-zero utility value is between 1 and $2n$, with at most a factor 2 loss in the optimal solution 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 is at most $2n$. Moreover, it is easy to see that this value is at least n: consider any agent A and the subset $\mathbf{S}$ of items assigned to A by OPT. The total utility of $\mathbf{S}$ for A is at least M, and at least $M/2$ of the utility is received from items i for which $u_{A,i} \geq M/2n$. Therefore, the new utility of set $\mathbf{S}$ for A is at least n.

It is easy to see that any α-approximate solution to the transformed instance implies a (2α)-approximate solution to the original instance.

In what follows, we assume that $M \leq 2n$.

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

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

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

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

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

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

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

This completes the definition of the canonical instance $\mathcal{I}'$. Let Φ be an optimal allocation for the instance $\mathcal{I}$. We will use Φ to construct an allocation Φ' for $\mathcal{I}'$ that 1-satisfies all agents. Consider some agent B in the original instance. The optimal solution Φ assigns a set $\Phi(B)$ of items to B. We can partition $\Phi(B)$ into $(s+1)$ sets, $\varphi_0, \varphi_1, \ldots, \varphi_s$, as follows. The set $\varphi_0 \subseteq \Phi(B)$ contains all items i with $u_{B,i} > 2^s$. For each $j \in \{1, \ldots, s\}$, the set φ_j contains all items i with $2^{j-1} < u_{B,i} \leq 2^j$. Assume first that $\varphi_0 \neq \emptyset$, and let i be any item in φ_0. Then we assign item i to heavy agent $H_0(B)$. The remaining s heavy agents corresponding to B now get assigned one item from Y_B each. The light agents $L_j(B)$ are assigned their heavy items $h(L_j(B))$. All agents corresponding to B are now 1-satisfied. Assume now that $\varphi_0 = \emptyset$. Then there is a $j \in \{1, \ldots, s\}$, such that $u_B(\varphi_j) \geq M/s$. Since each item in φ_j has utility at most 2^j for B, we have $|\varphi_j| \geq M/(s \cdot 2^j) = N_{L_j(B)}$. We assign all items in φ_j to $L_j(B)$, and $h(L_j(B))$ is assigned to the heavy agent $H_j(B)$. Now the remaining s heavy agents are assigned one item of Y_B each. For each one of the remaining light agents, we assign $h(L_{j'}(B))$ to $L_{j'}(B)$. Therefore, the canonical instance has a solution that 1-satisfies all agents.

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

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

Private Items and Flow Solutions. One of the basic concepts of our algorithm is that of private items and flow solutions defined by them. We temporarily assign private items to some of the agents. This assignment of private items then allows us to define a network flow problem, whose solution is equivalent to solving the original problem. The flow-paths in this network can be seen as a re-assignment of the private items. Throughout the algorithm we maintain an assignment π of private items to agents. Such an assignment is called *good* iff it satisfies the following properties: (1) For every light agent $A \in \mathbf{L}$, its private item is $\pi(A) = h(A)$; (2) An item i can be a private item of a heavy agent $A \in \mathbf{H}$ only if $i \in \Gamma_{\mathrm{H}}(A)$; and (3) An item can be a private item for at most one agent. We denote by $\mathbf{S}$ the set of items that do not serve as private items in the assignment π. The set of heavy agents that have private items is denoted by $\mathbf{H}_{\mathsf{pvt}}$, and the remaining heavy agents are called *terminals* and are denoted by $\mathbf{T}$. Notice that every light agent has a private item.

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

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

C1) All flow originates at the source s.
C2) Each terminal agent $A \in \mathbf{T}$ receives exactly one flow unit.
C3) For each heavy agent $A \in \mathbf{H}$, if the outgoing flow is 1 then the incoming flow is 1; otherwise both are 0.
C4) For each item $i \in \mathbf{I}$, if the outgoing flow is 1 then the incoming flow is 1; otherwise both are 0.
C5) For each light agent $A \in \mathbf{L}$, if the outgoing flow is 1 then the incoming flow is at least N_A; otherwise both are 0.

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

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

The next lemma show that the problem of α-satisfying all agents is equivalent to the problem of finding an α-feasible flow. The proof is omitted from this extended abstract.

Lemma 2 *Let $\pi : \mathbf{L} \cup \mathbf{H}_{\mathsf{pvt}} \to \mathbf{I}$ be any good assignment of private items for a given canonical instance $\mathcal{I}$. Then any optimal solution for instance $\mathcal{I}$ gives a feasible flow in $N(\mathcal{I}, \pi)$. Moreover, an integral α-feasible flow in $N(\mathcal{I}, \pi)$ gives an α-approximate solution for the canonical instance $\mathcal{I}$.*

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

An Equivalent Formulation. A flow-path p is called an *elementary path* iff it does not contain any light agents as intermediate vertices, and either a) originates and terminates at light agents, or b) originates at a light agent and terminates at a terminal, or c) originates at the source s and terminates at a light agent. It is easy to verify that the problem of finding an α-feasible flow is *equivalent* to finding a collection $\mathcal{P}$ of elementary paths such that:

$\hat{\mathrm{C}}1$) All paths in $\mathcal{P}$ are internally-disjoint (i.e. they do not share intermediate vertices).

$\hat{\mathrm{C}}2$) Each terminal has exactly one path $p \in \mathcal{P}$ terminating at it.

$\hat{\mathrm{C}}3$) For each light agent $A \in \mathbf{L}$, there is at most one path $p \in \mathcal{P}$ starting at A, and if such a path exists, then there are at least N_A/α paths in $\mathcal{P}$ terminating at A.

3. Almost Feasible Solutions

This section contains one of the main technical parts of our algorithm. We write a linear programming relaxation for the problem which has a feasible solution if $\mathcal{I}$ is an 1-satisfiable canonical instance. Given a feasible fractional solution to the LP, we perform a rounding procedure. The algorithm does not produce a feasible solution. Instead we obtain an *almost-feasible* solution in that all constraints ($\hat{\mathrm{C}}1$)–($\hat{\mathrm{C}}3$) hold, except that for some items and some heavy agents there may be two elementary paths containing them: one terminating at some light agent and another at a terminal. Similarly, some light agents A will have two elementary paths starting at A, one terminating at another light agent and another 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. (We refer the reader to the full version of this paper [8] for the lower bound on the integrality gap.) We bypass this obstacle in our final algorithm presented in Section 4, and obtain a much better approximation guarantee while using the LP-rounding algorithm from this section as a subroutine. The following theorem summarizes the properties of the almost-feasible solution that we obtain in this section.

Theorem 3 *Let $\mathcal{I} = (\mathbf{A}, \mathbf{I})$ be any 1-satisfiable canonical instance, and let $\pi : \mathbf{L} \cup \mathbf{H}_{\mathsf{pvt}} \to \mathbf{I}$ be a good assignment of private items to non-terminal agents, such that $N(\mathcal{I}, \pi)$ does not contain direct paths from source s to any terminal. Let $h = 9/\epsilon$ and $\alpha = O(h^5 \log n)$. Then we can find, in $n^{O(1/\epsilon)}$ time, two collections $\mathcal{P}_1$ and $\mathcal{P}_2$ of elementary paths in $N(\mathcal{I}, \pi)$ with the following properties.*

D1) *All paths in $\mathcal{P}_1$ terminate at the terminals and all paths in $\mathcal{P}_2$ terminate at light agents. Moreover, each terminal lies on 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 may share endpoints. A non-terminal agent or an item may appear in both $\mathcal{P}_1$ and $\mathcal{P}_2$.*

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

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

We briefly sketch the main ideas in the proof of Theorem 3. Given a canonical instance $\mathcal{I}$ and an assignment π of private items, an optimal solution OPT for $\mathcal{I}$ defines a feasible integral flow in $N(\mathcal{I}, \pi)$. Let H be the graph induced by the edges carrying one flow unit in OPT. Our first observation is that we can assume that every source-to-terminal path in H contains at most h light agents, at the cost of losing an $(h+1)$ factor in the approximation ratio, for $h = 9/\epsilon$. Therefore, we can partition graph H into h *levels*, where level i contains all vertices v that have exactly i light agents on the path connecting s to v in H. Using this observation, we create h copies of the original graph, where the ith copy is used to route the flow corresponding to level i in H. We then write an LP-relaxation for the problem of obtaining a 1-satisfying flow in this new graph and perform a standard randomized rounding procedure on it. Our goal is to ensure that, after the randomized rounding, the "congestion" on every vertex (the number of flow-paths using the vertex) is poly-logarithmic. It is easy to see that this is impossible to achieve with the standard flow LP, and instead we use a more complex LP, where we have a variable for every h'-tuple of light agents, for all $1 \leq h' \leq h$. The new LP has size $n^{O(h)} = n^{O(1/\epsilon)}$, and after the randomized rounding procedure we obtain two collections $\mathcal{P}_1$, $\mathcal{P}'_2$ of paths with properties D1 and D4 for $\alpha = (h+1)$. As for the other two properties, they hold for paths in $\mathcal{P}_1$ but not for paths in $\mathcal{P}'_2$, as a vertex (representing an item, a heavy agent or a light agent) may participate in a poly-logarithmic number of paths in $\mathcal{P}'_2$. In our last step, we use standard flow techniques to produce a new set $\mathcal{P}_2$ of paths, such that properties D1–D4 hold for $\mathcal{P}_1, \mathcal{P}_2$, at the cost of increasing the factor α to $O(h^5 \log n)$. The proof of Theorem 3 can be found in [8].

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

We now describe an iterative rounding procedure that uses almost-feasible solutions produced by Theorem 3 as a subroutine to produce an $O(h\alpha)$-approximate solution, where $h = 9/\epsilon$ and α is the approximation factor from Theorem 3.

We start with some notation. Let $\mathcal{Q}$ be a collection of elementary paths, such that all paths in $\mathcal{Q}$ terminate at light agents. We say that $\mathcal{Q}$ *α'-satisfies* $\mathbf{L}'$ for some subset $\mathbf{L}' \subseteq \mathbf{L}$ of light agents iff

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

The algorithm consists of $h = 9/\epsilon$ iterations. The input to iteration j is a subset $\mathbf{L}^j \subseteq \mathbf{L}$ of light agents, a set $\mathbf{T}^j \subseteq \mathbf{H}$ of terminals and an assignment of private items $\pi^j : \mathbf{A} \setminus (\mathbf{L}^j \cup \mathbf{T}^j) \rightarrow \mathbf{I}$. The associated canonical instance $\mathcal{I}^j$ of the problem is identical to $\mathcal{I}$, except that we remove all light agents in $\mathbf{L}^j$ from it. We ensure that π^j is a good assignment of private items for $\mathcal{I}^j$. Additionally, we ensure that in the resulting flow network $N(\mathcal{I}, \pi^j)$, there is a collection $\mathcal{Q}^j$ of elementary paths that α_j-satisfy agents in $\mathbf{L}^j$, for $\alpha_j = 2j\alpha$ (here α is the parameter from Theorem 3). The output of iteration j is a valid input to iteration $(j+1)$, that is, sets $\mathbf{L}^{j+1}$,$\mathbf{T}^{j+1}$, an assignment of private items $\pi^{j+1} : \mathbf{A} \setminus (\mathbf{L}^{j+1} \cup \mathbf{T}^{j+1}) \rightarrow \mathbf{I}$, and a collection $\mathcal{Q}^{j+1}$ of elementary paths in $N(\mathcal{I}, \pi^{j+1})$ that α_{j+1}-satisfy $\mathbf{L}^{j+1}$. The size of $\mathbf{T}^j$ decreases in each iteration and in h iterations $\mathbf{T}^j$ becomes empty, with π^{h+1} assigning a private item to each agent in $\mathbf{A} \setminus \mathbf{L}^{h+1}$. The set $\mathcal{Q}^{h+1}$ of paths in network $N(\mathcal{I}, \pi^{h+1})$ then defines an $\alpha_{h+1} = 2(h+1)\alpha$-approximate solution.

Initialization and Invariants: In the input to the first iteration, $\mathbf{L}^1 = \emptyset$, $\mathcal{Q}^1 = \emptyset$. Each light agent $A \in \mathbf{L}$ is assigned its heavy item as a private item, $\pi^1(A) = h(A)$, and the assignment of private items to heavy agents is performed by computing a maximum matching between the set of heavy agents and the remaining items. Recall that $\mathbf{H}^*$ is the set of heavy agents and $\mathbf{I}^*$ is the set of items directly reachable from the source s in network $N(\mathcal{I}, \pi^1)$, while $\mathbf{S}$ is the set of items that do not serve as private items in π^1. Clearly, each agent in $\mathbf{H}^*$ is assigned an item in $\mathbf{I}^*$ and there are no direct paths from s to any terminal in $\mathbf{T}^1$. Throughout the algorithm, the assignments $\pi^j(A)$ of private items to agents in $\mathbf{H}^*$ do not change, and the set $\mathbf{S}$ of un-assigned items remains unchanged. This ensures that for each iteration j, there are no direct paths from s to any terminal $t \in \mathbf{T}^j$ in $N(\mathcal{I}, \pi^j)$.

Iteration j: Iteration j, for $1 \leq j \leq h$ 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 $\mathbf{L}^j$ from this instance. Let $\mathcal{N}_j = N(\mathcal{I}^j, \pi^j)$ be the corresponding flow network. We will ensure that π^j is a *good* assignment for this instance. Note that we do not remove any items from the instance. So if instance $\mathcal{I}$ is 1-satisfiable, so is $\mathcal{I}^j$. Combined with the fact that there are no direct paths from s to $\mathbf{T}^j$, we can apply Theorem 3 to obtain two sets $\mathcal{P}_1$, $\mathcal{P}_2$ of elementary paths in $\mathcal{N}_j$, satisfying properties D1–D4.

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

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

Lemma 3 *We can find, in polynomial time, a set of internally-disjoint elementary paths $\mathcal{Q}^*$ in $N(\mathcal{I}, \pi^j)$ such that each agent A in $\mathbf{L}' \cup \mathbf{L}''$ has at least $\lfloor N_A/(\alpha_j + \alpha) \rfloor$ paths terminating at A. Moreover, only light agents in $(\mathbf{L}' \cup \mathbf{L}'') \setminus \mathbf{L}^j$ have paths in $\mathcal{Q}^*$ originating from them, with at most one path originating from any agent.*

Step 2: Re-Routing paths in $\mathcal{P}_1$. We say that elementary paths p, q intersect iff they either share intermediate vertices, or they start from the same vertex. Notice that each path in $\mathcal{P}_1$ may intersect many paths in $\mathcal{Q}^*$. In our next step, we re-route paths in $\mathcal{P}_1$ to get set $\mathcal{P}'$ of paths, so that each new path intersects at most one path in $\mathcal{Q}^*$.

Lemma 4 *We can find, in polynomial time, a set $\mathcal{P}'$ of disjoint paths, and a partition $\mathcal{Q}^* = \mathcal{Q}' \cup \mathcal{Q}''$, such that each path in $\mathcal{P}'$ starts at a distinct light agent in $\mathbf{L}'$ and ends at a distinct terminal in $\mathbf{T}^j$, and each terminal in $\mathbf{T}^j$ has one path terminating at it. Moreover, $|\mathcal{Q}''| \leq |\mathcal{P}'|$, and paths in $\mathcal{P}'$ do not intersect paths in $\mathcal{Q}'$.*

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

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

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

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

It is easy to see that a bad light agent can only be responsible for at most two terminals in $\mathbf{T}^{j+1}$, and removal of at most one path from $\mathcal{P}' \cup \mathcal{Q}'$. 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 now show how to produce the input to the next iteration.

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

Lemma 5 *The set* $\mathbf{S}$ *of items that are not assigned to any agent and the assignment of private items to agents in* $\mathbf{H}^*$ *remain unchanged throughout the algorithm.*

Since the original network $N(\mathcal{I}, \pi^1)$ did not contain any direct paths connecting s to terminals, Lemma 5 implies that this property is also true for $N(\mathcal{I}, \pi^j)$ for every j. Therefore we have produced a feasible input to iteration $(j+1)$. The lemma below bounds the size of $\mathbf{T}^{j+1}$.

Lemma 6 $|\mathbf{T}^{j+1}| \leq \left(\frac{32h^2\alpha}{n^\epsilon}\right) |\mathbf{T}^j|$.

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

Final Allocation: At the end of iteration $(h+1)$, we have an assignment, π^{h+1}, of private items to all agents apart from $\mathbf{L}^{h+1}$. Furthermore, in the network $N(\mathcal{I}, \pi^{h+1})$, the set $\mathcal{Q}^{h+1}$ of paths α_{h+1} satisfies $\mathbf{L}^{h+1}$. Thus, as in the proof of Lemma 2, we can use these paths to find the final allocation as follows. Each agent A that does not appear on any path in $\mathcal{Q}^{h+1}$ is assigned its private item. Agent A appearing on paths in $\mathcal{Q}^{h+1}$ is assigned all the items i such that (i, A) is an arc in one of the paths of $\mathcal{Q}^{h+1}$. Since these paths are internally disjoint, this allocation is feasible. Moreover, each heavy agent is satisfied and each light agent is α_{h+1}-satisfied.

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 $\alpha = O(h^5 \log n)$, the approximation factor from Theorem 3. Our algorithm assigns $N_A/(2h\alpha)$ items to each light agent A. So overall $\alpha^* = O(h\alpha) = O(h^6 \log n)$), yielding a $\max\left\{O(n^\epsilon \log n), O(h^6 \log n)\right\}$-approximation. When ϵ is chosen to be $(9 \log\log n)/\log n$, since $h = 9/\epsilon$, we get an $O(\log^{10} n)$-approximation algorithm. The algorithm in Theorem 3 runs in $n^{O(1/\epsilon)}$ time, and the algorithm presented in this section performs a poly-logarithmic number of calls to Theorem 3. Therefore, the overall running time of the algorithm is $n^{O(1/\epsilon)}$.

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

In this section, building on the $O(\log^{10} n)$-approximation algorithm of the preceding section, we show that it is possible to obtain an $O(m^\epsilon)$-approximation in quasi-polynomial time for any fixed $\epsilon > 0$. We start with the following easy lemma.

Lemma 7 *There exists an* $(\log n)^{O(m \log n)}$*-time* $O(1)$*-approximation algorithm for* Max-Min Allocation.

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

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

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

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

Using the above claim we can get the following.

Theorem 4 (Corollary 1) *For any constant $\epsilon > 0$, there exists a quasi-polynomial time algorithm which gives a $O(m^\epsilon)$-approximation to* MAX-MIN ALLOCATION.

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

6. THE 2-RESTRICTED MAX-MIN ALLOCATION PROBLEM

In this section we focus on the restricted version of MAX-MIN ALLOCATION, where each item i is wanted by at most 2 agents A_i and B_i (not necessarily distinct). Given a 2-restricted MAX-MIN ALLOCATION instance $\mathcal{I} = (\mathbf{A}, \mathbf{I})$, construct a graph $G(\mathcal{I}) = (\mathbf{A}, \mathbf{I})$ where an item i is an edge between agents A_i and B_i. Such a graph can have loops and parallel edges. Note that an allocation of items corresponds to the orientation of edges of this graph. The weights on the edges are non-uniform, that is, an edge has different weights for its two end points.

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

$$\forall i \in \mathbf{I}; \quad x_{A_i,i} + x_{B_i,i} \leq 1 \quad (1)$$

$$\forall A \in \mathbf{A}, \forall S \subseteq \delta(A); \quad \sum_{i \notin S} \hat{u}_{A,i} x_{A,i} \geq M - u_A(S) \quad (2)$$

where $u_A(S) := \sum_{i \in S} u_{A,i}$ and $\hat{u}_{A,i} := \min(u_{A,i}, M - u_A(S))$. The first inequality states that each item goes to at most one agent. The second states that, given any set of items S that give positive utility to agent A, the total utility of items *not in* S allocated to agent A must exceed $(M - u_A(S))$. Furthermore, this should also be true if the utility of any individual item is capped to $\hat{u}_{A,i}$ which is the minimum of $u_{A,i}$ and $(M - u_A(S))$. These type of valid inequalities are called *knapsack cover* inequalities (see, for instance, [9]).

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

Given a solution x, we now describe the set of inequalities that we need to check. We say that an item i is *integrally allocated* to agent A, if $x_{A,i} = 1$. For every agent $A \in \mathbf{A}$, let $\mathbf{I}(A)$ denote the set of items integrally allocated to A. An item i is said to be *fractionally allocated* if it is not allocated integrally to any agent. Let $\mathbf{I}'$ be the set of items allocated fractionally. For every agent A, let $\delta'(A)$ be the set of items fractionally allocated to it, that is, $\delta'(A) = \{i \in \mathbf{I}' : x_{A,i} > 0\}$. Let i^* be the item in $\delta'(A)$ with maximum value $u_{A,i}$. The inequality that needs to be checked for agent A is (2) with $S := (\mathbf{I}(A) \cup \delta'(A)) \setminus \{i^*\}$. From the above discussion, we can find in polynomial time a solution x which satisfies the constraints (1) of LP(M^*) and the following constraints

$$\forall A \in \mathbf{A}, \quad \sum_{i \notin S} \hat{u}_{A,i} x_{A,i} \geq M - u_A(S) \tag{3}$$
$$\text{for} \quad S := (\mathbf{I}(A) \cup \delta'(A)) \setminus \{i^*\}$$

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

Claim 1 *Fix an agent A. Then the total utility of the fractionally allocated items with a positive utility for agent A is at least $(M_A + u_{A,i^*})$, that is, $u_A(\delta'(A) \setminus \{i^*\}) \geq M_A$.*

Proof: Assume otherwise. Then we get for $S = (\mathbf{I}(A) \cup \delta'(A)) \setminus \{i^*\}$, $u_A(S) < M$. Inequality (3) is then contradicted, for the left hand side is precisely $\hat{u}_{A,i^*} x_{A,i^*}$ which is strictly less than $(M - u_A(S))$ if $M - u_A(S) > 0$, because by definition $x_{A,i^*} < 1$. ■

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

Theorem 5 *Given a non-uniformly weighted undirected graph $G(V, E)$ with weights $w_{u,e}$ and $w_{v,e}$ for every edge $e = (u, v) \in E$, there exists an orientation $\mathcal{O}$ such that in the resulting digraph, for every vertex v: $\sum_{e \in \Delta^-_{\mathcal{O}}(v)} w_{v,e} \geq \frac{1}{2}\left(\sum_{e \in \Delta(v)} w_{v,e} - \max_{e \in \Delta(v)} \{w_{v,e}\}\right)$.*

References

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

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

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

[4] M. H. Bateni, M. Charikar, and V. Guruswami. MaxMin Allocation via Degree Lower-Bounded Arborescences. *Proceedings of ACM Symposium on the Theory of Computation (STOC)*, pages 543–552, 2009.

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

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

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

[8] D. Chakrabarty, J. Chuzhoy and S. Khanna. On Allocating Goods to Maximize Fairness. Full version available at the authors' webpages.

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

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

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

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

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

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

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

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

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

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