New Approximation Algorithms for Degree Lower-bounded Arborescences and Max-Min Allocation

Technical Report TR-848-09

MOHAMMADHOSSEIN BATENI and MOSES CHARIKAR
Princeton University, Princeton, New Jersey
and
VENKATESAN GURUSWAMI
University of Washington, Seattle, Washington

We consider the problem of MaxMin allocation of indivisible goods. There are m items to be distributed among n players. Each player i has a nonnegative valuation p_{ij} for an item j, and the goal is to allocate items to players so as to maximize the minimum total valuation received by each player. There is a large gap in our understanding of this problem. The best known positive result is an $\tilde{O}(\sqrt{n})$-approximation algorithm, while there is only a factor 2 hardness known. Better algorithms are known for the restricted assignment case where each item has exactly one nonzero value for the players. We study the effect of bounded *degree* for items: each item has a nonzero value for at most D players. We show that essentially the case $D = 3$ is equivalent to the general case, and give a 4-approximation algorithm for $D = 2$.

The current algorithmic results for MaxMin Allocation are based on a complicated LP relaxation called the configuration LP. We present a much simpler LP which is equivalent in power to the configuration LP. We focus on a special case of MaxMin Allocation—a family of instances on which this LP has a polynomially large gap. The technical core of our result for this case comes from an algorithm for an interesting new optimization problem on directed graphs, MaxMinDegree Arborescence, where the goal is to produce an arborescence of large outdegree. We develop an n^{ϵ}-approximation for this problem that runs in $n^{O(1/\epsilon)}$ time and obtain a polylogarithmic approximation that runs in quasipolynomial time, using a lift-and-project inspired LP formulation. In fact, we show that our results imply a rounding algorithm for the relaxations obtained by t rounds of the Sherali-Adams hierarchy applied to a natural LP relaxation of the problem. Roughly speaking, the integrality gap of the relaxation obtained from t rounds of Sherali-Adams is at most $n^{1/t}$.

We are able to extend the latter result to a more general class of instances called InfDegreeTwo: each item has infinite utility for at most two players. Along the way, we prove a result about the existence of a perfect matching in a probabilistically pruned graph which may be of independent interest.

A short version of this paper appeared as an extended abstract in *Proceedings of the 41st ACM Symposium on Theory of Computing, ACM, New York, 2009.*
The works of M. Bateni and M. Charikar were supported by NSF ITR grants CCF-0205594 and CCF-0426582, NSF CAREER award CCF-0237113, MSPA-MCS award 0528414, and NSF expeditions award 0832797.
The work of M. Bateni was in addition supported by a Gordon Wu fellowship.
V. Guruswami is currently on leave visiting the Computer Science Department, Carnegie Mellon University. Some of this work was done when he was a member at the School of Mathematics, Institute for Advanced Study. His research was supported in part by a Packard Fellowship.
Authors' addresses: M. Bateni and M. Charikar, Princeton University, Department of Computer Science, 35 Olden Street, Princeton, NJ 08540, emails: {mbateni,moses}@cs.princeton.edu; V. Guruswami, University of Washington, Department of Computer Science and Engineering, Seattle, WA 98195, email: venkat@cs.washington.edu.

Categories and Subject Descriptors: F.2.2 [**ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY**]: Nonnumerical Algorithms and Problems—*Computation on Discrete Structures*; G.1.6 [**DISCRETE MATHEMATICS**]: Graph Theory—*Graph algorithms*

General Terms: Algorithms, Economics, Theory

Additional Key Words and Phrases: Approximation algorithm, fair allocation, hardness of approximation, LP relaxation, Sherali-Adams hierarchy

1. INTRODUCTION

We study the MaxMin allocation problem of indivisible goods. An instance of MaxMin Allocation consists of a set A of n players and a set B of m items, with a non-negative utility value p_{ij} for each player $i \in A$ and item $j \in B$. The utility of a player is an additive function, i.e., if player i is given a subset $B_i \subseteq B$ of items, she gets a utility $\sum_{j \in B_i} p_{ij}$. The goal is to find an allocation of items to players—$B_i \subseteq B$ for each player i, where $B_i \cap B_{i'} = \emptyset$ for players $i \neq i'$—such that the minimum of the utilities of the players, i.e., $\min_{i \in A} \sum_{j \in B_i} p_{ij}$, is maximized. The problem corresponding to the dual objective, i.e., minimizing $\max_{i \in A} \sum_{j \in B_i} p_{ij}$, is the classic Makespan minimization problem for unrelated parallel machines (with items corresponding to jobs and players corresponding to machines). In this work, we consider the MaxMin objective which is natural if we think of items as rewards and look for an equitable, fair allocation of them so that every player surpasses a certain "happiness" threshold. Accordingly, this problem has also been called the "Santa Claus" problem [Bansal and Sviridenko 2006; Asadpour et al. 2008]. The MaxMin Allocation problem has received a fair bit of attention lately [Bansal and Sviridenko 2006; Asadpour et al. 2008; Asadpour and Saberi 2007; Bezáková and Dani 2005; Woeginger 1997; Golovin 2005; Feige 2008; Khot and Ponnuswami 2007]. We discuss some of the previous work next before mentioning the contributions of this paper.

1.1 Previous work

Lenstra et al. [1990] give a factor 2 approximation algorithm for the MinMax version, and also show a factor $\frac{3}{2}$ inapproximability result. Closing this gap has been a longstanding open problem. Despite much attention recently, there is a large gap in our understanding of the approximability of the dual objective, MaxMin. The best known algorithm achieves an $\tilde{O}(\sqrt{n})$ approximation ratio in the general case, but only a factor 2 hardness result is known. The objective function for MaxMin is rather fragile, since a small mistake in the allocation might starve a player completely, and leading to a very poor approximation ratio. This in some sense captures the high level difficulty faced by algorithms for the MaxMin objective.

Bezáková and Dani [2005] achieve an additive approximation for the MaxMin Allocation using the natural LP formulation, guaranteeing a value of at least $\mathrm{OPT} - \max_{ij} p_{ij}$. If the maximum value of some items are close to the optimum, then this could be a poor guarantee. Besides, they modify the hardness proof of [Lenstra et al. 1990] to obtain a factor 2 inapproximability result for MaxMin Allocation.

Various special cases of the problem have also been studied. The uniform case, when each item has the same value for every player, i.e., $p_{ij} = p_j$ for all i, admits a PTAS [Woeginger 1997]. Subsequent works, ending in [Alon et al. 1998], achieve

PTAS for the same setting while considering more general objective functions.

Another important special case has been the *restricted assignment* case considered in [Bansal and Sviridenko 2006; Feige 2008; Asadpour et al. 2008], where each item j has some intrinsic value p_j and a player is interested in a subset of items; more formally, $p_{ij} \in \{p_j, 0\}$ for all i, j. Bansal and Sviridenko [2006] introduce a new linear programming relaxation called the *Configuration LP* to study the MaxMin Allocation problem, and by rounding this LP are they able to achieve an $O(\log\log n/\log\log\log n)$-approximation for the restricted assignment case. Feige [2008] gave a non-constructive proof that the configuration LP has $O(1)$ integrality gap for this special case. Subsequently, Asadpour, et al. [2008] gave a simpler exponential time 5-approximation algorithm, proving the integrality gap is no more than 5.

Golovin [2005] presents an $O(\sqrt{m})$-approximation for the so-called *big goods/small goods* case, which is a further special case of the *restricted assignment* case where item values need to be either 1 or ∞. For the general problem, Bansal and Sviridenko [2006] exhibit an example with integrality gap $\Omega(\sqrt{n})$ for the configuration LP. Asadpour and Saberi [2007] show that this is essentially tight, by giving an algorithm to round it with an $O(\sqrt{n}\log^3 n)$ guarantee. This is the best currently known guarantee for the general MaxMin Allocation problem. The best known hardness result remains a factor of 2, which holds even for the restricted assignment case.

Khot and Ponnuswami [2007] also consider a special case where $p_{ij} \in \{0, 1, \infty\}$ for all i, j—this is believed to be as hard as the general case. They establish a tradeoff between runtime and approximation ratio. Specifically, they give an $\frac{n}{\alpha}$-approximation algorithm which runs in time $m^{O(1)}n^{O(\alpha)}$, for any given $\alpha \leq n/2$. They also give a $(2n-1)$-approximation algorithm for a generalization of MaxMin Allocation where utility functions are subadditive.

Ebenlendr et al. [2008] recently revisited the MinMax objective function in the case when each item has nonzero value for (at most) two players, and further these values are the same for the two players (called the "symmetric" version). This problem can be equivalently viewed as a *Graph Balancing* problem, where one has to orient edges of a weighted undirected graph so that the maximum weighted in-degree is minimized. They are able to improve the approximation factor from 2 down to 1.75, while proving the same 1.5 hardness. In light of this, it is natural to study the effect of bounded degree items in the MaxMin setting.

Very recently, it was brought to our attention that Chakrabarty et al. [2009] study the MaxMin Allocation problem and give an $O(m^{\epsilon})$-approximation algorithm that runs in time $O(m^{\frac{1}{\epsilon}})$ for the general case. As a result they can achieve a polylogarithmic approximation in quasipolynomial time. As explained below, we are able to obtain the same guarantee for an interesting special case which is a natural intermediate point towards solving the general case. Though our works were independent, there are some parallels between our approaches. They also give a $(2+\epsilon)$-approximation algorithm for an instance of MaxMin Allocation that we will call DegreeTwo. In such instances, each item has nonzero utility for at most two players.

1.2 Contributions

We consider the instances in which the *degrees* of items are limited. Degree of an item j is defined as the number of players competing for it, i.e., $|\{i \in A : p_{ij} > 0\}|$. In Sections 4 and 5 for the ease of demonstration, we focus on instances where $\forall i, j : p_{ij} \in \{0, 1, \infty\}$. The *infinity degree* of an item j is the number of players drawing infinite utility from the item, i.e., $|\{i \in A : p_{ij} = \infty\}|$. We then define instance classes DegreeTwo, DegreeThree, InfDegreeOne and InfDegreeTwo as follows. DegreeTwo and DegreeThree denote the classes of instances in which the degree of an item is at most two and three, respectively. Similarly, InfDegreeOne and InfDegreeTwo denote the classes of instances in which the infinity degree of an item is at most one and two, respectively.

An instance of MaxMin Allocation can be conveniently represented using a weighted (bipartite) graph $G(V, E)$. The set V of vertices consists of players and items. There is an edge of value p_{ij} between player i and item j. An edge of value infinity is called an *infinity* edge. Any connected component of the subgraph induced by the infinity edges is called an *infinity component*. We do not consider the trivial components having no edges. Infinity components have two categories: *simple* infinity components have exactly one edge, and *complex* infinity components which have more than one edge. InfDegreeOne consists only of simple infinity components.

We show that any instance of MaxMin Allocation can be transformed into one of DegreeThree with only a polynomial size growth. This also holds for the MinMax objective. Next, we give a 4-approximation algorithm for DegreeTwo. We stress that our algorithm works in the *asymmetric* case when the item can have distinct nonzero utility values for the two players interested in it. This is the only positive result for the asymmetric case besides the $\tilde{O}(\sqrt{n})$-approximation [Asadpour and Saberi 2007] for the general MaxMin Allocation. We also show a modification of the hardness result in [Bezáková and Dani 2005] gives a factor 2 inapproximability of DegreeTwo, even for the symmetric case. We show the same construction essentially gives a similar hardness for InfDegreeOne. We should note that, prior to our work, the best approximation factor for the symmetric DegreeTwo case prior to our work is also only 4. This follows from our result, but can also be obtained by customizing the algorithm in [Bansal and Sviridenko 2006].

We also present a new LP relaxation for the MaxMin Allocation problem. We show that this is equivalent in power to the configuration LP, while being simpler and having a compact formulation. (Thus it can be solved directly, unlike the complicated dual Knapsack methods required to solve the configuration LP.)

We focus on an interesting special case of the MaxMin Allocation problem, i.e., InfDegreeOne, for which the LP formulations, prior to our work, have polynomial integrality gaps. For these instances, we devise a polylogarithmic approximation that runs in quasipolynomial time (and an $O(m^{\epsilon})$-approximation in time $m^{O(1/\epsilon)}$). The technical core of this result is the development of an algorithm for an interesting new optimization problem on digraphs: MaxMinDegree Arborescence. Given a directed graph with designated sources and sinks, the problem asks us to produce a collection of disjoint arborescences T_i, one rooted at each source such that every non-sink vertex in T_i has out-degree M; the goal is to maximize M. While the problem formulation seems similar to degree bounded Steiner tree problems that

have received much attention recently (see, for instance, [Chan et al. 2008] and the references therein), the nature of the problem is quite different and requires the development of new techniques. Our algorithm uses a lift-and-project inspired LP relaxation with a dependent rounding procedure that exploits the additional variables and constraints. In fact, we show that the LP we use can be obtained by t rounds of the Sherali-Adams hierarchy applied to a basic LP relaxation for the problem. Our techniques can be interpreted as a rounding algorithm for the relaxation obtained after t rounds of Sherali-Adams. They imply an interesting approximation-rounds tradeoff: the relaxation after t rounds has integrality gap at most $\max\{\text{poly}\log(m), m^{O(1/t)}\}$. This is especially interesting as there has been a lot of recent interest in understanding lift-and-project procedures and very few positive results using lift-and-project are known. The rounding algorithm itself is similar to a rounding algorithm used for the *group Steiner tree* problem. However, our analysis is quite different and requires bounding higher moments for a dependent random process on a tree.

Next, in an attempt to pull through the result for the general case, we extend our results to more general instances, i.e., NoInfCycle and InfDegreeTwo. The class of instances NoInfCycle are those whose graphic representation does not contain a cycle consisting merely of infinity edges. Here, we have to cope with the subtleties corresponding to the handling of the infinity components. Along the way, we prove a result about existence of a perfect matching in a probabilistically pruned graph. We show that given a fractional matching LP solution for a bipartite graph in which vertices of the first partition can have positive *deficiency*, throwing away vertices with probabilities proportional to their deficiencies leaves us with a graph admitting a perfect matching. Another nuance is dealing with the event of assigning items in an infinity component to players outside the component. Players in the infinity component may no longer be satisfiable with the remaining items from the component and this may trigger some of them to seek small items from outside. This is the main point of departure from our InfDegreeOne algorithm, since these effects may need to propagate along long chains. The bounded length of *reaction chains* is crucial in our InfDegreeOne algorithm. Thus to extend the results to InfDegreeTwo, we need to deal with this very issue. We finally show that InfDegreeTwo instances are similar to NoInfCycle. Were we able to eliminate the *infinity cycles* in some way, we could derive the same result for the general case of MaxMin Allocation, matching the results of [Chakrabarty et al. 2009].

1.3 Organization

The new LP relaxation as well as the universality of degree three instances is explained in Section 2. In addition, we show our simpler LP formulation is essentially as powerful as the current champion, i.e., the Configuration LP. In Section 3, we give a 4-approximation algorithm for DegreeTwo. Then in Section 4, the machinery for solving InfDegreeOne is introduced. We next solve NoInfCycle in Section 5 and show how this implies a similar result for InfDegreeTwo. After that, we illustrate in Section 6 the connection between our linear program of Section 4 and the Sherali-Adams hierarchy. Before concluding the paper, we discuss a barrier for showing better hardness results in Section 7.

2. PRELIMINARIES

Recall that we are given a set A of players and a set B items and p_{ij} represents the utility of item j for player i. Such an instance is usually considered as a weighted bipartite graph that has players on one side and items on the other. In figures, we will use squares to represent players and circles to depict items.

The natural LP for the problem has an unbounded integrality gap. A more powerful LP, called the *Configuration LP*, was introduced in [Bansal and Sviridenko 2006], and has been crucially used in [Bansal and Sviridenko 2006; Asadpour and Saberi 2007; Feige 2008; Asadpour et al. 2008].

Definition 2.1 *Config-LP.* There is a variable x_{iC} for each player i and each valid configuration (aka bundle) C of items. A bundle $C \subseteq B$ is called *valid* for i if and only if $\sum_{j \in C} p_{ij} \geq M$. Let $\mathcal{C}_i$ denote the set of valid bundles for player i. Then, the C-LP relaxation is as follows.

(C-LP)

$$\sum_{i \in A} \sum_{\substack{C \in \mathcal{C}_i \\ C \ni j}} x_{iC} \leq 1 \qquad \forall j \in B \tag{1}$$

$$\sum_{C \in \mathcal{C}_i} x_{iC} = 1 \qquad \forall i \in A \tag{2}$$

$$x_{iC} \geq 0 \qquad \forall i \in A, C \in \mathcal{C}_i. \tag{3}$$

This LP has exponential size, but as noted in [Bansal and Sviridenko 2006], the separation oracle needed for the dual is the knapsack problem. So, we can find an approximate value (with arbitrary fixed precision) for this LP in polynomial time. Although the enhancements of C-LP eliminates certain bad examples for the natural LP, the integrality gap still remains as large as $\Theta(\sqrt{n})$ [Bansal and Sviridenko 2006].

Note that the LPs studied for this problem are used for feasibility of a guessed value M. When dealing with Config-LP, we usually have a threshold $\tau = M/\lambda$ and we revalue any item of value no less than τ to M. This way, we might increase the value of the solution by a factor of λ, but we create a gap between "small" and "big" items. One big item is sufficient to satisfy a player and thus we assume a configuration is either "big" (i.e., it has only one big item) or is "small" which means the configuration is comprised of multiple (at least λ) small items in it. Not present in Config-LP but implied are the natural variables $x_{ij} = \sum_{C \ni j} x_{iC}$, which show to what extent an item j is assigned to a player i.

2.1 A simpler LP

Here, we introduce a simplified linear program with which we work. In sharp contrast to Config-LP, it has only polynomially many variables and constraints. Besides making arguments simpler, this gives room to enhance the LP with additional constraints as we do in Sections 4 and 5.

Definition 2.2 *M-LP.* There is a variable x_{ij} for each player i and each item j. Items are denoted *small* or *big* depending on how their value compares to M/λ. Furthermore, z_i, called the *small usage* of player i, indicates how much small items

contribute to the utility of this player. Note that the notion of small/big is with respect to a specific player. Here is the relaxation M-LP:

(M-LP)

$$\sum_{i \in A} x_{ij} \leq 1 \qquad \forall j \in B, \tag{4}$$

$$\sum_{\substack{j \in B \\ p_{ij}=M}} x_{ij} + z_i \geq 1 \qquad \forall i \in A, \tag{5}$$

$$\sum_{\substack{j \in B \\ p_{ij}<M}} p_{ij} x_{ij} \geq z_i M \qquad \forall i \in A, \tag{6}$$

$$x_{ij} \leq z_i \qquad \forall i \in A, j \in B : p_{ij} < M \tag{7}$$

$$x_{ij}, z_i \geq 0 \qquad \forall i \in A, j \in B. \tag{8}$$

Removing constraints (7) gives the natural LP to which we refer by S-LP. Clearly, any Config-LP solution can be specified as a solution to M-LP; one just needs to use the implied x_{ij} values and appropriate values for $z_i := \frac{1}{M} \sum_{j \in B: p_{ij} < M} p_{ij} x_{ij}$. We claim that the algorithms and analyses of previous works using C-LP can be tailored to use M-LP values instead. Moreover, any solution to M-LP can be turned into a solution of C-LP of roughly the same value; see Theorem 2.3. We do not know how to use M-LP (without losing the extra factor two) to get the recent tight result of [Chakrabarty et al. 2009] for DegreeTwo.

THEOREM 2.3. *An M-LP solution of value M can be translated in polynomial time to a Config-LP solution of value $M/2$.*

PROOF. The proof is constructive. First revalue any item of value $\geq M/2$ to M. Consequently, decrease the value of z_i by $\sum_{j: p_{ij} \geq M/2} x_{ij}$, and then decrease the x_{ij} to $\min(z_i, x_{ij})$ if $p_{ij} < M/2$. It can be easily verified that the new values show a valid M-LP solution and each small item has value at most $p^{\max} \leq M/2$.

We leave the assignment of big items unaffected. Take any player i and its corresponding small items. Ideally, they should build z_i units of small configurations of value M. We show that, compromising on the value of configurations to be $M - p^{\max}$, we can achieve this task. We restate the set of constraints we need to work on, and remove unnecessary indices.

$$\sum_j p_j x_j \geq z\, M \tag{9}$$

$$x_j \leq z \qquad \forall j \tag{10}$$

$$x_j \geq 0 \qquad \forall j. \tag{11}$$

The algorithm proceeds by creating bundles and reducing x_j's and z accordingly. At any step, it maintains that the current values of x_j and z are feasible in the above set of equations 9-11; if z^* and x^*_j are the initial values, we have made at least $z^* - z$ units of bundles where each item j has been used for at most $x^*_j - x_j$ units. At each step, we rename items so that x_j's are nonincreasing. We know from (9) and (10) that the sum of the values of the remaining items is at least M; since, otherwise, $\sum_j p_j x_j \leq \sum_j p_j \sum_{j'} x_{j'} < z\, M$ leading to a contradiction. Let q be the

smallest index such that $\sum_{j \le q} p_j \ge M$. We will do one of the following until $z = 0$, in which case we are done.

- If $x_q \neq x_1$, make a bundle out of the first $q-1$ items. Use this bundle for $\delta = \min\{x_{q-1}, x_1 - x_q\}$ units. Decrease concerning z and the involved x_j variables by δ. Notice that $\delta > 0$, since $x_{q-1} \ge x_q > 0$ and $x_1 \neq x_q$. The value of this bundle is at least $M - p_{\max}$ and at most M. So the left-hand side of Inequality (9) is reduced by at most δM which is the decrease in the right-hand side. By the choice of δ, the maximum x_j after this operation is at most $x_1 - \delta \le z - \delta$. Hence, Inequality (10) is still valid after this operation.
- If $x_q = x_1$, we say we have a *plateau*. Let q' be the largest index where $x_1 = x_{q'}$. In this case, $q' \ge q$ and we cannot simply make a bundle out of the first $q-1$ items, because then the maximum x_j would not change, whereas this change may be necessary to keep (10) satisfied. The trick is to decrease x_j for all variables equal to x_1 simultaneously.
  To this end, we build a *balanced* collection of bundles which in total contains each item of our concern (i.e., all items j such that $x_j = x_1$) exactly r times. The size of each bundle is at most M. We then decrease the corresponding x_j's by $\delta = x_1 - x_{q'+1}$; i.e., we use each configuration for δ/r units. We let $z = \frac{1}{M}\sum_j p_j x_j$ after this operation. It is easy to see that we make at least $\frac{1}{M}\sum_{j \le q'} p_j \delta \ge z^{\text{old}} - z^{\text{new}}$ units of configuration in this step. The solution is thus feasible since $z \ge \frac{1}{M}\sum_{j \le q'} p_j x_1 \ge x_1$ and x_1 is the maximum x_j value.
  Building the balanced collection of bundles is done as follows. Take the group of items attaining the same maximum value. We know that their sum of values is at least M (by the choice of q). Start from the first one and include items until we make a bundle of value at least $M - p_{\max}$. Then, start from the next element and do the same thing (possibly wrapping around), and so on. We repeat this bundle-making process until a bundle is repeated. This has to happen because there would be at most m different bundles formed in such a process. Then, we can take the collection of bundles from the first occurrence of this bundle up until (but not including) its second occurrence. Clearly, each item appears the same number of times in these bundles. An example is shown in Figure 1.

Performing this two-stage procedure, we can form a Config-LP solution from any M-LP solution, provided that we allow configuration values to be as low as $M - p_{\max}$. Finiteness follows from the fact that at each step, either some variables go zero, or the number of variables attaining the maximum goes up. □

So, there will be at most $O(m)$ steps and each step adds at most m bundles. Thus, the total number of bundles produced for each player is bounded by $O(m^2)$ which is a polynomial.

2.2 Degree reduction

We show that any instance of the problem with n players and m items can be converted into an instance of DegreeThree.

THEOREM 2.4 DEGREE REDUCTION. *Any instance of MaxMin*[1] *with n players*

[1]A similar reduction can be done for the MinMax objective, by replacing ∞ with M (our guess)

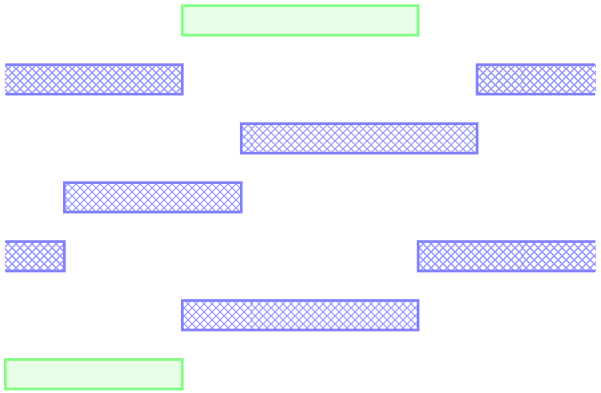


Fig. 1. Making bundles in case of a plateau: items are put in a line, and we start from the left and find maximal bundles of value at most M. Each row in the picture shows one bundle. Note how each bundle starts after the previous one, and how we wrap around at the end. Plain (green) bundles (first and last ones) are discarded. In this example, we end up with 5 bundles and each item is covered exactly twice.

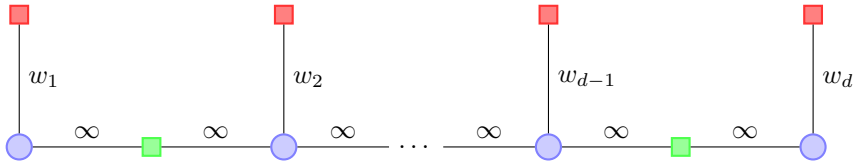


Fig. 2. Degree reduction for an item. An item which has degree $d > 3$, can be replaced by a gadget as shown here. The upper row players are the ones which are interested in this item, possibly with different valuations $w_1, w_2, \ldots, w_{d-1}, w_d$. The bottom-row players and items are virtual ones. One can observe that at most one of the original links can be used.

and m items can be changed into an equivalent DegreeThree *instance with mn players and mn items.*

Proof. Take any item of degree d, and replace it by a gadget similar to Figure 2; i.e., make d copies of the item, one for each of its takers (with the associated value), and also add $d-1$ dummy players, one between any two consecutive copies (with value ∞). Each of the dummy players needs to get hold of one of the item copies to satisfy its demand. Exactly one copy will be free to serve its real player. □

There is a caveat here. The number of players in the DegreeThree instance may be significantly larger than the original instance, and this can affect the approximation guarantee. Especially, this may be more evident for instances where the number of items and players substantially differs. However, as long as polylogarithmic approximation ratios are sought, the degree three instance is equivalent to the original.

---

and noting that we want approximation guarantees better than two.

3. DEGREETWO INSTANCES

Knowing that DegreeThree can model the general case of the problem motivates the study of the DegreeTwo special case. Customizing the algorithm of [Bansal and Sviridenko 2006], one can obtain a 4-approximation for the symmetric degree two case. Also, implicit in [Feige 2008; Bansal and Sviridenko 2006] is that the constant degree symmetric case has a constant factor approximation algorithm. More specifically, a restricted assignment instance with items having degree at most $d > 2$, admits a $4d$-approximation algorithm. Yet, in the presence of asymmetry, nothing better than the general $\tilde{O}(\sqrt{n})$-approximation algorithm of [Asadpour and Saberi 2007] was known prior to this work.

We present a factor 2 hardness result for this special case. This is the same as the best-known hardness for the general case.

THEOREM 3.1 DEGREETWO HARDNESS. *DegreeTwo cannot be approximated to within a factor better than 2 unless P = NP. This also holds if we restrict our instances to be symmetric.*

PROOF. The reduction is from a special 3SAT where each literal appears at most twice. Let the 3SAT instance have variables $v_1, \ldots, v_n$ and clauses $\phi_1, \ldots, \phi_m$. In our MaxMin Allocation instance, we have two players for each variable, one corresponding to v_t and one for $\overline{v_t}$. There is an item of value two shared between them. Each clause has an exclusive item of value 1 and one item shared with any of its literals, with value 1. If some literal occurs l times in our formula, we add $2 - l$ exclusive items of value 1 for it. We claim that the value of the instance is two if and only if the given 3SAT instance is satisfiable.

If it is satisfiable, let the value 2 items be given to the true literals. Each false literal takes its other items to have utility two. Any clause has an exclusive item and at least one shared item, which give it a utility of at least two.

Now, if we have a solution of value at least two, we let the literals which receive big items to be true. False literals take in all their small items. Each clause receives some shared item other besides its exclusive item. So, that should correspond to a true literal.

If the value of the instance is not two, it is at most one, and hence comes the gap of two. □

In Section 7, we discuss a common element in all current hardness reductions of the problem; this is a barrier for proving hardness beyond factor 2.

To solve an instance of DegreeTwo, we first solve the corresponding M-LP (with big items rounded with $\lambda = 4$). Then, we massage the fractional solution to simplify its structure. Finally, we do the rounding. In what follows, we describe in more details the steps of our algorithm which is presented in Figure 3.

3.1 Restructuring the LP solution

Assume we are given a solution $\{x^*_{ij}, z^*_i\}$ to the M-LP, where all constraints (6) are tight. Otherwise, we first decrease x^*_{ij} values for small items appropriately to obtain this property. We build the new solution x_{ij} as follows. We change all the nonintegral values to 1/2. Next if any small variable has a value larger than 1/2, we decrease it to 1/2. Then, a player receiving any small item forgoes all its big

items. Finally, set $z_i = \frac{2}{M} \sum_{j \in B: p_{ij} < M} p_{ij} x_{ij}$ for any player i. The restructured solution has the following properties.

(SP1) Value of a small variable x_{ij} is either zero or $1/2$;

(SP2) Each player uses either big items or small ones; and

(SP3) The restructured solution, x_{ij}, is a valid M-LP solution of value $M/2$.

The truth of (SP1) and (SP2) is immediate. For the last property, note that we have a half-integral solution. There are at most two nonzero variables for each item. If a constraint (4) is violated, then at least one of them has to be one. However, this means that the other one is zero, since it was so in the previous solution. The constraints (6) hold by definition. For the constraints (7), observe that $x_{ij} \leq 1/2$ for small items. If any such constraint is violated, it should be a $z_i < 1/2$ and some $x_{ij} = 1/2$. So in the original solution, x^*_{ij} and thus z^*_i are both positive. Constraints (7) and (6) imply that there are at least M different positive x^*_{ij} for this player i. All these variables have a value $1/2$ in our new solution. Hence $z_i = 1$. This also shows that for a nonzero z^*_i, we have $z_i = 1$ in the new solution, which ensures constraint (5). Now consider constraint (5) for a player with $z^*_i = 0$. Either the player got only one big item or more. In the former case, we do not change the variable value. In the latter, there are at least two nonzero values, each $1/2$, which yield the constraint.

3.2 Rounding the fractional solution

Having the above properties in our fractional solution, we are ready to produce an integer assignment. We are going to use the following result of [Bezáková and Dani 2005] which is very similar in nature to the main result in [Lenstra et al. 1990].

THEOREM 3.2 ALGORITHM LP-ROUND [BEZÁKOVÁ AND DANI 2005]. *Given a solution to S-LP of value M where the maximum size of an item used is $p^{\max}$, we can round the LP using the algorithm* LP-ROUND *to get a solution of value at least $M - p^{\max}$.*

Although the algorithm cannot give any guarantee when items are large compared to the solution value, it will prove useful in our algorithm.

We build a graph on players and items, where an edge connecting player i and items j has weight x_{ij}. Such an edge is called *big* if item j is big for player i; otherwise, we call it *small*. For now, we ignore the small edges and any player who uses them. Remember that by (SP1)-(SP3), any such player derives enough profit from small items so as not to need any big items. We can perform rotations as in [Asadpour and Saberi 2007] to eliminate cycles in the graph of big edges. Roughly speaking, we pick a small ϵ and increase the weights of even-indexed edges in the cycle by ϵ and decrease those of the odd-indexed edges by the same amount. We do this for an appropriate ϵ such that the weight of at least one edge drops to zero, when we can remove it. Notice that if an edge gets weight one, in which case, we should also stop, the adjacent edge will have weight zero. Each component of the remaining graph will be a tree T_t. No leaf of a tree T_t can be a player, since otherwise, that would be a player with no small items utility and with only one big item.

Algorithm DegreeTwo Solver
Input: *An instance of DegreeTwo.*
Output: *An assignment of value at least OPT/4.*

(1) Solve M-LP by repeated guessing to get $\{x^*_{ij}, z^*_i\}$. When the guess value is M, revalue any item bigger than $M/4$ to M.
(2) Build x_{ij} as follows:
  (a) decrease x^*_{ij} variables as much as possible without violating the constraints;
  (b) let $x_{ij} = x^*_{ij}$;
  (c) change all the nonintegral x_{ij} to 1/2;
  (d) make sure no $x_{ij} > 1/2$ for small item j; and
  (e) make $x_{ij} = 0$ if item j is big for player i but player i has a nonzero $x_{i,j'}$ for a small item j'.
(3) Do rotations on big edges to eliminate cycles, and identify trees T_t to construct the modified instance.
(4) Round the fractional solution using LP-Round.
(5) Deal with the assignment of big items in each tree T_t.

Fig. 3. The algorithm for DegreeTwo.

We assume, without loss of generality, that any non-leaf item in T_t contributes one unit to the players in the tree. Note that because we have a DegreeTwo instance, no other player can be interested in any such item. If we pick an arbitrary leaf j of T_t and root the tree from it, we can give any item to its child in the tree (because they have degree two unless they are leaves) to satisfy all the players in T_t. All the other leaves will be free. Any solution needs to use exactly one out of the k_t leaf items of each T_t. The usage of small items is at most 1/2. So, the LP usage of small items in T_t from the outside is at most $k_t/2$.

Now, replace each tree having $k_t \geq 2$ leaves with $\lceil k_t/2 \rceil \leq k_t - 1$ items j_{tl} for $l = 1, \ldots, \lceil k_t/2 \rceil$. These are meant to play the role of the original $k_t - 1$ remaining items. Name the original leaf items in T_t, by $j'_{t,1}, \ldots, j'_{t,k_t}$. Each of them, say j'_{tl}, has exactly one taker outside, say i_{tl}; we know that the usage is exactly 1/2. The case there is no such player is trivial. Now, we connect j_{tl} to $i_{t,2l-1}$ and $i_{t,2l}$ with the corresponding values they had for $j'_{t,2l-1}$ and $j'_{t,2l}$. The last item $j_{t,\lceil k_t/2 \rceil}$ might get only one player. We let the weights of all these edges be 1/2. There is a minor technicality here if two players requesting some j_{tl} are the same. Then, there should be a single edge of weight one whose value is the larger of the two values the player has for the two items.

Now, ignoring the big players altogether, we have a feasible S-LP solution of value $M/2$, and we can round it to $M/2 - M/\lambda$ using Theorem 3.2. After rounding, we know that at most $\lceil k_t/2 \rceil \leq k_t - 1$ items from T_t have been used and so the remaining one can help satisfy the players in T_t with value at least $\tau = M/\lambda$. Choice of $\lambda = 4$ leads to a 4-approximation algorithm for the problem.

4. INFDEGREEONE INSTANCES

From here onwards, we will be focusing on instances in which utilities come from $\{0, 1, \infty\}$. This is assumed mostly for simplicity of presentation, however, once we guess the value of the solution, say M, we revalue any item larger than $\tau = M/\lambda$ to ∞ creating a multiplicative gap of λ between *small* utilities and M. Then, the

algorithms and analysis can be modified for the argument to go through provided that the resulting instance conforms with the requirements of the algorithm (i.e., it is InfDegreeOne or InfDegreeTwo).

If a player i has utility 1 for item j, we say that j is a *small* item for i. If a player i has utility ∞ for item j, we say that j is a *large* item for i. In this section, we study InfDegreeOne instances. As we will show, the LP formulation used in the currently best known results for MaxMin Allocation has integrality gap of $\Omega(n^{1/6})$ for such instances.[2] We design an algorithm that achieves an approximation ratio of m^{ϵ} in time $m^{O(1/\epsilon)}$ and a polylogarithmic approximation in $m^{O(\log m)}$ time.

In order to do this, we strengthen the LP formulation for MaxMin Allocation; here our simplified M-LP formulation lends itself to incorporating additional constraints. We begin with pointing out the limitations of current LP formulations for these restricted instances.

LEMMA 4.1. *M-LP (and similarly Config-LP) has a gap of $\Omega(n^{1/6})$ for InfDegreeOne.*

PROOF. Consider the gadget H consisting of $T+1$ players and $2T$ items. Player M_0 is connected to item $J_1, \ldots, J_T$ and has value 1 for each of them. For $1 \leq t \leq T$, item J_t is big (value infinity) for M_t. For each $1 \leq t \leq T$, player M_t has value 1 for $J_{T+1}, \ldots, J_{2T}$. It is easy to note that there's a fractional solution having value 1 ($1/T$ units of small usage) for M_0 and value T for everybody else. In an integral solution, however, if M_0 claims any αT items of his, the corresponding αT players will have only T small items available for them in total. So, the integral solution of the gadget would be at most $\sqrt{T}$.

Let's call the player M_0 of H its distinguished player. Have T^2 copies of H. Call their distinguished players $A_1, \ldots, A_{T^2}$. Attach an item B_t to any of them with value infinity. There is also one player C which has value one for any of these new items B_t. This player can fractionally use $1/T$ from any of these items and so, each A_t needs only $1/T$ from inside its gadget. So, there is a fractional solution of value T. In an integral solution, C needs several of his items. Suppose he claims B_1 along with several others. Then, A_1 needs to be satisfied internally, which gives a solution of value at most $\sqrt{T}$. The size of the instance is $O(T^3)$. The gap is thus $\Omega(n^{1/6})$. □

We further show a hardness result for these instances similar to the previous ones. In fact, we prove this for instances in the intersection of InfDegreeOne and DegreeTwo.

THEOREM 4.2. *InfDegreeOne cannot be approximated to within better than a factor two unless P = NP.*

PROOF. The proof is similar to that of Theorem 3.1. The reduction is from a special 3SAT where each literal appears at most twice. Let the 3SAT instance have variables $v_1, \ldots, v_n$ and clauses $\phi_1, \ldots, \phi_m$. In our MaxMin Allocation instance, we have three players for each variable, one corresponding to v_t and one for $\overline{v_t}$; the third one is a dummy. There are two items of value infinity, each shared between the dummy and one of the literals. These items have value one for the dummy,

[2] A similar argument gives a gap of $\Omega(n^{1/4})$ for InfDegreeTwo.

which has another exclusive item of value one. Each clause has an exclusive item of value 1 and one item shared with any of its literals, with value 1 on both sides. If some literal occurs l times in our formula, we add $2-l$ exclusive items of value 1 for it. We claim that the value of the instance is two if and only if the given 3SAT instance is satisfiable.

If it is satisfiable, let the true literals receive their infinity edges. The dummy player of each variable takes its exclusive item and the infinity item of the false literal. Each false literal takes its noninfinity items to have utility two. Any clause has an exclusive item and at least one item shared with a true literal, which give it a utility of at least two.

Now, suppose we have a solution of value at least two. Notice that the two literals corresponding to a variable cannot both claim their infinity item. Thus, we let the literals which receive infinity items to be true. False literals take in all their noninfinity items. Each clause is given some item other than its exclusive item. So, that should correspond to a true literal.

If the value of the instance is not two, it is at most one, and hence comes the gap of two. □

4.1 The MaxMinDegree Arborescence problem

Our main technical contribution in this section is a solution to a natural new optimization problem on directed graphs which we introduce in this section and relate to the InfDegreeOne problem. In a digraph, any vertex with no incoming edges is called a *source*. Similarly, a vertex with no outgoing edges is called a *sink*.

Definition 4.3. An *M-pyramid*[3] of a digraph G is a subgraph of G with the following properties:

(1) it is an arborescence, i.e., the in-degree of every vertex is at most 1;
(2) it contains all the sources in G;
(3) any sink of the arborescence is also a sink in G; and
(4) the out-degree of any non-sink vertex is at least M.

Note that an M-pyramid of G need not contain all the vertices of G. Technically, an M-pyramid is a forest of arborescences, but for convenience, we simply say arborescence.

Definition 4.4. In an instance of MaxMinDegree Arborescence problem, we are given a digraph G, and the goal is to find an M-pyramid of G that maximizes M.

We now show how to reduce MaxMin Allocation on instances in InfDegreeOne to MaxMinDegree Arborescence. Consider an instance in InfDegreeOne. We first remove from the instance, all players that have at least two big items. We will assign items to these players later. For the remaining instance, we build a dependency graph as follows.

Definition 4.5. The *dependency graph* is a directed graph G. G has a vertex corresponding to every player in the instance, and a vertex corresponding to every item whose utility is only 0 or 1 (i.e., an item whose infinity-degree is zero). If v is a vertex corresponding to a player i_v, we also associate v with the unique big item

[3]So named because of *pyramid schemes*. See `http://en.wikipedia.org/wiki/Pyramid_scheme`.

of i_v (if it exists). Thus, each vertex in G corresponds to exactly one item. The edges in G are defined as follows: there is an edge from vertex u to v if and only if the player corresponding to u has value 1 for the item corresponding to v.

If there is an edge between u and v and both vertices correspond to players (i_u and i_v respectively), the player i_u may be assigned the unique big item of i_v, and i_v must then be assigned many small items to compensate. Note that sinks in the dependency graph are exactly the items whose infinity-degree is zero.

LEMMA 4.6. *Let G be the dependency graph of an InfDegreeOne instance $\mathcal{I}$. Then, the existence of an M-pyramid in G is equivalent to the existence of an assignment of value M for $\mathcal{I}$. In addition, we can do the mapping in polynomial time.*

PROOF. For a player i, let v_i denote the vertex in the dependency graph corresponding to player i. Also, for an item j with infinity-degree zero, let v_j denote the vertex corresponding to item j.

First, assume that we have an M-pyramid in G. We build the assignment for MaxMin Allocation as follows. For a player i, if v_i is present in the M-pyramid, we assign i the items corresponding to the children of v_i in the M-pyramid. If v_i is not present in the M-pyramid, we assign i her big item. Observe that each player gets items worth a value of M, since a player either gets his big item or he receives M items (his children in the graph) each having a value one. We claim that no item is picked by more than one player. It is clear that no two players can claim an item, if both players value it at one. Neither can an item be infinity for both players. Thus if a collision is going to happen, that is between a small item of one player and the big item of another. In this case, by definition of the dependency graph, the latter player should also be present in the arborescence and thus cannot get any big items at the end.

Next, suppose we have a valid assignment of value M. At first, we make sure a player picks his big item if it is available. As long as there is a set of players $P_1, P_2, \ldots, P_k$ such that each P_t gets the big item of P_{t+1} ($P_{k+1} = P_1$, of course), change the assignment such that each P_t gets his big item. Besides, we remove the assignment of a small item to a player, if the player is also assigned a big item. The M-pyramid is built from the assignment of small items. More precisely, the vertices of the M-pyramid consist of vertices v_i for players i who are assigned small items, as well as vertices v_j corresponding to small items j (with infinity-degree zero) that are assigned to players. The sources have to be present in the arborescence, since the corresponding players do not have any big items. Each non-sink vertex clearly has out-degree M. We claim that there are no cycles in the produced subgraph, which means it is a valid M-pyramid, as desired. For the sake of contradiction, assume there is a cycle $P_1, P_2, \ldots, P_k$. Note that all the vertices of the cycle represent players, since item vertices are sinks. By construction of the dependency graph, in the original solution, these players should form a cyclic dependence; i.e., each gets the big item of the next one. However, we removed such chains. □

Finally we show how to handle the players who have more than one big item; these were removed from the instance initially.

LEMMA 4.7. *Given an instance $\mathcal{I}$ of InfDegreeOne, let $\mathcal{I}'$ be the instance obtained by removing any player with more than one big item. Given a solution of value M to $\mathcal{I}'$, we can obtain a solution of value $\lfloor M/2 \rfloor$ for $\mathcal{I}$ in polynomial time.*

PROOF. Let I be the set of all players of infinity degree larger than one that we ignored initially. For all players with infinity degree more than 2, reduce it to exactly 2 by ignoring some items that they value at infinity.

Now consider the existing assignment of small items to players. Let S be the set of players who have been assigned small items. We need to assign infinity items to players in I by taking away some small items from players in S. Build an auxiliary graph G_A with vertices corresponding to players in S. Edges in this graph correspond to players in I. Suppose a player i in I has two items j_1 and j_2 that have value infinity for i. Further suppose that j_1 is currently assigned to player $i_1 \in S$ (as a small item) and j_2 is assigned to player $i_2 \in S$ (as a small item). Then we have an edge (i_1, i_2) in G_A between the vertices corresponding to players i_1 and i_2. This edge is labeled with player i. Notice that i_1 and i_2 may be the same player.

In order to assign items to players in I, we will orient the edges of graph G_A. The orientation corresponds to a reassignment of items as follows: consider an edge (i_1, i_2) corresponding to player $i \in I$. If this edge is oriented from i_1 to i_2, then i is assigned its infinity item that is currently assigned to i_1 and i_2 keeps the other infinity item for i that i_2 is currently assigned. In case $i_1 = i_2$, either orientation implies that player i grabs either of the items and $i_1 = i_2$ loses that item yet keeps the other one. This ensures that every player in I receives an infinity item. In order to ensure that players in S still have a large number of small items assigned to them, we need to ensure that that the out-degree of v is small compared to M for every vertex v in G_A.

Let d_v be the degree of vertex v in G_A. The claim is that there is a way to orient the edges of G_A such that every vertex v has out-degree at most $\lceil d_v/2 \rceil$. Notice if the player v received M_v items in the assignment of $\mathcal{I}'$, then her remaining items are at least $M_v - \lceil d_v/2 \rceil \geq \lfloor M_v/2 \rfloor$, since $d_v \leq M_v$. In order to prove this claim, we add a matching of dummy edges between the vertices of odd degree find an Eulerian tour of G_A (with the dummy edges) and orient the edges accordingly. Then the out-degree of a vertex v is at most $\lceil d_v/2 \rceil$. □

4.2 The algorithm for MaxMinDegree Arborescence

Consider a graph G as an instance of MaxMinDegree Arborescence problem. Let T be the optimal M-pyramid for the instance. We show that there is a *nearly optimal* solution T^* to this instance with small depth; the depth of an acyclic directed graph is defined as the number of edges on its longest path. Throughout this section, we use N to denote the number of vertices in G.

LEMMA 4.8. *If there is an M-pyramid in the instance, then there also exists an $\lfloor M/2 \rfloor$-pyramid of depth no more than $\frac{\log N}{\log \lceil \frac{M}{2} \rceil}$.*

PROOF. The proof is constructive. The vertices of T can be partitioned into levels. Level zero includes all the sources of G. For $i > 0$, level i is the set of vertices in T that have an edge from level $i-1$. For a vertex v in T, let $z(v)$ denote

the size of a subtree rooted at v (including v itself).

We build T^* as a union of disjoint trees, one rooted at each source. Consider the component of T rooted at r. Remove the $\lceil M/2 \rceil$ children of r having the largest subgraph sizes. For any remaining child v of r, we have $z(v) \leq z(r)/\lceil M/2 \rceil$; otherwise, since each vertex appears at most once in the tree, the number of descendants of r in the largest $\lceil M/2 \rceil$ subtrees is at least $1 + \lceil M/2 \rceil z(v) > z(r)$ giving a contradiction. Now we recurse for every remaining child of r. For a vertex v of level l that is not pruned during this process, we have $z(v) \leq z(r)/\lceil M/2 \rceil^l$. Since $z(r) \leq N$ and $z(v) \geq 1$, we get $l \leq \log_{\lceil M/2 \rceil} N = \frac{\log N}{\log \lceil M/2 \rceil}$. □

COROLLARY 4.9. *If $M \geq 3$ then there is a degree $M/2$ solution of depth $O(\log N)$.*

COROLLARY 4.10. *If $M = \Omega(N^\epsilon)$ then there is a degree $M/2$ solution of depth $O(1/\epsilon)$.*

Definition 4.11 *Unfolded tree.* There is a vertex v_π in the *unfolded tree of depth* h for any *simple* path π of length at most h starting from a source. These will include all the paths of length zero, which correspond to the set of sources in G. There is an edge from v_π to $v_{\pi'}$ if and only if $\pi' = \langle \pi, e \rangle$ is the concatenation of π and some edge e. We say a vertex v_π in G' is a *copy* of vertex u in G if π ends at u.

If T^* has depth h, then it can be mapped to a solution on the unfolded tree of depth h such that for every vertex v, at most one copy of v is included in this solution. In the reverse direction, consider any solution on the unfolded tree of depth h such that for every vertex v, at most one copy of v is picked in the solution. Then this solution can be mapped to a solution for the original instance. Hence, we focus on solving the instance corresponding to this unfolded tree, say G'. Note that the number of nodes in G', denoted by N', is $O(N^{h+1})$.

Now we are looking for a solution of minimum out-degree $M' = \lfloor M/2 \rfloor$ in G'. We write the LP as follows. Add a virtual node r' that has edges to all the sources in G'. There is a nonnegative variable x_e for each edge e. We stipulate that for any edge from r' to a source, $x_e = 1$. We denote by R the set of sources in G'. Define $p(v)$ to be the parent edge of v, for any vertex v of G'. For any vertex v in G' and a vertex v' of G, let $N^v(v')$ denote the copies of v' that are descendants of v in G'.

The complete linear program, AR-LP, is shown below. Let $v \in V(G')$ correspond to a path π in G. Then, $x_{p(v)}$ denotes the extent to which the path π appears in the solution.

(AR-LP)

$$\sum_{e \in \delta^+(v)} x_e \geq M \cdot x_{p(v)} \qquad \forall v \in V(G'), v \neq r' \qquad (12)$$

$$\sum_{u \in N^v(v')} x_{p(u)} \leq x_{p(v)} \qquad \forall v \in V(G'), v' \in V(G) \qquad (13)$$

$$x_{(r',v)} = 1 \qquad \forall v \in R \qquad (14)$$

$$x_e \geq 0 \qquad \forall e \in E(G). \qquad (15)$$

We first show that this is a valid relaxation for the problem. Consider an M-pyramid of depth h in the original graph. For any vertex v included in the pyramid,

the path from source to v can be mapped to a path in the unfolded tree. Take the union of all such paths in the unfolded tree and set $x_e = 1$ for all edges in this union of paths. For all remaining edges e, $x_e = 0$. Note that for any original vertex v', at most one copy of v' is included in this solution. It is easy to verify that this solution satisfies all the constraints of AR-LP. Constraint (12) corresponds to the fact that every non-terminal v in the solution, the out-degree of v is at least M. Constraint 13 follows from the fact that at most one copy of a vertex v is included in the solution, and $x_e = 1$ for all edges e along some path from a source to v.

TreeRound. The rounding algorithm, TreeRound, is as follows. We maintain a queue of vertices; these are vertices included in the solution produced so far. Initially, the queue consists of all sources in the graph. Draw vertices from this queue until the queue is empty, and perform the following steps for each vertex drawn. Suppose v is the vertex drawn from the queue. Recall that $p(v)$ denotes the parent edge of v. Pick a random subset of the outgoing edges from v as follows: pick outgoing edge e with probability $x_e/x_{p(v)}$ (where the choices are made independently for each edge). Notice that because of (13) and (15) these are valid probability values. We say that the current vertex claims every outgoing edge that is picked. (Note that this vertex will eventually receive a subset of the edges it claims.) Now the endpoints of the picked edges are placed on the queue if they are not sinks of G'. Roughly speaking, the rounding algorithm follows a *chain reaction*: when a node is activated, it can trigger its children. The LP variables model these reaction chains.

By (12) each vertex picked claims a subset of vertices of expected size at least M. By Chernoff bounds, with high probability, for every such vertex, the actual number of claimed neighbors is $\Omega(M)$ (assuming $M = \Omega(\log N')$, where N' is the number of vertices in G'). However, several copies of a vertex could be included in the solution. Using union bound and Lemma 4.12 we prove that with high probability, i.e., $1 - 1/\operatorname{poly}(N)$, no more than $O(\log N \log N')$ copies of any vertex are included in the solution produced. We first establish this fact and then show how to produce a feasible solution by eliminating multiple copies of the same vertex.

Lemma 4.12. *For any original vertex j of G, with probability $1 - \frac{1}{\operatorname{poly}(N)}$, the number of copies of j included in the solution produced is $O(\log N' \log N)$.*

Proof. In this discussion, c is an appropriate constant parameter. First, we make a transformation to eliminate small x_e values. For any $x_e < 1/N'^{c+1}$, we set its value to 0. Note that the probability that vertex i claims vertex j is exactly x_e if $e = (i, j)$. Consider the rounding procedure applied to the old x_e values versus the rounding procedure applied to the new x_e values. The total variational distance between the distribution of outcomes of the rounding procedure on the two different sets of x_e values is at most $2/N'^c$. Henceforth, we analyze the rounding procedure applied to the transformed x_e values; here any non-zero x_e has value at least $1/N'^{c+1}$.

For a subtree T rooted at v, let $X(T)$ be the random variable denoting the number of realized edges in T to copies of j given that the parent edge of T is chosen in the rounding process, i.e., $X(T)$ is the number of times that copies of j are included in the solution produced within T. Let $S(T) = \mathbf{E}[e^{\alpha X(T)}]$. Let

$p(T) = x_{p(v)}$; note that $p(v)$ is the parent edge of tree T. Let $f(T)$ denote the total usage of copies of a vertex j in T, i.e., $f(T) = \sum_{u \in N^v(j)} x_{p(u)}$. Note that the LP enforces the constraint (13) that $f(T) \leq p(T)$. We choose α such that $e^\alpha = 1 + 1/(4(c+1)\log N')$. The following claim is proved below.

CLAIM 4.13. *For any subtree T such that $f(T) > 0$,*

$$S(T) \leq 1 + \frac{1}{4(c+1)\log N'} \frac{f(T)}{p(T)} \left(1 + \frac{\log\left(N'^{c+1} f(T)\right)}{(c+1)\log N'}\right).$$

Let X be the total number of copies of j included in the solution produced. We have proved that $\mathbf{E}[e^{\alpha X}] \leq 2$, for $\alpha = \Theta(1/((c+1)\log N'))$, which implies that $\Pr[X > c(c+1)\log N' \log N] \leq 2/N^c$. □

Now, we give the proof of the claim we used above to finish this part of the proof.

PROOF OF CLAIM 4.13. The proof goes by induction on the height of the subtree T. For convenience, we define the base case to be a tree of height 0, where the parent edge of the tree is an edge e such that $x_e > 0$. For this tree $f(T) = p(T) = x_e$, $X(T)$ is always 1 and

$$\mathbf{E}\left[e^{\alpha X(T)}\right] = e^\alpha = 1 + \frac{1}{4(c+1)\log N'}.$$

The base case thus follows from $f(T) \geq 1/N'^{c+1}$.

Consider a subtree T consisting of a root connected to subtrees $T_1, \ldots, T_k$, via edges $e_1, \ldots, e_k$ respectively. Let e be the parent edge of T. Suppose the inductive hypothesis holds for each subtree T_i, i.e.,

$$S(T_i) \leq 1 + \frac{1}{4(c+1)\log N'} \frac{f(T_i)}{p(T_i)} \left(1 + \frac{\log\left(N'^{c+1} f(T_i)\right)}{(c+1)\log N'}\right).$$

Then, as the $X(T_i)$'s and the selections of e_i's conditioned on e being selected are independent, we get

$$\begin{aligned} S(T) &= \prod_i \left[\left(1 - \frac{p(T_i)}{p(T)}\right) \cdot 1 + \frac{p(T_i)}{p(T)} S(T_i)\right] \\ &= \prod_i \left[1 + \frac{p(T_i)}{p(T)} \left(S(T_i) - 1\right)\right] \\ &\leq \prod_i (1 + \delta_i), \end{aligned} \tag{16}$$

with δ_i defined as

$$\delta_i = \frac{1}{4(c+1)\log N'} \frac{f(T_i)}{p(T)} \left(1 + \frac{\log\left(N'^{c+1} f(T_i)\right)}{(c+1)\log N'}\right) \tag{17}$$

$$\leq \frac{1}{2(c+1)\log N'} \frac{f(T_i)}{p(T)}, \tag{18}$$

where the last inequality is due to $f(T_i) \leq 1$. Thus,

$$\begin{aligned}\sum_i \delta_i &\leq \frac{1}{2(c+1)\log N'} \frac{f(T)}{p(T)} \\ &\leq \frac{1}{2(c+1)\log N'} \\ &\leq \frac{1}{2}.\end{aligned}$$

Hence,

$$\prod_i (1+\delta_i) \leq \prod_i e^{\delta_i} = e^{\sum_i \delta_i} \leq 2.$$

Now, by Inequality (16),

$$\begin{aligned}S(T) &\leq 1 + \sum_i \delta_i + \frac{1}{2}\left(\sum_{i_1 \neq i_2} \delta_{i_1}\delta_{i_2}\right)\prod_i (1+\delta_i) \\ &\leq 1 + \sum_i \delta_i + \sum_i \delta_i \sum_{i' \neq i} \delta_{i'}.\end{aligned}$$

Note that, by Equation (17),

$$\begin{aligned}1 + \sum_i \delta_i = 1 + \frac{1}{4(c+1)\log N'} \frac{f(T)}{p(T)} \\ + \frac{1}{4(c+1)^2 \log^2(N') p(T)} \sum_i f(T_i) \log(N'^{c+1} f(T_i)),\end{aligned}$$

and, by Inequality (18),

$$\begin{aligned}\sum_{i' \neq i} \delta_{i'} &\leq \frac{1}{2(c+1)\log N'} \sum_{i' \neq i} \frac{f(T_{i'})}{p(T)} \\ &= \frac{1}{2(c+1)\log N'} \frac{f(T) - f(T_i)}{p(T)} \\ &\leq \frac{1}{2(c+1)\log N'} \int_{f(T_i)}^{f(T)} \frac{1}{x} dx \\ &= \frac{1}{2(c+1)\log N'} \log \frac{f(T)}{f(T_i)}.\end{aligned}$$

Hence, by Inequality 18,

$$\delta_i \sum_{i' \neq i} \delta_{i'} \leq \frac{1}{4(c+1)^2 \log^2(N') p(T)} f(T_i) \log \frac{f(T)}{f(T_i)},$$

Algorithm MaxMinDegree Arborescence Solver
Input: *The digraph G*
Output: *The pyramid T*

(1) If $M \leq M^*$, produce a solution with out-degree one. This can be simply done using a matching algorithm.
(2) Otherwise, build the unfolded tree G' of depth $H = \log N / \log(M^*/2)$.
(3) Solve AR-LP for G'.
(4) Round it using TreeAlg: select the sources; while there is a non-sink node v selected but not yet handled, pick each edge e of v independently at random with probability $x_e/x_{p(v)}$.
(5) By having an assignment of $1/\gamma = 1/O(\log N \log N')$ for each edge of the solution of the previous step, build a fractional solution to an LP for $\frac{M}{\gamma}$-matching, and obtain an integral solution thereafter. Take the union of all edges of the integral solution that are reachable from sources.

Fig. 4. The algorithm for MaxMinDegree Arborescence problem.

and we get

$$\begin{aligned}
S(T) &\leq 1 + \sum_i \delta_i + \sum_i \delta_i \sum_{i' \neq i} \delta_{i'} \\
&\leq 1 + \frac{1}{4(c+1)\log N'} \frac{f(T)}{p(T)} \\
&\quad + \frac{1}{4(c+1)^2 \log^2(N') p(T)} \sum_i f(T_i) \\
&\qquad \cdot \left(\log(N'^{c+1} f(T_i)) + \log \frac{f(T)}{f(T_i)} \right) \\
&= 1 + \frac{1}{4(c+1)\log N'} \frac{f(T)}{p(T)} \\
&\quad + \frac{1}{4(c+1)^2 \log^2(N') p(T)} \sum_i f(T_i) \log(N'^{c+1} f(T)) \\
&\leq 1 + \frac{1}{4(c+1)\log N'} \frac{f(T)}{p(T)} \left(1 + \frac{\log(N'^{c+1} f(T))}{(c+1)\log N'} \right). \quad \square
\end{aligned}$$

At the end, there might be vertices v of G more than one copies of which are used. We remedy this issue by building a fractional solution to a natural linear program of k-matching and finding an integral solution.[4] The entire algorithm is summarized in Figure 4. We pick the parameter $M^* = \Omega(\log^3 N / \log\log N)$ according to the desired approximation ratio. If $M < M^*$, we can find a solution of value one which is an M^*-approximation. Otherwise, we show that the solution produced via the algorithm is good.

[4]In the k-matching problem, we are given a bipartite graph and each vertex in the first partition is to receive k exclusive vertices from the second partition. To see that the polytope is indeed integral, note that one can build a maximum flow instance out of it by adding a source vertex which is connected to all vertices of the first partition, and a sink which has edges from all the second partition vertices. Then, we can put capacity k on the edges going out of the source and capacity one on all the other edges. Feasibility of the original k-matching instances leads to existence of a flow that saturates all the edges out of source.

LEMMA 4.14. *If $M \geq M^*$ then the above algorithm runs in time $O(N^H)$ and with high probability produces an M''-pyramid with $M'' = \Omega(M \log\log N/\log^3 N)$ for G.*

PROOF. We set $H = \log N/\log(M^*/2)$ as the depth of the unfolded tree. Since $M \geq M^*$, Lemma 4.8 guarantees that there exists an M'-pyramid in the unfolded tree of depth H for $M' = M/2$. The algorithm clearly runs in time polynomial in $N' = N^{O(H)}$. We also know that $H = O(\log N/\log\log N)$ from the choice of M^*.

We next show each vertex is good: if selected in the process, it has *enough* outgoing edges. By the first set of constraints (12), the expectation of the number of outgoing edges of such a vertex is at least M'. A simple application of Chernoff bounds and union bound shows that since $M = \Omega(\log N')$, with high probability, say $1 - 1/\operatorname{poly}(N')$, all nontrivial out-degrees are $\Omega(M)$.

There are N vertices in G whose in-degree we want to bound. From union bound and Lemma (4.12), we get that with probability $1 - 1/\operatorname{poly}(N)$, no vertex of G has in-degree more than $\gamma = O(\log N \log N') = O(H \log^2 N) = O(\log^3 N/\log\log N)$.

Having shown that all in-degrees are small, we build a fractional solution to an instance of MaxMin Allocation where all the item values are 0 and 1; it is indeed a bipartite M''-matching LP, for $M'' = \Omega(M/\gamma)$. We just need to divide all the variables by a factor no more than the in-degree upper bound, γ. We have a fractional solution of value M'', and the polytope is integral. Thus we can find an integral solution of the same value, which implies an $M/O(\log^3 N/\log\log n)$-pyramid for G. □

The following is an immediate corollary. We set $M^* = \Theta(N^\epsilon)$ to get the polynomial running time or pick $M^* = \Theta(\log^3 N/\log\log N)$ for the polylogarithmic approximation ratio.

THEOREM 4.15. *For any integer $t > 1$, we can get a $\max\{\Omega(\log^3 N/\log\log N), 2N^{1/t}\}$-approximation in $N^{O(t)}$ time. In particular, we get an $\Omega(\log^3 N/\log\log N)$-approximation in $N^{O(\log n)}$ time where $N = O(n + m)$.*

5. NOINFCYCLE INSTANCES

New complications arise in treating instances with infinity degree larger than one. A major issue is the presence of complex infinity components, i.e., a connected component containing at least two infinity edges. The assignment among the players and items in such a component is fragile unlike the assignments in the previous section.

Roughly speaking, we contract these infinity components into super-nodes, and carry out a rounding procedure similar to that of InfDegreeOne. Before putting things together, we are going to treat two types of super-nodes: those at the start of reaction chains, and the ones in the middle. The super-nodes take some feedback from their inputs—some of their items may be taken away—and they respond by outsourcing certain players of the component—i.e., these players do not get any items from inside the component—in such a way that the rest of the players can be matched to the available items of the component.

Firstly, in Section 5.1, we analyze a probabilistic procedure which serves as our treatment of the first type of super-nodes. We define a generalized fractional match-

ing linear program for bipartite graphs: vertices of one side may have positive *deficiencies*, say, in case there is no perfect matching. We then randomly throw away some of these vertices, with probability proportional to their deficiencies. Theorem 5.1 states that the resulting graph has a perfect matching.

The subsequent subsection provides the tools that allow the infinity components to act as a middle super-node of a reaction chain. There, we introduce a mechanism that captures the effect of a *trigger*—i.e., an item grabbed away from the component—on the rest of graph.

In the interest of simplicity, we do not make any effort to optimize the parameters.

5.1 Infinity components

The *deficiency matching* linear program is defined as follows. We have a bipartite graph $G(A, B, E)$, players on one side, say A, and items on the other, say B. There are fractional values $x_e \in [0, 1]$ on each edge e, such that the sum of the values of the edges incident on an item is at most one, i.e., $\forall j \in B : \sum_{e \in \delta(j)} x_e \leq 1$. Furthermore, each player $i \in A$ may have some *deficiency*, defined as $\Delta_i = \max\{0, 1 - \sum_{e \in \delta(i)} x_e\}$. Were all deficiencies zero, we would have a feasible solution to the natural linear program for bipartite matching on this graph. Here, we prove that if we throw out players independently with a probability proportional to their deficiencies, we will end up with a graph having a perfect matching for the remaining players. The deficiency matching linear program (DM-LP) is given below.

(DM-LP)

$$\sum_{e \in \delta(j)} x_e \leq 1 \qquad \forall j \in B \tag{19}$$

$$\sum_{e \in \delta(i)} x_e + \Delta_i \geq 1 \qquad \forall i \in A \tag{20}$$

$$x_e, \Delta_i \geq 0 \qquad \forall i \in A, e \in E. \tag{21}$$

We say we *outsource* a player if we choose to throw it out. Each outsourcing experiment is repeated $\rho = \Theta(\log^3 n)$ times to boost the probability of outsourcing a player.

THEOREM 5.1. *Suppose we have a solution $\langle x_e, \Delta_i \rangle$ to the deficiency matching LP (DM-LP). If we indepedently at random throw away each player $i \in A$ with probability $1-(1-\Delta_i)^\rho$, for a suitable $\rho = \Theta(\log^3 n)$, there will be a perfect matching saturating the set of remaining players.*

Let $G'(A', B, E')$ be the induced subgraph of G after probabilistically removing some of the player vertices. The general approach is to show that the Hall's condition is valid for G'. In particular, for a subset $S \subseteq A$, let $\Gamma(S) \subseteq B$ denote the set of neighbors of S in G'. We show that for any subset $S \subseteq A'$, the Hall's condition $|\Gamma(S)| \geq |S|$ holds. This, in turn, implies the desired perfect matching exists by a classic graph theory result; see for instance [West 2000].

Removing cycles. For the sake of analysis, we will assume that the bipartite graph G is a forest. To show this is without loss of generality, we iteratively remove cycles as follows to get a graph G^*. Take any cycle all whose edges have positive values,

and for a suitable $\epsilon > 0$, increase the values of odd-numbered edges and decrease those of even-numbered edges by ϵ, till one edge of the cycle becomes zero. It is easy to see there is always a suitable $\epsilon > 0$, such that (1) at least one edge in the cycle becomes zero; and (2) all the values on edges remain in the range $[0, 1]$. Next, we remove edges of value zero. Doing this we remove all the cycles, yet leave unchanged the deficiency probabilities and item usage constraints above. Hence, the resulting graph G^* exhibits the same behavior in the probablistic procedure as that of G. Since G^* includes only a subset of edges of G, the existence of a perfect matching in $G^{*\prime}$ implies the same for G'. Therefore, we assume for the sake of the analysis in this subsection that the input graph G is a forest.

Suppose for the sake of reaching a contradiction, that there is no perfect matching in the resulting graph. Thus, there should be a *witness*.

Definition 5.2 *Witness.* A set $S \subseteq A'$ is a violated Hall's condition set if and only if $|\Gamma(S)| < |S|$. A *witness* is a minimal violated Hall's condition set S, i.e., there is no violated set S' which is properly included in S.

Next, we characterize such minimal violated sets. A minimal violated set, simply called a *witness* henceforth, has the following structure.

(W1) The subgraph induced by $S \cup \Gamma(S)$ is connected;

(W2) Degree of items in this subgraph (i.e., the subgraph induced by $S \cup \Gamma(S)$) is exactly two; and

(W3) The deficit in the witness is exactly one, i.e., $|\Gamma(S)| = |S| - 1$.

We use the term witness to refer to either the set S of its players, or the set $S \cup \Gamma(S)$ of both players and items. If $S \cup \Gamma(S)$ is not connected for a violated set S, one of its connected components is also a violated set which contradicts minimality of witness S. If degree of an item is one in a witness, then removing it and its sole taker gives a smaller violated set. For the sake of reaching a contradiction with (W2), assume an item j has a set Λ of $k \geq 3$ neighbors in the witness S. Noting that the witness is a tree, let S_i denote the set of players in the subtree of $i \in \Lambda$ including i itself. Thus, $S = \bigcup_{i \in \Lambda} S_i$. Minimality of S implies that $\forall i \in \Lambda, |\Gamma(S_i)| \geq |S_i|$. Furthermore, if there are two players $i, i' \in \Lambda$ such that $|\Gamma(S_i)| = |S_i|$ and $|\Gamma(S_{i'})| = |S_{i'}|$, then $S_i \cup S_{i'}$ will be a smaller violated set. Thus, all but possibly one of these subtrees have at least one excess item. Hence,

$$\begin{aligned} |\Gamma(S)| &= \sum_i |\Gamma(S_i)| - k + 1 \\ &\geq \sum_i |S_i| + k - 1 - k + 1 \\ &= \sum_i |S_i|. \end{aligned}$$

We get to a contradiction which proves the original assumption was false, hence establishing (W2). It is easy to verify (W3), because if the deficit is larger than one, one can arbitrarily remove a player and get a smaller violated set.

The scheme. Take any particular witness S. Since $|\Gamma(S)| = |S| - 1$, the total deficiency of the player S is at least one, i.e., $\sum_{i \in S} \Delta_i \geq 1$. If $\rho = \Omega(\log n)$,

and we sample each player i, ρ times with probability Δ_i, with high probability $1 - 1/\operatorname{poly}(n)$, the witness is nullified. In other words, at least one player from the witness is outsourced and thus, the witness is void. One may be tempted to make this argument for all the possible witnesses to pull through Theorem 5.1. However, this is not possible, because the number of witnesses need not be polynomially bounded. Roughly speaking, in what follows, we show how to carefully pick a small number of witnesses such that (1) they nullify all the witnesses, and (2) their small number allows us to use union bound.

We compile a list of witnesses adaptively. For each witness, we run the experiments corresponding to each player in it (each repeated $O(\log n)$ times). The subsequent witnesses are chosen based on the result of these experiments. The witnesses are chosen in such a way that no player appears in more than $O(\log^2 n)$ witnesses. We continue adding witnesses until the currently outsourced players suffice to refute any conceivable witness.

For this purpose, we maintain a graph $\mathcal{G}$ called *the working forest*—initially the graph G—and let the *size* of the graph be the number of players in it. The working graph is affected by the results of the experiments so far (corresponding to witnesses already picked). We make sure that throughout the process any still-possible witness be (or more precisely, correspond to) a subgraph of $\mathcal{G}$. To keep the structure of $\mathcal{G}$ as simple as possible, we apply the following transformations whenever possible.

(R1) Remove an item of degree one along with its only taker.

(R2) If we have a player i of degree two connected to items j_1 and j_2, remove i and merge j_1 and j_2 into a single new item j.

(R3) In a component with more than two players, if a player i_1 has a branch that leads to only one leaf player i_2, then contract the entire branch (including i_2) into i_1.

The reason for (R1) is clear: minimality of the witness disallows an item j of degree one; thus, the player i asking for j cannot appear in any witness either. A witness not including player i will not be affected by (R2). To include i, both its edges should be picked too. This means that one other edge of j_1 and one of j_2 should be included as well. On the other hand, including any two edges of j in the new instance can be transformed to a witness of the original graph: this is trivial if both edges correspond to either of j_1 and j_2, and otherwise, the original player i along with its two edges should be included in the witness. As for (R3), note that anytime i_1 is selected in a witness, so should be i_2. Recall that leaves of a witness have to be player nodes. If i_1 is included, all its edges are also included, which implies presence of some nodes from the particular branch under consideration. As i_2 is the only leaf player in this branch, it has to be present along with its path to i_1. Hence in (R3), the whole branch is removed while the path connecting i_1 to i_2 is contracted into i_1.

The above transformations guarantee that we do not have any player of degree two and all the leaves are players. A player of degree one is called a *leaf* and those of degree at least three are called *joints*. The witnesses are built in stages. In each stage, the size of the graph goes down by a factor of at least 2/3; see Lemma 5.3. This ensures there are $O(\log n)$ stages in total.

Note that the graph is partitioned into several *pieces* by the joints. The leaves in each piece have joint-less paths to each other. Take notice that any such path is a valid witness. We do not include all of them in the list though. A simple observation is this: if a tree has $2k$ leaves, one can find k disjoint paths connecting pairs of leaves in the tree; see [Wu and Manber 1992] for instane. If it has $2k+1$ leaves, we ignore one leaf and do the same to the rest. Let's call this a *path matching*. Now consider the forest, say $\mathcal{F}$, induced from $\mathcal{G}$ after removing all the joints, say $\mathcal{J}$; each piece $C \in \mathcal{C}$ is a tree component of $\mathcal{F}$. We repeatedly find a path matching (disjoint sets of paths connecting pairs of leaves) of leaves in each component $C \in \mathcal{C}$ and add these witnesses to our list. Thus we do the experiments for the players included therein. We get a list of players which are outsourced. This may increase the number of components in the working graph $\mathcal{G}$. We continue doing this until no tree component has more than one leaf—a tree with one leaf is a trivial graph with one vertex and no edge. As proved later, there cannot be too many steps in each stage.

We will shortly prove Theorem 5.1. For this purpose, we need to establish the following lemma.

LEMMA 5.3. *The size of the working graph goes down by a constant factor in each stage.*

PROOF. If a piece $C \in \mathcal{C}$ is only connected to one joint $i \in \mathcal{J}$, then C is called a *leaf piece*; i is then called the parent joint of C. We let $\mathcal{L} \subseteq \mathcal{C}$ denote the set of leaf pieces. Each of them obviously has at least one leaf. In addition, we let $L \subseteq A$ be the set of leaves of $\mathcal{G}$. We use $\mathcal{J}, \mathcal{C}, \mathcal{L}$ and L to denote their respective values prior to the particular stage of our concern, and further use their primed version—$\mathcal{J}', \mathcal{C}', \mathcal{L}'$ and L'—to represent such definitions after the stage. If we do not end in a component with only two leaves (which is too easy to handle), all the leaf pieces are absorbed into their parent joints. Let us, in addition, use $\mathcal{C}^s \subseteq \mathcal{C} - \mathcal{L}$ to denote the set of non-leaf pieces that survive this stage. As nothing remains of leaf pieces, $|\mathcal{J}'| + |L'| \leq |\mathcal{J}| + |\mathcal{C}^s|$. It is also clear that $|\mathcal{J}| + |L| \geq |\mathcal{J}| + |\mathcal{C}^s| + |\mathcal{L}|$.

Think of a (bipartite) forest in which nodes are the joints and pieces. There is an edge between between a piece and a joint if they are adjacent in $\mathcal{G}$. Noting that degree of non-leaf pieces is at least two, we get, by a degree argument,

$$2(|\mathcal{L}| + |\mathcal{C} - \mathcal{L}| + |\mathcal{J}|) - 2 \geq \text{total degree} \geq |\mathcal{L}| + 3|\mathcal{J}| + 2|\mathcal{C} - \mathcal{L}|,$$

which implies $|\mathcal{L}| \geq |\mathcal{J}| + 2$.

We further claim that $|\mathcal{C}^s| \leq |\mathcal{J}|$. Root each component $C \in \mathcal{C}$ of the forest $\mathcal{F}$ prior to the stage from an arbitrary leaf player of a leaf piece, and assign any player to its parent joint. No two surviving players of non-leaf pieces can be mapped to one joint, since this would imply there is a joint-less path between them which is a

contradiction. The ratio of size after and before the stage is thus

$$\begin{aligned}\frac{|\mathcal{J}'|+|L'|}{|\mathcal{J}|+|L|} &\leq \frac{|\mathcal{J}|+|\mathcal{C}^s|}{|\mathcal{J}|+|\mathcal{C}^s|+\mathcal{L}|} \\ &\leq \frac{|\mathcal{J}|+|\mathcal{C}^s|}{|\mathcal{J}|+|\mathcal{C}^s|+\mathcal{J}|+2} \\ &\leq \frac{2|\mathcal{J}|}{3|\mathcal{J}|+2} \\ &< \frac{2}{3}. \quad \square\end{aligned}$$

We are now ready to prove Theorem 5.1.

Proof of Theorem 5.1. We assume without loss of generality that G is a forest. Then, produce the witnesses as described. The number of witnesses will be polynomially bounded. Thus, we can assume the experiments corresponding to all of them are successful by union bound.

Lemma 5.3 ensures an $O(\log n)$ bound on the number of stages. We claim each stage has at most $O(\log n)$ steps. To prove the claim, note that the size of each piece is at most n. At each step, the size of any piece C (with more than one leaf) goes down by roughly a factor two. Suppose L_C and L'_C denote the set of leaves in C before and after a round, respectively. Clearly, $|L_C| \geq 2|L'_C - 1$, since each (except for possibly one) of the surviving leaves has had a pair which is now not in C. This proves the $O(\log n)$ upper bound on the number of steps in each stage.

In each step, any player appears in at most one witness. Thus, each player repeats its outsource experiment $O(\log^3 n)$ times. $\square$

Combining Theorem 5.1 with the algorithm of InfDegreeOne, we would be done if no item in an infinity component had any small edges. The following discussion addresses this very issue.

5.2 Floods

An item j in a complex infinity component C that has at least one small edge, is called a *trigger*. We say that trigger j is *realized* or *activated* if this item is claimed by a player outside C. In this case, we might need to change the assignment inside C, or even be forced to *outsource* some of the players in C. We say a player i in C is *outsourced* if we decide it is going to use its small items only.

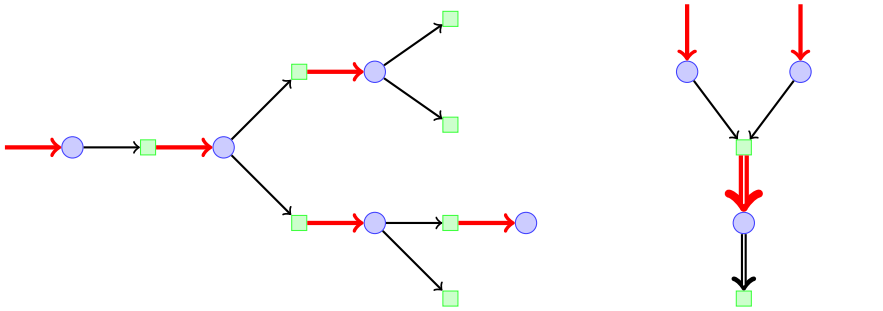
In this subsection and the next, we mostly make the assumption that there is no cycle in the input graph that merely consists of infinity edges. In other words, all infinity components are acyclic. Later, we will make an effort to deal with the issue of cycles. Let us focus our attention on a particular complex infinity component C. To simplify the treatment, we assume that

(S1) any trigger is a leaf;

(S2) only leaf players can be outsourced; and

(S3) the degree of each item is at most three.

These can be obtained using elementary transformations. In particular, (S1) can be achieved for a trigger item j by introducing a new item j' and a new player i. The



(a) Example of a (canonical) flood. (b) Floods need to overlap.

Fig. 5. Demonstration of floods.

small edges of j are now connected to j' whereas its infinity edges remain incident on j. The player i has infinite utility for both j and j'. Hence, j is no longer a trigger and j' is a leaf. To achieve (S2) for a player i, we make a similar transformation; a new item j and a new player j' are added. The small edges of i become connected to i' while the infinity edges stay incident on i. Both i and i' have infinite utility for the new item j. One can readily verify these two transformations (1) give new equivalent instances, (2) achieve (S1) and (S2), and (3) only polynomially increase the size of the instance. Next, we can use the transformation of Theorem 2.4 to obtain (S3).

We introduce the notion of a *flood*, an object that captures the effect of taking a particular trigger away from the infinity component. It determines which players should be outsourced so as to make up for the loss of the trigger. The definition is in some sense tailored for the worst case, as in whatever the assignment inside the infinity component is, the outsourcing of these players will be sufficient to handle the effect of the item's being claimed by players outside the component.

Definition 5.4. A *flood* F corresponding to a trigger j is a directed subgraph of the infinity component with the following properties:

- F includes j;
- if an item node j' is in F, so should be all the edges incident on j';
- the in-degree of each item in F is one, except for the source trigger whose in-degree is zero; and
- each player in F is either outsourced, or has one outgoing edge.

In the case of acyclic infinity components—what we are mostly dealing with here—a flood is an arborescence rooted at the trigger. We let $\mathcal{F}_j$ denote the set of possible floods rooted at a trigger j. Figure 5(a) illustrates a flood. By definition, the flood starts at the trigger and reaches several other vertices. Whenever it reaches a player i, it either continues on an outgoing edge of i or outsources the player i (in which case, it should be a leaf by (S3)). When a flood enters an item j, it should go out through all the other edges of j. The interpretation is that if the item j is taken away, then all the corresponding players are affected, whereas when a player loses

any (or many) items, she only needs to secure one other item. A flood is called *canonical* with respect to an assignment of items to players, if at each non-leaf player, the outgoing edge is that of the assignment. The (red) thick edges in the figures show the assignment, and thus the depicted floods are canonical.

The linear program, Eqs. (22)–(33), cannot enforce the last condition of floods. We are only able to make sure the fractional solution has a relaxed structure. The LP gives us a linear combination of *pseudo-floods*. A pseudo-flood is a flood except that the last condition is modified to "each player in F is either outsourced, or has *at least* one outgoing edge." To change a pseudo-flood into a flood, we simply prune the extra outgoing edges of such players, and remove portions of the pseudo-flood that are unreachable from the source trigger.

Suppose we are looking at a component C consisting of players A and items B with edges E connecting them. Further, let $E' \subseteq E$ be a matching saturating a subset $A - A^o$ of players; if we outsource the subset $A^o \subseteq A$ of players, the remaining players have a perfect matching. Let $F \in \mathcal{F}_j$ be a flood associated with a trigger $j \in B$. Then outsourcing the leaf players of F, denoted by $L^A(F)$, can compensate for the absence of the item j in the matching.

LEMMA 5.5. *There is a matching saturating $A - A^o - L^A(F)$ that does not use the item j if $F \in \mathcal{F}_j$ is a flood corresponding to the item j.*

PROOF. Build the path $p = (v_0, v_1, \ldots, v_k)$ as follows. The first vertex is $v_0 = j$. The vertices are items and players alternately. For each even index $t < k$, choose v_{t+1} such that (v_t, v_{t+1}) is an edge of the matching E'. At an odd-indexed vertex v_t for $t < k$, take the outgoing edge of F towards another vertex v_{t+1}. We stop when v_k is either an item not saturated in E' or an outsourceable player.

Now we show the construction is sane. We claim v_t is in F for all $t \leq k$. For an odd index t, the vertex v_t is a player. According to the claim, if v_t is not outsourceable, it should have an outgoing edge e. The proof of the claim is by induction on the index. Clearly v_0 is part of F. For an even $t' < k$, if item $v_{t'}$ is reachable, so is $v_{t'+1}$, since floods include all the neighbors of the reachable items. On the other hand, if v_t is reachable for an odd $t < k$, then v_{t+1} is the other endpoint of its only outgoing edge in F. It remains to argue the above process stops. If it does not, we should get stuck in a cycle. Let v_t be the first repeated vertex of such a cycle. It cannot be a player, as then the incoming edge should be an edge of E' which is unique; the previous vertex is repeated too. By the same token, neither can v_t be an item, since there is only one edge going into an item in F.

Notice that if v_k is an item, it is not used in E' and if it is a player, it can be outsourced and is part of $L^A(F)$. The edges of p are alternately in and outside E'. We claim that by flipping the status of these edges, we get a matching E'' satisfying the condition of the lemma. The only vertices affected are those on p. Clearly j is not used in E'', and the only player that may be outsourced is in $L^A(F)$. All the other players in $A - A^o$ are still saturated. It remains to show E'' is indeed a matching. If there is a vertex v with more than one edge in E'', it should be on the path p. Except possibly for v_k, any vertex of p already had an edge in E' and now has at most one. As for v_k, if it is saturated in E'', it implies it is a leaf item of C. Thus it cannot have more than one edge. □

The lemma shows that given a matching in the component, a flood reduces to a path whose flipping gives us a new matching. This is part of our algorithm. A major problem may arise if two triggers of a component are claimed and their respective floods have common portions showing up in their pathified version.

Overlaps. Ideally we want these floods to be disjoint. However, this is not always possible. Consider the example given in Figure 5(b). This component has three items and two players. Two of the items are triggers, and one of the players may be outsourced. Suppose in the integral solution, the player is outsourced and the two triggers are claimed from outside. A possible flood correponding to either trigger is a path starting at the trigger and ending at the outsourceable player; these are the canonical floods. Other possible non-canonical floods are those that start at one trigger and end at the other. It is easy to observe one cannot assign a flood to each realized trigger such that the floods are disjoint. The trick is to allow a small congestion on the edges. The following lemma ensures this is possible. This will be useful, because the triggers realized in the rounding correspond to small items for their takers. So, it is sufficient to keep only a good fraction thereof. In the next subsection, we describe how the conflict resolution for the realized intersecting floods is done.

LEMMA 5.6. *We can assign floods to all the realized triggers of an integral solution of* NoInfCycle *such that the congestion on each edge or item is at most two while only outsourcing players that are outsourced in the integral solution.*

It is worth noting that the congestion cannot be bounded if we insist on canonical floods. Canonical floods have a nice interpretation, but we need to resort to other floods to prove this lemma.

PROOF OF LEMMA 5.6. Without loss of generality, for the ease of demonstration, we assume the degree of players in the infinity component cannot be more than three. This can be achieved using a transformation similar to that of Theorem 2.4.

We start with the induced graph H of the union of all the canonical floods. Then we prove by inducton on the size of this graph that we can find good alternate floods. During the *rerouting* process, we will only use the edges in H, but possibly in the opposite direction. Hence, we can consider each connected component separately, and without loss of generality, we assume H and H' are trees in the following discussion. In fact, we prove the following two stronger claims simultaneously. Before stating the claims, let us define a player in a subtree including it as *accepting* if it is an outsourceable leaf or has its assignment edge inside the subtree. In a similar fashion, we call an item *accepting* with respect to a subtree if it is an unsaturated leaf or has its assignment edge inside the subtree.

(C1) A subgraph H' of H can be fixed so that the congestion on all its edges and items is at most two even if we have two extra units of flood generated at a particular accepting player of H'.

(C2) A subgraph H' of H can be fixed so that the congestion on all its edges and items is at most two even if we impose one additional flood unit into an accepting item of H'.

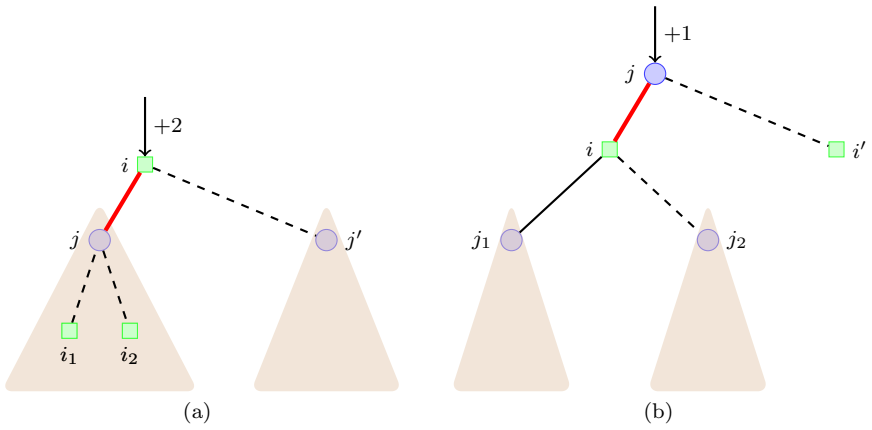


Fig. 6. Demonstration of the proof of Lemma 5.6.

First of all, let's see why these two are sufficient to finish the proof. Take a trigger j in H. It is a leaf. Let i be its only neighbor. The edge (i, j) is clearly not in the integral solution. A trivial case is when i is an outsourceable player. Otherwise, i has its assignment edge in H. Then, $H - \{j\}$ can be fixed via (C1), while i can accept two additional flood units. Finally, j will use one unit from this capacity. Notice that here and further in the following proof, we make critical use of the fact that H is the union of canonical floods of realized triggers of an integral solution. This will impose a nice structure on H.

Proof of (C1). Let i be the special accepting player node of H'. If i is an outsourceable leaf, we are done. Otherwise, it has its assignment edge included. Say, it receives an item j; refer to Figure 6(a). If j is the only child of i, we can use induction to get the claim as follows. The item j has zero, one or two children. With no child, we can just stop the two additional flood units (the only flood in this component) at j. Otherwise, all the children have to be accepting. Thus, we use (C1) on each of the subtrees to say we can sort out those subtrees and also route the two additional flood units. Now consider the case when i has another child in H', say j'. We can apply (C2) on j' and its subtree. By (C2) there is at most one flood unit coming out of j', and one more can go in. We do the following: the two additional flood units coming into i are divided between j and j'; j' can take it by (C2). The possible flood coming out of j' is directed through i to j. We know that j can accept these two flood units.

Proof of (C2). Suppose the special accepting item of H' is j. The case of j being a leaf is trivial. Otherwise H' includes its assignment edge, say, to a player i; refer to Figure 6(b). The fact that j is at all present in H indicates that all its children are accepting. Assume for now i is the only child of j. Player i has one or more subtrees, say, through j_1 and j_2. We can apply (C2) on j_1 and j_2 themselves. If j_1 is the only child, it sends its only one flood unit up through i to j, and j sends its additional unit of flood down through i to j_1. If j_2 is also present, j_1 sends its

outgoing flood through i to j_2, and j_2 sends its outgoing flood through i to j while j sends its additional flood unit through i to j_1. If a subtree i' is also present in H', we observe that we can apply (C1) to it. What happens inside the subgraph corresponding to i does not change, but two additional floods are fed to i'; one coming from outside to j and one coming out of i through j towards i'. □

5.3 The algorithm

The characteristic graph. The characteristic graph is different from the dependency graph of Section 4, yet has the same overall spirit. We give an informal description of the graph here which is followed by a more detailed precise definition. The characteristic graph has a vertex for any item or player. The connections outside complex infinity components are similar to those of dependency graph except that in the construction of dependency graph, we collapsed the two edges connected to an intermediary item node into one. In the new construction, though, we include both edges: if a player i asks for an item j with small utility and the item j is big for another player i', we then have an edge from i to j and one from j to i'. The complex infinity components are handled differently. Each such component is present in the construction and we stipulate a trigger thereof may only be used as a small item by outside players if provisions for a flood rooted at the said trigger exist in the complex infinity component. In addition, we have an extra copy of complex infinity components that serve as the *super-sources* of the solution/rounding: each such component provides a matching for a subset of its players and outsources the rest. An outsourced player acts as a source of the dependency graph.

Let $G(A, B, E)$ be the original instance. The characteristic graph $D(V_D, E_D)$ is defined as follows. Let $A^\infty \subseteq A$ be the subset of players in complex infinity components. Similarly, define $B^\infty \subseteq B$ as the subset of items in complex infinity components. Then, D has a vertex for any item or player and an extra copy for any item in B^∞ or player in A^∞. For an object with two copies, the first is the normal copy while the second one is called the *matching* copy. In other words, each complex infinity component appears twice in the characteristic graph; once as a source super-node (the matching copy which can initiate some chain reactions by outsourcing several players) and another time (the normal copy) as a middle super-node. We have four types of edges, defined as follows; $E_D = E_1 \cup E_2 \cup E_3 \cup E_4$.

- E_1 For any small edge $e = (i, j) \in E$ between player i and item j, there is one (or two) corresponding directed edges in E_D. Let i and j denote the normal copies of the corresponding vertices in D, and let i^* denote the matching copy of i if present. Then, there is a directed edge from i to j and possibly one from i^* to j in D.
- E_2 For an infinity edge $e = (i, j) \in E$ between player i and item j where $j \notin B^\infty$, we add to E_D a directed edge going from j to i. This corresponds to a simple infinity component.
- E_3 Any infinity edge $e \in E$ which is inside a complex infinity component, appears in D as an undirected edge between the normal copies of its endpoints.
- E_4 Any infinity edge $e \in E$ which is inside a complex infinity component, also appears in D as an undirected edge between the matching copies of its endpoints.

Roughly speaking, the matching copies of complex infinity components host a matching between items and players. Any player in the characteristic graph outside the normal copies of infinity components that does not receive an infinity item, should receive about M small items. Each trigger item which is actually claimed from outside its infinity component has a corresponding flood in that component. The leaf players of the respective floods are also activated as a player that does not receive an infinity item.

The intended solution. Suppose we have a solution $f : A \mapsto 2^B$ of value M to the original instance $G(A, B, E)$. We show how we interpret the characteristic graph by turning that solution to a *solution* of D. Look at a complex infinity component $C(A_C, B_C)$. Let $A^o \subseteq A_C$ be the set of players in C that do not receive infinity items from B_C, and let $B^t \subseteq B_C$ be the set of trigger items taken by players not in A_C. We use the matching $\mathcal{M}$ on $A - A_o$ saturating items $B^{\mathcal{M}} \subseteq B - B^t$ in the matching copy of component C. Let $F_j \in \mathcal{F}_j$ be the canonical flood originating at $j \in B_t$ with respect to the matching $\mathcal{M}$. By definition of floods, the set of leaves of F_j is a subset of $A_o \cup B - B^{\mathcal{M}}$. We reroute these floods to get a collection of floods F'_j with small overlap, say μ. These are the floods we use in the normal copies of the components. The small assignment of the players is trivial from f. Notice that each player is outsourced for at most $\mu + 1$ times; once for the matching copy and at most μ times for the floods. Thus, for any such player i, we can divide the set of items $f(i)$ into $\mu + 1$ disjoint pieces while each has at least a fraction $1/(\mu + 1)$ of the value. Doing this, any player in a matching copy either receives an infinity item or is outsourced. Any player in a simple infinity component takes hold of her infinity item unless she is activated. Small items are only used once due to the $\mu+1$ partitioning.

Bounded depth solution. Now, equipped with the tools for the probabilistic existence of perfect matchings and the flood discussion, we are almost ready to describe the algorithm. To this end, we first prove a generalized version of Lemma 4.8 to bound the depth of our *characteristic graph*. Roughly speaking, we want to take a solution to the characteristic graph, say as described above, and contract the infinity components. Then, the claim is that this graph can be trimmed to a small depth without substantially affecting the solution value. A directed path in the characteristic graph is called a *flow path* if it follows the corresponding floods inside complex infinity components.

LEMMA 5.7. *If there is a solution of value M where overlap of floods is at most μ, then there is also a solution of value $\lfloor M/2(\mu + 1) \rfloor$ for which the depth of the characteristic graph is $\frac{\log m}{\log \lfloor M/2(\mu+1) \rfloor}$, where the depth is defined as the maximum number of type one edges in any flow path.*

PROOF. The argument is similar to that of Lemma 4.8. However, we first need to change the graph slightly for the argument to go through. During this process, what remains of an infinity component is basically its triggers and copies of its outsourceable players. Remove all the items and the saturated players in the matching copies of infinity components. For a trigger item j in a component C that is claimed from outside C, there should be a realized flood $F \in \mathcal{F}_j$ in the solution. Using an edge of the second type, connect j to a fresh copy of any player

in $L^A(F)$. Notice that there are at most $\mu + 1$ occurences of any outsourceable player, μ from the normal copy and one from the matching copy of the component. Each of them receives at least M items in the solution. We partition the set of such items into groups of size at least $\lfloor M/(\mu+1) \rfloor$ and each copy of the player receives an exclusive partition. It is easily observed that (1) the new solution has value at least $\lfloor M/(\mu+1) \rfloor$, (2) each player appears at most $(\mu+1)$ times, (3) there are no edges of type three or four, and (4) the solution forms a tree which means each item is used by at most one player. Now we can carry out the same argument as in proof of of Lemma 4.8 to establish the lemma. □

The reaction graph. We also add nodes for the edges and items in the complex infinity components. A trigger is connected to all the elements (edges and items) in its flood by an edge of the second type. Let us for future reference name this transformed graph as the *reaction graph*. By the above lemma, we know that the reaction graph can be trimmed to a small depth. We will later use this graph in the congestion bound argument.

COROLLARY 5.8. *If there is a solution of value M for an instance of* NoInfcycle*, then there is a solution of value $M/6$ to the characteristic graph for which the depth of the characteristic graph is $\frac{\log m}{\log \lfloor M/6 \rfloor}$ after contracting the infinity components.*

The linear program. We let S^A denote the set of player nodes in the characteristic graph that have any small items. Further, denote by $\mathcal{N}^B$ the set of items *inside* normal copies of infinity components. That is, $\mathcal{N}^B$ consists of the items in complex infinity components that have no small edges. We use $\delta(v)$, $\delta^-(v)$ and $\delta^+(v)$ to denote the set of all, incoming and outgoing incident edges of v, respectively. Similarly $\Gamma(v)$, $\Gamma^-(v)$ and $\Gamma^+(v)$ are used to denote the set of all, incoming and outgoing neighbors of v, respectively. Let h be the depth of the characteristic graph, and M_h the value of the trimmed solution (see Lemma 5.7). Define $M_h^* = M_h/\mu$ as the value of the linear program. With slight misuse of notation, we let A^∞ and B^∞ denote matching copies of vertices and A and B refer to the normal copies. We add dummy nodes s and s' to V_D; these nodes serve as the starting point of all the chain reactions. In addtion to x_e and Δ_{i^*} variables corresponding to the deficiency matching subproblem, there is a variable f_e^π for any directed edge e and any ordered list $\pi \in \Pi$ of vertices V^π whose size is bounded by $2h$. The set V^π of recordable vertices consists of s, s', $B - \mathcal{N}^B$ and S^A. The list π serves as a selected subset of nodes on the particular chain reaction leading to this edge. Roughly speaking, it remembers all the nodes except for those inside complex infinity components, which is in fact the set of all the nodes on the corresponding path in the reaction graph. The LP is given in (22)–(33), and description of each set of constraints is included inline.

$$\sum_{e \in \delta(j^*)} x_e \leq 1 \qquad \forall j^* \in B^\infty \tag{22}$$

$$\Delta_{i^*} + \sum_{e \in \delta(i^*) \cap E_1} x_e = 1 \qquad \forall i^* \in A^\infty \tag{23}$$

$$\Delta_{i^*} = 0 \qquad \forall i^* \in A^\infty - S^A \tag{24}$$

These three constraints enforce the deficiency matching in matching copies of complex infinity components of D, allowing nonnegative deficiencies for outsourceable players.

$$f_{(s,s')}^{(s,s')} = 1 \tag{25}$$

$$f_{(s',i^*)}^{(s,s',i^*)} = \Delta_{i^*} \qquad \forall i^* \in A^\infty \tag{26}$$

Dummy source node s will be the starting point of all reaction chains. It is connected to s' which is, in turn, connected to all the outsourceable players in matching copies, i.e., $A^\infty \cap S^A$. We assume all the edges incident on s' have utility infinity except for the one coming from s whose utility is one.

$$\frac{1}{M_h^*} \sum_{\substack{j' \in \Gamma(i) \\ j' \neq j}} f_{(i,j')}^{\pi \circ j'} \geq f_{(j,i)}^{\pi} \qquad \forall i \in S^A, j \in \Gamma^\infty(i) - B^\infty \tag{27}$$

$$\sum_{\substack{j' \in \Gamma(i) \\ j' \neq j}} f_{(i,j')}^{\pi} \geq f_{(j,i)}^{\pi} \qquad \forall i \in A - S^A, j \in \Gamma(i) \tag{28}$$

Here $\pi \circ \pi'$ denotes the concatenation of the two lists π and π' and $\Gamma^\infty(v)$ denotes the set of neighbors of v having infinity edges to v. Depending on whether a player is in S^A or $A - S^A$, the flood/flow pseudo-conservation is written as above. The former is for small edges and ensures if a player is outsourced (or activated) it should receive M_h^* small items. In this case, there is a unique $j \in \Gamma^\infty(i) - B^\infty$ because of our transformations. The latter constraints make sure if there is any flood of a particular kind entering a player node i, at least as much should exit from i. This concerns the inside of normal copies of infinity components.

$$\sum_{\substack{i' \in \Gamma(j) \\ i' \neq i}} f_{(i',j)}^{\pi} \leq f_{(j,i)}^{\pi} \qquad \forall j \in \mathcal{N}^B, i \in \Gamma^\infty(j) \tag{29}$$

$$\sum_{\substack{i' \in \Gamma(j) \\ i' \neq i}} f_{(i',j)}^{\pi \circ j} \leq f_{(j,i)}^{\pi} \qquad \forall j \in B - \mathcal{N}^B, i \in \Gamma^\infty(j) \tag{30}$$

When a flood enters an item node, it has to exit through all the other possible edges. Similarly, if an item j is taken away by a player, the infinity takers of j are affected. There is only a slight difference between the two sets of constraints: we only store j in π if j is not inside a complex infinity component.

$$\sum_{\substack{\pi' \\ \pi \circ v \circ \pi' \in \Pi}} f_e^{\pi \circ v \circ \pi'} \leq f_{\sigma(\pi,v)}^{\pi \circ v} \qquad \forall \pi \in \Pi, v \in V^\pi - \{s\} \tag{31}$$

$$\sum_{\substack{\pi' \\ \pi \circ v \circ \pi' \in \Pi}} \sum_{e \in \delta(j)} f_e^{\pi \circ v \circ \pi'} \leq f_{\sigma(\pi,v)}^{\pi \circ v} \qquad \forall \pi \in \Pi, v \in V^\pi - \{s\}, j \in B \tag{32}$$

Here $\sigma(\pi, v)$ represents the edge leading to v on the list π. Notice that the other endpoint of the edge may not be in π as not all vertices are recorded. In particular, $\sigma(\pi, v)$ is defined as

$$\sigma(\pi' \circ u \circ v \circ \pi'', v) = \begin{cases} (s, s') & \text{if } v = s', \\ (u, v) & \text{if } v \text{ is an item,} \\ (u', v) & \text{if } v \text{ is a player and } p_{v,u'} = \infty. \end{cases}$$

Notice that $\sigma(\pi, v)$ is well-defined for any $v \in V^\pi - \{s\}$ that appears on π: if v is a player, it has exactly one infinity item. The constraints (31)–(32) stipulate the conditional variables should be consistent. In particular, the total flood passing through an edge e conditioned on another flood has to be at most one. Combined with the dummy edge (s, s'), this gives an unconditional bound of one on such variables as well.

$$\Delta_{i^*}, x_e \geq 0 \qquad \forall i^* \in A^\infty, e \in E_D. \tag{33}$$

Note that we do not have conservation for the flows inside infinity components. At item nodes, the flows may double. To turn the solution described above to a solution for the LP, we divide all the small edge values as well as the floods by μ. This decreases the value of the LP by a factor of μ.

The definition given above for the LP for inside of normal copies of infinity components, ensures a linear combination of pseudo-floods of value (at least) f originating at a trigger whose incoming flow is f.

Now we show the LP described above is a valid relaxation of the problem.

Claim 5.9. *If the MaxMin Allocation instance has a solution of value M with flood packing upper bound of μ, the above LP has a feasible fractional solution of value M_h^*.*

Proof. Take the reaction graph corresponding to the integral solution on the bounded depth solution. The assignment of values for Δ_{i^*} and x_e are straightforward. Let the integral assignment inside complex infinity components dictate the values for x_e variables for any edge $e \in E_1$. A player $i^* \in A^\infty$ saturated in the matching gets $\Delta_{i^*} = 0$, whereas a player $i^* \in A^\infty$ using its small edges in the integral solution gets $\Delta_{i^*} = 1$.

Notice the reaction graph is a forest. For any edge $e = (u, v)$ appearing in the solution, let π_e be the path from s to v in the reaction graph. Define π'_e as a restricted version of π_e that only includes vertices from V^π. If e is a real edge of the reaction graph, define $\nu(e) = e$. Otherwise, v is an added element to the reaction graph. If v corresponds to an edge e' of the original graph (inside a complex infinity component), define $\nu(e) = e'$. We do not define $\nu(e)$ if v is an element corresponding to an item. Finally, we let $f^{\pi'_e}_{\nu(e)} = 1/\mu$ if $\nu(e)$ is defined.

All the other variables are set to zero except those dictated by (25) and (26). We now show that all the constraints in the LP are satisfied by the above assignment. Clearly all the variables are nonnegative in accordance with (33). The matching constraints (22)-(24) hold by definition noting we have a matching for such nodes in the integral solution. Constraints (25) and (26) are valid by construction.

In the reaction graph, each player using small edges has M_h outgoing edges. By

Algorithm MaxMin Solver
Input: *The special MaxMin Allocation instance (NoInfCycle or InfDegreeTwo)*
Output: *The assignment of items to players*

(1) If $M \leq M^*$, produce a solution with out-degree one (using a matching algorithm).
(2) Build characteristic graph after reducing degrees, etc.
(3) Solve the LP given in (22)-(33).
(4) Do independent randomized outsourcing with $\rho = \Theta(\log^3 m)$ in the matching copies of the complex infinity components.
(5) Find a perfect matching $\mathcal{M}$ on the remaining players of such components.
(6) Simplify the fractional floods in the normal copies into flows using $\mathcal{M}$.
(7) Peform TreeRoundPlus which is similar to TreeRound
- Start from initial set of activated nodes (outsourced players of the above steps and the sources) and continues according to TreeRound except for the following points.
- Apply a dampening factor of $\Theta(\log^3 m)$ to sampling of small items for an activated player
- Whenever a trigger item is activated, randomly pick one of the paths in its simplified flood using the associated conditional probabilities (boosted by a factor $\Theta(\log m)$) and activate (i.e., outsource) the endpoint of the path if it is a player.

(8) Take away some of the triggers from their current players using the Flow Integralization.
(9) For each surviving trigger path, flip all the assignments along the path.
(10) Do a conflict resolution on the items similar to the previous algorithm.

Fig. 7. The algorithm for NoInfCyle and InfDegreeTwo problems.

our construction, definition of $M_h^* = M_h/\mu$ leads to constraint (27). Constraint (28) is inferred from the definition of floods in the reaction graph. Definition of floods in the reaction graph gives constraints (29) and(30) for items inside complex infinity components and the definition of the reaction graph for items in simple infinity components gives it for others.

Constraint (31) is true since the reaction graph is a forest and the assignment of f_e^π is derived from a flow-like structure. For (32), notice that an item either has one incoming edge or it is in a complex infinity component. In the latter case, we assume without loss of generality that each item appears at most once in the flood associated with any single trigger. □

The algorithm. The algorithm, summarized in Figure 7 is a generalization of the one described in Section 4. We first carry out the necessary transformations and then build the characteristic graph, trim it and solve the LP. The rounding is then done in stages. Some players in the matching copies of complex components are outsourced using using the procedure described in Subsection 5.1 with $\rho = \Theta(\log^3 m)$. The discussion therein guarantees the existence of a perfect matching saturating the nonoutsourced players. We then find such a perfect matching $\mathcal{M}$ using conventional matching algorithms. Afterwards $\mathcal{M}$ is used to simplify the fractional floods in the normal copies into flows; see Claim 5.10 below. The next step is TreeRoundPlus which is a generalization of TreeRound of Section 4. More specifically, place all the source nodes as well as the outsourced players of the complex infinity components into a queue. Next, draw nodes one by one from this queue and process them.

- When a player is drawn, we sample an expected $M_h^*/\Theta(\log^3 m)$ number of her items according to the *dampened* conditional probabilities dictated by the LP.
- Upon drawing a nontrigger item, its (sole) infinity taker is placed in the queue.
- Suppose a trigger item j is drawn. Notice that the *flow* out of j in the corresponding infinity component can be decomposed into paths reaching leaves (either outsourceable players or unused items). Randomly pick one of them and outsource the endpoint if it is an outsourceable player, by putting it in the queue. More precisely, form the conditional probabilities for these paths, and sample them independently with probability boosted by a factor $\Theta(\log m)$. This guarantees at least one of them is selected during the sampling.

Next, we take away some of the triggers from their current players using the FLOW INTEGRALIZATION described below. This serves as part of the conflict resolution for multiply-assigned items, and in addition, makes it possible to handle the players inside complex infinity components. Flip all the assignments along the paths corresponding to surviving triggers. Finally, we perform a conflict resolution on the items similar to the previous algorithm, MAXMINDEGREE ARBORESCENCE SOLVER.

Flood to flow simplification. Floods are worst-case guaranteed structures, as previously noted. When a matching $\mathcal{M}$ is provided, we can focus on paths. In the worst case, when we reach an item node j, we have to go out through all the edges. Whereas if we know which player i currently claims j, we can wisely decide which outgoing edge to pick. If no player uses j in the matching, the path can stop at j. Otherwise, the path continues along the current matching edge towards i.

In the above LP, roughly speaking, we have a linear combination of floods. Given a matching $\mathcal{M}$, this fractional solution turns into a flow (a linear combination of paths). To this end, we cross out any edge $e \notin \mathcal{M}$ going out of an item and any item or node not reachable from the trigger afterwards. We say a set of edge values is a *pseudoflow* if we have a relaxed conservation property: in particular, for any vertex v, we have $\sum_{e\in\delta^-(v)} x_e \leq \sum_{e\in\delta^+(v)} x_e$. The value of a pseudoflow is the sum of values of edges going out of the source vertex.

CLAIM 5.10. *The result of the above trimming is a pseudoflow of the same value as the original flood.*

PROOF. The above process produces a graph G_F. Clearly, there is at most one edge entering any item and at most one edge going out of a player. In addition, if an item j except source is reachable from the source vertex j_s, then j has exactly one incoming edge. Then, writing (29) and (30) for j implies pseudo-conservation at j. On the other hand, if a player i is reachable in G_F but is not outsourceable, then i has exactly one outgoing edge. Hence, using (28) for i gives pseudo-conservation at i. Therefore, we have pseudo-conservation at all nodes except the source and sinks. □

It is straightforward to reduce the values of a pseudo-flow into those of a flow. Using the pseudo-flow edge values as capacities, we can find a maximum flow and that is the desired flow. One can observe that the minimum cut of the pseudo-flow is at least as large as the value of the pseudo-flow.

LEMMA 5.11. *In the above random process, with high probability, no element (edge or item) is sampled more than* $O(\text{poly}\log m)$ *times.*

PROOF. The reaction graph is very similar to the dependency graph except that each intermediate edge is doubled and the flood out of a node need not be M times its inflood. Claim 5.10 says the trimming performed in the third step after getting the matching reduces the flood sections of the LP into flows. By the LP constraints (31) and (32), we know that the total flow into any item or edge is at most one, even if conditioned on any relevant event. Hence, we can focus on any item or any edge and carry on the same argument as Lemma 4.12 to finish the proof. In fact, we can obtain a feasible solution to (AR-LP) on the reaction graph via the solution of the above LP. The bound will be $O(\log N' \log N) = O(h \log^2 N)$ for one single element, i.e., item or edge. We assume $h = O(\log m)$ without loss of generality. Knowing $N = O(m)$, we can then apply union bound to get the guarantee for all the elements. □

Flow integralization. Another complication in the algorithm is flipping of paths that are not disjoint. The constraints we put in the LP prevent this in an integral solution. However, we might have different fractional paths that share segments, and many such paths might have been realized in the rounding. The good point is that not many paths will share a single edge. This is an extension of the congestion bound we proved in Section 4. Let λ be this bound. The idea is to take away some trigger from their current takers, without reducing the solution value significantly, while on the other hand, the paths corresponding to the yet-present triggers become disjoint. During this, some paths might be rerouted to different players, but the target players have already been outsourced.

Roughly speaking, we build a graph of all these paths. Then we put capacities of one on the edges (and item nodes) and add edges from a new source vertex to the triggers and from outsourced players to a new sink vertex. The capacities on the edges out of source are picked such that all capacities are integral and further, the maximum flow saturates all such edges. Then, we can find an integral maximum flow which is exactly what we want; i.e., paths are edge-disjoint.

Now we give a precise explanation of how this process is carried on. This is integrated with the final conflict resolution for InfDegreeOne. Build a graph G_F as follows. All the items and players are vertices of G_F. There is a source node s connected to all the players that use their small items as a result of rounding. The capacity of the edge from s to player i is $\lfloor M_h^*/\lambda \rfloor$. There is a sink node t that has edges from all the leaves of the flood paths as well as from non-trigger items used as small items in the rounding. The capacity of any such edge is one. The rest of the edges are the union of the flood paths, and the capacities of these edges are also one. To enforce capacity one on the item nodes, we apply the following well-known trick. We break each item j inside complex infinity components into two nodes j_1 and j_2. There is an edge of capacity one from j_1 to j_2. All incoming edges of j are now connected to j_1 and all outgoing edges head out of j_2. We put a value $1/\lambda$ on the edge between player i and item j if the LP rounding decides i should use j. Then, by reducing the value of (flood) paths to $1/\lambda$ and removing the extra paths of each player, we can get a fractional feasible flow from s to t that saturates all the edges out of s. Since all capacities are integral, there is an integral solution of the

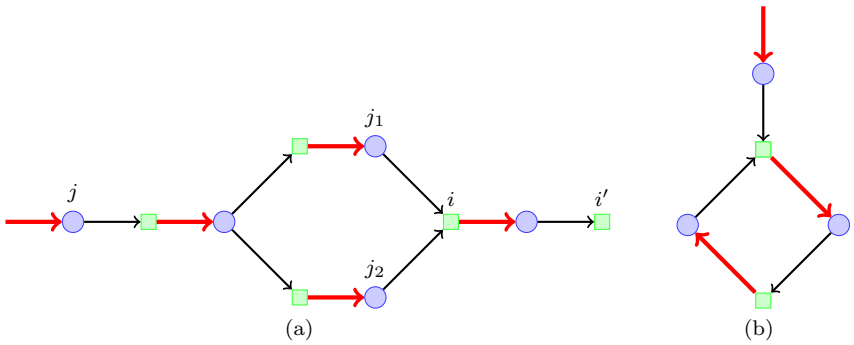


Fig. 8. Floods with cycles.

same value. Thus each player i keeps a total of $\lfloor M_h^*/\lambda \rfloor$ trigger paths non-trigger items of hers. These paths are edge-disjoint and item-disjoint, but may intersect at player nodes. Hence, we can perform the flipping procedure to modify the complex infinity components in effect releasing the realized triggers.

THEOREM 5.12. *The algorithm above can be used to give an $\tilde{O}(n^\epsilon)$-approximation that runs in time $O(n^{1/\epsilon})$.*

PROOF SKETCH. We guess the value of the solution, say M. If M is smaller than $O(n^\epsilon)$ we can get an $O(n^\epsilon)$-approximation by a matching algorithm. Otherwise, by Claim 5.9, the LP has an integral solution of value $M' = \Omega(M/\mu^2 n^\epsilon)$ of depth $h = O(1/\epsilon)$.

The LP value inside the matching copies of complex infinity components is a deficiency matching. Hence by Theorem 5.1, we can find a perfect matching in Step 2 that saturates all the remaining players, i.e., those not outsourced. Each player receives $M'/O(\log^3 m)$ items.

The congestion analysis works out as before with minor changes to show that (1) each item is used by at most $\lambda = O(\log^3 m)$ players, and (2) each infinity edge (or item) is used by at most λ realized flood paths; see Lemma 5.11.

Then using the flow integralization and previous conflict resolution, we lose a factor $O(\lambda)$ to resolve all conflicts. Finally, we flip the edges along all the now-disjoint remaining trigger paths. The running time comes from the size of the LP which is $\text{poly}(n^h)$. □

5.4 InfDegreeTwo instances

If there are infinity cycles in the instance, the floods can be quite complicated. Floods need not be arborescences anymore. Definition 5.4 is suitable for such cases as well. Figure 8(a) shows a flood with a cycle. The item j is a trigger and the only leaf player of its flood is i'. Notice that although two items, namely j_1 and j_2, of the player i are taken, one outgoing edge is sufficient for i. Recall the interpretation is that a flood shows which players are affected by a trigger's being taken away. The presence of such subtle structures makes is more challenging to reroute the floods that have cycles. In fact, there are examples that show we cannot get a properly

bounded congestion via rerouting the floods; see, for instance, Example 5.13 below. In this section, we show that congestion 8 is achievable in case of InfDegreeTwo.

Example 5.13. Consider a component C consisting of an outsourceable player i_0 connected to an item j_0. There are k other players $i_1, \ldots, i_k$ in C connected to j_0. Each i_t for $1 \leq t \leq k$ has an exclusive trigger j_t. In an integral solution, it is possible that all these triggers are realized when outsourcing i_0 is sufficient to sort things out in C. However, the flood corresponding to any trigger i_t has to include the edge from j_0 to i_0, leading to congestion k. Here, the degree of the item j_0 is not bounded by three. Yet, we can use the gadget in proof of Theorem 2.4 to get away with this issue.

LEMMA 5.14. *We can assign floods to all the realized triggers in an integral solution of InfDegreeTwo such that the congestion on each edge is at most* 8 *while the new leaves of the floods are either previously outsourced players or unused items.*

We can perform the transformations to make outsourceable players leaves of the infinity components; however, we cannot ensure that triggers are leaves as well. The previous transformation would need items of infinity degree three which we disallow here. The flood associated with a trigger j may have two branches, as the degree of the item can be two. Each branch will be a (not necessarily simple) path, because (1) upon entering a player node, we only follow one edge out, and (2) whenever we enter an item node, we have already used one edge and there is at most one more edge remaining. The two branches may later merge at a player node. They cannot cross at that point, since we are considering canonical floods at present. Furthermore, notice that in a feasible solution, the canonical flood corresponding to activated trigger j cannot pass through another activated trigger j'. Since, the implication would be that triggering of j makes j' unavailable to the outside.

Now let us consider the two branches separately. Consider two instances. In the first instance, take the first branch of each flood, and in the second, look at the respective second branches. We prove that each of these instances can be rerouted to give a good congestion. Then, we use the following claim to finish the proof.

CLAIM 5.15. *If the two instances corresponding to first and second branches of the floods can be rerouted to give congestion λ each, then the floods in the original instance can be rerouted to give congestion 2λ.*

PROOF. Superimpose the two rerouted solutions. Clearly, the congestion is at most 2λ. Slight change is needed to make the floods valid. Let p_1 and p_2 be the rerouted paths for different branches of a single trigger. Nothing is to be done if they do not intersect. If they do, however, let i be the first player on p_1 that is also in p_2. Suppose $p_1 = p_1^1 p_1^2$ where p_1^1 ends at i. The new flood for the corresponding trigger consists of p_1^1 and p_2. It is easy to verify that this forms a valid flood and its only intersection point is i. □

As noted earlier, these paths need not be simple; see Figure 8(b) for instance. Now, we characterize the cycles in the union of single branches from the canonical floods.

(Q1) There is no trigger in a cycle;

(Q2) Edges come into cycles only through player nodes; and

(Q3) The cycles are edge-disjoint.

If a trigger j falls in a cycle, it means it has an incoming edge. This means that realization of some trigger inhibits triggering of j, proving (Q1) by contradiction. The degree bound on items implies (Q2). To prove (Q3), note that each player has *exactly* one outgoing edge and each item (that is not a trigger) has *exactly* one incoming edge, since we have a union of canonical floods. We claim each cycle is directed, that is, all the edges of the cycles have the same orientation. Take a player node i on a cycle C. At least one of the two edges of the player on the cycle is incoming. Go back through that edge to j. The item j has exactly one incoming edge, and it has to be part of C. Go back through that edge and continue until we reach i. We will stop since we only walk on the cycle C. This proves the claim. Now take two distinct intersecting cycles C_1 and C_2. Since cycles are directed, there has to be a common point at which C_1 and C_2 have different outgoing edges. Such a node can neither be an item nor a player which proves (Q3).

Now, we are ready to prove the congestion lemma for InfDegreeTwo which in turn gives our final result.

Proof of Lemma 5.14. Using Claim 5.15, we just need to consider one branch from each trigger and pack them with low congestion. Take either of the two instances. The above discussion shows that the cycles in such an instance are disjoint, which means there are no nested cycles. We make a small transformation to remove cycles, and then solve the problem using Lemma 5.6 Finally, we show putting cycles back in, the increase in the congestion is small.

Let a cycle C have players $i_1, i_2, \ldots, i_k$, in order, which have incoming edges from outside C. Further, let j be the item in the cycle with an edge to i_1. Remove this edge. This may render some portions of the corresponding flood unreachable from the source triggers. Remove any such portions. Doing this iteratively for all the cycles, we end up with an instance without any cycle, and thus by Lemma 5.6, floods can be rerouted to give congestion at most two. There are at most two flood units reaching j. Suppose j_1 and j_2 are the respective origins of these floods, say F_1 and F_2. These two are clearly outside C. Let i'_1 and i'_2 be the (not necessarily distinct) players through which F_1 and F_2 enter C, respectively. Now, restore the edge between j and i_1. Add the path around C from j to i'_1 to F_1, and similarly add the path from j to i'_2 to F_2. These are valid floods for the original instance and congestion on any edge is now at most four. By combining the two branches, we get the lemma. □

The congestion lemma shows that our approach can be generalized to InfDegreeTwo.

6. CONNECTION TO SHERALI-ADAMS LIFT-AND-PROJECT METHOD

We show that there is a close connection between MaxMinDegree Arborescence Solver (see Section 4) and the rounding of a natural lift-and-project LP. Let G be the given directed graph, and G' the unfolded tree of depth t for G. For simplicity, assume that G already includes the additional dummy root r'. However, in the following LP formulations, we do not include the constraints that stipulate

the variables corresponding to the dummy edges should be set to one. Those are always implicit. We start with a natural LP for the MaxMin Degree Arborescence problem on G.

(LP0)

$$\sum_{e\in\delta^+(v)} x_e = M \cdot \sum_{e\in\delta^-(v)} x_e \qquad \forall \text{ vertex } v \neq r' \tag{34}$$

$$\sum_{e\in\delta^-(v)} x_e \leq 1 \qquad \forall \text{ vertex } v \text{ of } G \tag{35}$$

$$x_e \geq 0 \qquad \forall \text{ edge } e \text{ of } G \tag{36}$$

This is the very S-LP for the corresponding InfDegreeOne instance. Next we obtain a lift-and-project LP, called $\mathfrak{SA}(t)$, which is the linear program resulting from performing t rounds of Sherali-Adams lift-and-project on LP0. There is a variable x_S in $\mathfrak{SA}(t)$ for any set S of edges in G if $|S| \leq t$. In particular, there are variables x_p for any path p of length at most t in G. (Unless specified otherwise, paths are simple.) We claim that these variables specify a solution to AR-LP corresponding to G'. Let us rewrite AR-LP in a different way to make this connection more conspicuous. We have a variable x_p in GR-LP corresponding to any variable x_v for $v \in G'$, where p is the path from r' to $v \in G'$. Let $\mathcal{P}$ be the set of paths starting at r' in G.

(GR-LP)

$$\sum_{p'=\langle p,e\rangle} x_{p'} \geq M \cdot x_p \qquad \begin{array}{r}\forall \text{ path } p \in \mathcal{P} \text{ of } 1 \leq \text{ length } < t \\ \text{not ending at a sink}\end{array} \tag{37}$$

$$\sum_{\substack{|p=\langle p',p''\rangle|\leq t \\ p \text{ ends at } v}} x_p \leq x_{p'} \qquad \forall \text{ vertex } v \in G, \text{ path } p' \in \mathcal{P} \tag{38}$$

$$x_p \geq 0 \qquad \forall \text{ path } p \in \mathcal{P} \text{ of length } \leq t \tag{39}$$

We should also have some constraints of the form $x_p = 0$ if $|p| = t$ and p does not end at a sink. Before worrying about these constraints, let us show that the constraints in GR-LP are in fact present in $\mathfrak{SA}(t)$.

We present a handier version of the Sherali-Adams derivation of constraints that is more convenient for our purposes; its proof comes later on. Roughly speaking, we can $\mathfrak{SA}$-multiply one constraint of $\mathfrak{SA}(t-1)$ by one of the base LP to get a new constraint.

Lemma 6.1. *If there is a constraint*

$$\sum_{i=1}^{k} \alpha_i x_{S_i} \leq \sum_{i=1}^{k'} \alpha'_i x_{S'_i} \tag{40}$$

in $\mathfrak{SA}(t-1)$ *and a constraint*

$$\sum_{i=1}^{m} \beta_i x_{e_i} \leq \sum_{i=1}^{m'} \beta'_i x_{e'_i} \tag{41}$$

in the base LP, there will be a constraint

$$\left(\sum_{i=1}^{k} \alpha_i x_{S_i}\right) * \left(\sum_{i=1}^{m} \beta_i x_{e_i}\right) \leq \left(\sum_{i=1}^{k'} \alpha'_i x_{S'_i}\right) * \left(\sum_{i=1}^{m'} \beta'_i x_{e'_i}\right) \tag{42}$$

in $\mathfrak{SA}(t)$*, where* $*$ *denotes the Sherali-Adams product operator.*

Now we use induction on t to prove the constraints (38) are present in $\mathfrak{SA}(t)$. It is trivial for $t = 0$. Suppose $t > 0$: if v is the endpoint of p', then the constraint is simply $x_p \leq x_p$. Otherwise, let $u_1, \ldots, u_k$ be the parents of v. Let $e_1, \ldots, e_k$ be the respective edges that connect each u_i to v. By (35), we know that

$$\sum_{i=1}^{k} x_{e_i} \leq 1. \tag{43}$$

By the inductive hypothesis, we have

$$\sum_{\substack{|p=\langle p', p''\rangle| \leq t-1 \\ p \text{ ends at } u_i}} x_p \leq x_{p'}.$$

$\mathfrak{SA}$-Multiplying by (43) (i.e., we multiply the left-hand side (LHS) by LHS and the right-hand side (RHS) by RHS; see Lemma 6.1), and noting that $x_S \geq 0$ for any set S of edges, we get

$$\sum_{\substack{|p=\langle p', p''\rangle| \leq t \\ p \text{ ends at } v}} x_p \leq x_{p'},$$

where we have removed the resulting variables on the LHS which do not correspond to valid paths.

Now we show how the constraints (37) are derived. Let $e' = (u, v)$ be the last edge of the path p. Let $e'_1, \ldots, e'_k$ be the parent edges of v corresponding to parents $u_1, \ldots, u_k$. Similarly, let $e_1, \ldots, e_m$ be the outgoing edges of v. We first note that the variable $x_{\{e'_i, e'_j\}} = 0$ for $i \neq j$, since $\sum_{i=1}^{k} x_{e_i} \leq 1$ and nonnegativity of the variables give

$$\sum_{i=1}^{k} x_{e_i} \leq \sum_{i=1}^{k} x_{e_i} + \sum_{i \neq j} x_{\{e_i, e_j\}} = \left(\sum_{i=1}^{k} x_{e_i}\right) * \left(\sum_{i=1}^{k} x_{e_i}\right)$$
$$\leq \left(\sum_{i=1}^{k} x_{e_i}\right) * 1 = \sum_{i=1}^{k} x_{e_i}.$$

By (34)

$$\sum_{i=1}^{m} x_{e_i} = M \cdot \sum_{i=1}^{k} x_{e'_i}.$$

$\mathfrak{SA}$-multiplying by $x_{e'}$ we get

$$\sum_{i=1}^{m} x_{\{e', e_i\}} = M \cdot \sum_{i=1}^{k} x_{\{e', e'_i\}} = M \cdot x_{e'}.$$

Now, $\mathfrak{SA}$-multiplying by the rest of the path p (edges other than e'), we get the constraints (37).

Now we give the proof of Lemma 6.1.

Proof of Lemma 6.1. Writing the Equation (40) in standard Sherali-Adams form and $\mathfrak{SA}$-multiplying by $\beta_j x_{e_j}$, we get

$$\sum_{i=1}^{k'} \alpha'_i \beta_j x_{S'_i \cup e_j} - \sum_{i=1}^{k} \alpha_i \beta_j x_{S_i \cup e_j} \geq 0. \tag{44}$$

By $\mathfrak{SA}$-multiplying Equation (41) by variables corresponding to the elements in S'_j one at a time, we get

$$\sum_{i=1}^{m'} \alpha'_j \beta'_i x_{S'_j \cup e'_i} - \sum_{i=1}^{m} \alpha'_j \beta_i x_{S'_j \cup e_i} \geq 0. \tag{45}$$

Summing Equation (44) and Equation (45) over all respective values for j, and adding up,

$$\sum_{j=1}^{k'} \sum_{i=1}^{m'} \alpha'_j \beta'_i x_{S'_j \cup e'_i} - \sum_{j=1}^{m} \sum_{i=1}^{k} \alpha_i \beta_j x_{S_i \cup e_j} \geq 0,$$

which is equivalent to Equation (42). □

There is still a caveat in interpreting our rounding algorithm of the unfolded tree as rounding $\mathfrak{SA}(t)$: we do not have any variable in GR-LP corresponding to paths of length more than t. However, we only showed that if all the long paths are available, the constraints (37) are satisfied. Now we show that if $M > 2n^{1/t}$, we can prune some of the paths such that no *long* path remains, yet the solution is valid for $\mathfrak{SA}(t)$ of value $M' = M/2$. The following lemma is an analog of Lemma 4.8 for the fractional solution of the lift-and-project LP.

Lemma 6.2. *If $M > 2N^{1/t}$, we can prune the solution of $\mathfrak{SA}(t)$ such that no path longer than t has a nonzero variable, while the value of the LP does not go below $M/2$.*

Proof. To simplify the demonstration, we work with the unfolded tree (without the depth restriction) and all references to subtrees are subtrees of this unfolded tree. We define a measure of size of a subtree in terms of the LP variables as follows: For a subtree T (of the unfolded graph), let $\text{flow}(T)$ denote the sum of flows into all nodes in T (this includes flow into root of T as well). Let $\text{in}(T)$ denote the flow into the root of T. Then we define $\text{size}(T) := \text{flow}(T)/\text{in}(T)$.

Suppose a subtree T has subtrees T_i. Then, note that

$$\begin{aligned} \text{flow}(T) &= \text{in}(T) + \sum_i \text{flow}(T_i) \\ &= \text{in}(T) + \sum_i \text{in}(T_i) \cdot \text{size}(T_i). \end{aligned}$$

Thus,

$$\text{size}(T) = 1 + \frac{\sum_i \text{in}(T_i) \cdot \text{size}(T_i)}{\text{in}(T)}.$$

Also note that $\sum_i \text{in}(T_i) \geq M \cdot \text{in}(T)$.

Here is how the pruning works. Proceed in a top-down fashion starting with all the sources. Suppose we are currently processing node v. Let T be the subtree rooted at v. For convenience, assume the subtrees T_i of T are in decreasing order of $\text{size}(T_i)$. Let j be the smallest index such that $\sum_{i=1}^{j} \text{in}(T_i) \geq \frac{M}{2}\text{in}(T)$. Recall that $\sum_i \text{in}(T_i) \geq M \cdot \text{in}(T)$; so such a j exists.

We claim that

$$\text{size}(T_j) < \frac{\text{size}(T)}{M/2}.$$

Since for $i \leq j$, $\text{size}(T_i) \geq \text{size}(T_j)$, we have

$$\begin{aligned} \text{flow}(T) &> \sum_{i=1}^{j} \text{in}(T_i) \cdot \text{size}(T_i) \\ &\geq \text{size}(T_j) \sum_{i=1}^{j} \text{in}(T_i) \\ &\geq \text{size}(T_j) \cdot \frac{M}{2} \cdot \text{in}(T). \end{aligned}$$

Hence, $\text{size}(T_j) < (\text{flow}(T)/\text{in}(T))/(M/2) = \text{size}(T)/(M/2)$, as claimed.

We delete all subtrees $T_1, \ldots, T_{j-1}$. This ensures that all remaining subtrees have size less than $\text{size}(T)/(M/2)$, and the total flow out of the root of T is at least $(M/2) \cdot \text{in}(T)$. Repeat this process in a top-down fashion. When you get to a tree T such that $\text{size}(T) < M$, the root of T must be a sink. This proves that the depth is at most $\log N / \log(M/2)$ and this is a feasible LP solution for $M' = M/2$. □

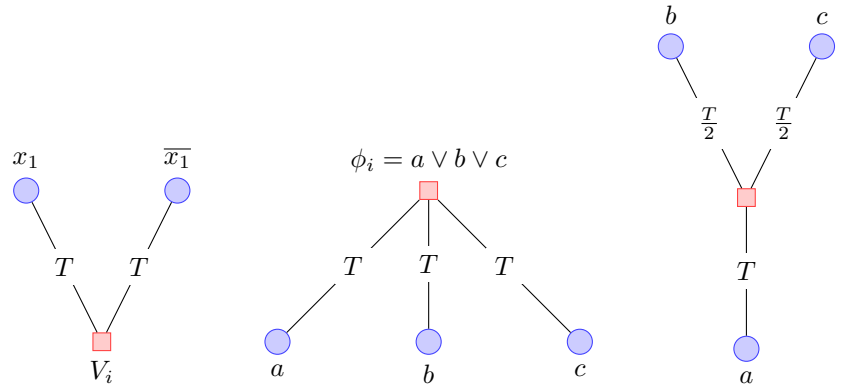
To summarize, this is the interpretation of our algorithm as rounding the lift-and-project LP: write and solve $\mathfrak{SA}(t)$; prune the solution; and round it. In the rounding step, a path $p = \langle p', e \rangle$ is selected with probability $x_p/x_{p'}$ if and only if p' is already selected.

7. HARDNESS BARRIER

Theorem 2.4 is a motivation to study DegreeTwo, as instances which are in a sense one notch easier than the general problem. Here, we show another motivation. We first give an approach to generalize Theorem 3.1, to possibly get better hardness results. Our approach uses a gadget for which we do not know any construction with guarantee better than 2. So, we fail to establish any hardness better Theorem 3.1. But, one might still hope to build better gadgets and use them to get better bounds. We show no such gadgets exist to give a hardness better than a factor 4.[5] And then, observe that all the previous hardness results for the problem (all achieving a factor 2) are somewhat using this gadget, explicitly or implicitly. This signifies we need something else if we are after better hardness results.

The gadget that we will need is the following.

[5]Considering a recent $(2+\epsilon)$-approximation algorithm due to [Chakrabarty et al. 2009] actually ensures no hardness result better than what is currently known can be obtained in this way.



(a) Variable gadget. (b) Clause Gadget. (c) Duplication Gadget.

Fig. 9. Parts of our generalized hardness approach: (a) for each variable, the item corresponding to one literal can be used outside the gadget; (b) for each clause, the item corresponding to one of its literals has to be used; (c) a simple $(2-\epsilon, T)$-duplication gadget.

Definition 7.1 *Duplication Gadget.* An (α, T)-duplication gadget is an instance of MaxMin Allocation with three distinguished items a, b and c, satisfying the following properties.

(1) Having a, all the players can get a load of T;
(2) Having b and c, all the players can get a load of T; and
(3) Without a and either of b or c, we cannot obtain a solution of value T/α or better.

The gadget can be used to copy the status (being available or not) of a to both b and c. We use such a gadget to get a hardness of factor α for the problem. We build a 3SAT instance as in proof of Lemma 3.1. There are two items and one player for each variable x_i; one item for each literal and a dummy player. The player values each item at T. Thus, in any nonzero solution, one of those items would be taken. Each literal is duplicated via the above gadget as necessary and the new copies can form clauses. We know that for each variable, the copies for exactly one choice (positive or negative) are available outside duplication gadgets. Such literals can help satisfy the clauses, i.e., provide them with load T. Figure 9 depicts different parts of our construction.

Details of the construction are omitted, but this gives a hardness of factor α for MaxMin Allocation. It is not immediately obvious that this gives an instance of DegreeTwo. The following lemma helps establish this.

Lemma 7.2. *Any (α, T)-duplication gadget can be changed to have degree at most two for items, and furthermore in such a way, that the distinguished items have degree one.*

Proof. By Definition 7.1, there is a solution in case we only have a. Let E_1 be

all the edges (assignment of items to players) in this case. Let E_2 similarly be the assignment in the case when we have b and c, but not a. Taking $E_1 \cup E_2$ gives what we seek. The degree of a, b and c is at most one, and no other item has degree more than two. As we only have a subset of the original edges, it is obvious that the third condition of the definition also holds. □

Coupling this lemma with our construction, we get an instance of DegreeTwo. We do not know any way to get an (α, T)-duplication gadget for $\alpha \geq 2$. For $\alpha = 2 - \epsilon$, this is obvious; see Figure 9(c). The algorithm in Section 3 refutes existence of such gadgets for $\alpha \geq 4$. The better algorithm given in [Chakrabarty et al. 2009] shows there is no such gadget with guarantee better than $2 + \epsilon$. Hence, insisting to use such a method cannot yield results better than what we currently have.

8. CONCLUDING REMARKS

In this work, we give quasipolynomial time polylogarithmic approximation algorithms for the bounded degree case of MaxMin allocation of indivisible goods. In particular, we show that solving DegreeThree is equivalent to solving MaxMin Allocation, and on the other hand, we study InfDegreeTwo.

Although the recent results of [Chakrabarty et al. 2009] essentially settles this question using a different approach, it is still interesting to see if our approach can be extended to solve DegreeThree. In particular, were we able to eliminate infinity cycles in some way, we could derive the same result of [Chakrabarty et al. 2009]. Otherwise, we might be able to change the definition of floods to make this possible.

Getting better approximation ratios and/or running times is definitely another next step. As the MaxMinDegree Arborescence problem seems to capture the essence of the problem, it might be a reasonable choice for getting better hardness results or approximation ratios. Coupled with a recent result of [Chakrabarty et al. 2009], the discussion of Section 7 shows what we cannot do in order to get any hardness results better than a factor 2.

It will also be interesting to gain a better understanding of the Sherali-Adams lift-and-project LP for this problem. In particular, finding a tight integrality gap is a nice direction to pursue.

REFERENCES

Alon, N., Azar, Y., Woeginger, G. J., and Yadid, T. 1998. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling 1,* 1, 55–66.

Asadpour, A., Feige, U., and Saberi, A. 2008. Santa claus meets hypergraph matchings. In *Proceedings of the 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. Springer, 10–20.

Asadpour, A. and Saberi, A. 2007. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, New York, NY, 114–121.

Bansal, N. and Sviridenko, M. 2006. The santa claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, New York, NY, 31–40.

Bezáková, I. and Dani, V. 2005. Allocating indivisible goods. *SIGecom Exchange 5,* 3, 11–18.

Chakrabarty, D., Chuzhoy, J., and Khanna, S. 2009. On allocating goods to maximize fairness. *CoRR abs/0901.0205*.

Chan, Y. H., Fung, W. S., Lau, L. C., and Yung, C. K. 2008. Degree bounded network design

with metric costs. In *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 125–134.

Ebenlendr, T., Krčal, M., and Sgall, J. 2008. Graph balancing: A special case of scheduling unrelated parallel machines. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, Philadelphia, PA, 483–490.

Feige, U. 2008. On allocations that maximize fairness. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, Philadelphia, PA, 287–293.

Golovin, D. 2005. Max-min fair allocation of indivisible goods. Tech. Rep. CMU-CS-05-144, School of Computer Science, Carnegie Mellon University. June.

Khot, S. and Ponnuswami, A. K. 2007. Approximation algorithms for the max-min allocation problem. In *Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. Springer, 204–217.

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

West, D. B. 2000. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall.

Woeginger, G. J. 1997. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters 20,* 4, 149–154.

Wu, S. and Manber, U. 1992. Path-matching problems. *Algorithmica 8,* 2, 89–101.