Approximation Algorithms Workshop June 13-17, 2011, Princeton

Lecture: On Allocating Goods to Maximize Fairness — June 15, 2011

Julia Chuzhoy *Scribe: Parinya Chalermsook*

1 The problem

In the Max-Min Allocation problem, we are given a set $\mathbf{A}$ of m agents and set $\mathbf{I}$ of n items where each agent $A \in \mathbf{A}$ has utility $u_{A,i}$ for item $i \in \mathbf{I}$. If $u_{A,i} > 0$, we say that agent A is *interested in* item i. Let $\pi : \mathbf{I} \to \mathbf{A}$ be an allocation that assigns items to agents. The total utility each agent receives from π is defined as $u_A(\pi) = \sum_{i \in \mathbf{I}:\pi(i)=A} u_{A,i}$. Our goal is to find an assignment π that maximizes the minimum utility among the agents, that is, we want to compute π that maximizes $\min_{A \in \mathbf{A}} u_A(\pi)$.

We sketch the key ideas used in an $\tilde{O}(n^\epsilon)$ approximation algorithm due to Chakrabarty, Chuzhoy, and Khanna [CCK09].

Literature: This problem was first studied by Bezakova and Dani [BD05] where they show $(n-m+1)$-approximation and prove that the problem is hard to approximate to within a factor of 2. Later Asadpour and Saberi [AS07] showed an $\tilde{O}(\sqrt{m})$ approximation algorithm by rounding a configuration LP of Bansal and Sviridenko, matching the lower bound on the integrality gap of the LP, as shown in [BS06]. Recently, Chakrabarty, Chuzhoy and Khanna [CCK09] significantly improved the approximation ratio to $O(n^\epsilon)$ for any $\epsilon = \Omega(\log\log n/\log n)$. The running time of their algorithm is $n^{O(1/\epsilon)}$. Their algorithm also implies a poly-logarithmic approximation ratio in quasi-polynomial time. Independently of [CCK09], Bateni, Charikar, and Guruswami [BCG09] gave similar approximation ratio of $O(n^\epsilon)$ in time $n^{O(1/\epsilon)}$ for restricted instances of the following forms: First they assume that the utilities are in the set $\{0, 1, M\}$, and they consider the graph whose vertices are $\mathbf{A} \cup \mathbf{I}$ where there is an edge (A, i) if and only if $u_{A,i} = M$. Their algorithm gives the above ratio if either: (i) the graph has no cycles, or (ii) each item has at most two adjacent edges.

A very interesting special case of Max-Min Allocation is when we assume that each item $i \in \mathbf{I}$ has the same utility u_i for a set of agents $\mathbf{A}_i \subseteq \mathbf{A}$, and the utility is zero for all other agents. This special case is called the Santa Claus Problem and has received a lot of attention. Bansal and Sviridenko [BS06] showed an $O(\log\log m/\log\log\log m)$ approximation algorithm by rounding the configuration LP they use. Later, Feige [Fei08] and subsequently Asadpour, Feige, and Saberi [AFS08] showed that the integrality gap of the Bansal-Sviridenko LP is bounded by a constant, but their proofs are not constructive. Finally, a constant-factor approximation algorithm for the Santa Clause problem was given by Haeupler, Saha, and Srinivasan [HSS10].

2 Canonical Instances and Flow Network

Let M denote the value of the optimal solution. For simplicity of the presentation, we assume that the utilities only take values in $\{0, 1, M\}$. This special case still captures the challenges of the problem. In the optimal solution, there are two ways to assign items to agents so that each agent gets enough utility: either assign M utility-1 items, or assign one utility-M item to each agent. An α-approximate solution would ensure that each agent is either assigned one item of utility M, or M/α items of utility one.

2.1 Canonical Instances

It is easy to show that, we can assume w.l.o.g. that the input instance is in a specific form, called *canonical instances*, in which there are only two types of agents: **heavy agents** and **light agents**.

- Each heavy agent A has all utilities $u_{A,i} \in \{0, M\}$.
- Each light agent A has utility M for one unique item $h(A)$, and her utility is either zero or one for all other items.

If $A \neq A'$, then $h(A) \neq h(A')$. Our goal is to find an $O(n^\epsilon)$ approximate solution to these canonical instances.

2.2 Flow network

Now we formulate the problem as an (integral) flow routing problem in network. We will first compute an *initial temporary assignment* π' of items to agents. In this initial assignment, some agents may not get any item, and we will create a flow network for which feasible flow solution gives us final assignment π, satisfying all the agents. The main idea of the algorithm is to iteratively reassign the items using flow routing.

We compute an initial assignment of private items as follows: Each light agent A is assigned the item $h(A)$. We then construct a bipartite graph where vertices on the left-hand-side represent the heavy agents, and the right-hand-side has the set of items that have not been assigned yet. We connect agent A to item i iff A is interested in i. Then we compute maximum matching in this graph. This matching, together with the assignment of items to light agents, is our initial assignment π'. If heavy agent A is assigned an item, we say that such agent has a private item in the initial assignment. Observe that, each light agent has a private item in π', while some heavy agents may not have a private item. Such heavy agents are called *terminals*.

Now we create the flow network where the vertex set is $\mathbf{A} \cup \mathbf{I} \cup \{s\}$. We will connect the vertices by directed edges using the following rules (see Figure 1 for illustration):

- Let $\mathbf{S} \subseteq \mathbf{I}$ be the set of items that are not assigned to any agent by π'. We connect s to each vertex in set $\mathbf{S}$.
- For each terminal A and item i that agent A is interested in, we have an edge from i to A.

- For each non-terminal A (light or heavy), there is an edge from A to her private item. There is an edge from each item i that is not a private item to A iff A is interested in i.

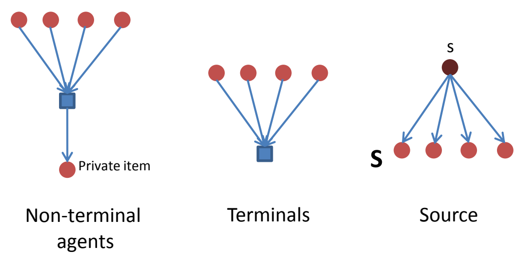

Figure 1: Edges in the flow network. Squares represent agents, while circles are items and the source s.

Flow constraints: We will be looking for an integral flow satisfying the following constraints:

- A non-terminal heavy agent sends a unit flow iff she receives a unit flow.
- Each light agent sends one flow unit iff she receives M flow units.
- Conservation of flow on items.
- No more than one flow unit going through any vertex.
- Each terminal receives one flow unit.

In an *α-relaxed flow*, each light agent sends one flow unit iff she receives M/α flow units. It is easy to see that finding an α-approximate solution is equivalent to finding an α-relaxed flow in this network.

Interpretation of flow:

1. Each flow-path represents a "reassignment chain", e.g. a flow path from $i_1 \to A_1 \to \ldots \to i_k \to A_k$ (where $A_1, \ldots, A_k$ are heavy agents, and $i_1, \ldots, i_k$ are items) can be interpreted as an assignment of i_1 to agent A_1, who is releasing her private item i_2 which is in turn assigned to A_2, and so on.
2. The flow through a light agent corresponds to the agent releasing her private item and being assigned a set of M/α light items instead.

If we forget about the source s and consider the graph induced by edges carrying one flow unit, the resulting graph is simply a collection of directed trees where each tree is rooted at a terminal and has a subset of vertices in $\mathbf{S}$ as leaves (as shown in figure 2). Since heavy agents have in-degree one and out-degree one, if we break the trees at light agents, we get a collection of paths, called *elementary paths.* Each elementary path corresponds to a flow-path. There are three possible types of elementary paths: (i) a path from a vertex in $\mathbf{S}$ to a light agent, (ii) a path from a light agent to a light agent, and (iii) a path from a light agent to a terminal (observe that, due to the property of maximum matching, it is impossible to have an elementary path that does not go through light agents).

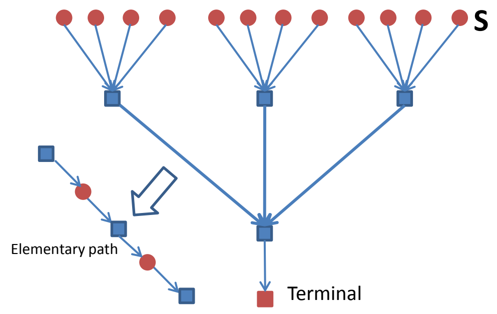

Figure 2: A tree in the flow network. The arcs in this network are elementary paths.

Equivalent problem formulation: Find a collection of disjoint directed trees τ, each having height at most h and rooted at a terminal, while maximizing the minimum degree of light agents. Furthermore, we need to ensure that we get one such tree for each terminal. The arcs of these trees correspond to elementary paths. If we are hoping for $O(n^\epsilon)$ approximation, we may assume that $h \leq 1/\epsilon$. From now on we focus on solving this problem by LP rounding. The LP rounding algorithm is used as a subroutine to get the final solution.

3 The LP and the Algorithm

We proceed to write an LP relaxation for the flow problem. First it is easy to see that the standard flow LP relaxation is too weak, so we will be using a stronger LP. For each $h' \leq h$, for each h'-tuple of light agents, we have an indicator variable for having a flow-path containing these light agents. Besides the flow conservation constraints, we also need a set of constraints that ensure consistency between these variables. The size of this LP is $n^{O(h)}$.

Unfortunately, even this LP has integrality gap of $\Omega(\sqrt{m})$, so we cannot hope to get a low-cost feasible solution by rounding this LP. We will instead use LP rounding to find an "almost-feasible" solution whose value is within a poly-logarithmic factor of the optimal, and such rounding procedure is used iteratively as a subroutine to compute the final solution. In an almost feasible solution, each item is allowed to be assigned at most twice. More precisely, we have two types of elementary paths: *blue paths* going to the terminals, and *green paths* going to the light agents. The paths of the same color are (internally) disjoint, but each item or agent may be shared by one green and one blue path (i.e. congestion two is allowed). Such solutions can be computed by rounding the LP. We will not get into the detail of the LP rounding procedure here.

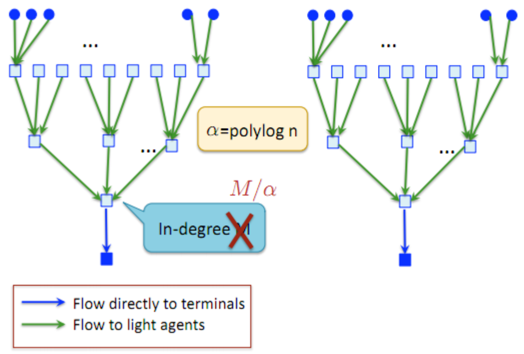

Figure 3: Structure of an almost-feasible solution.

Overcoming the LP integrality gap: Let us first try an obvious way to lower the integrality gap. Although the integrality gap is $\Omega(\sqrt{m})$, it is easy to see that, roughly speaking, it is not more than the number of terminals in the instance. So an obvious direction one might try is to minimize the number of terminals in the initial assignment so that the integrality gap is as small as possible. However, our algorithm already computed a maximum matching between heavy agents and items, which would minimize the number of terminals. So this approach cannot lower the integrality gap.

We now revise the plan. Our algorithm proceeds in iterations. In each iteration, we find a partial assignment of items to a subset $\mathbf{A}'$ of light agents, to be removed from the instance (but we do not remove any items). After removal of such agents, we can reassign their private items to other agents, so the number of terminals definitely goes down in the next iteration, thus improving the integrality gap. However, observe that the items that were assigned to agents in $\mathbf{A}'$ may be assigned again in later iterations, and this could cause problems if we are not careful with the way we select $\mathbf{A}'$, e.g. in Figure 4 (the "not nice" case), we assign the same utility-M item to two agents; in the end, at most one agent could get this item, while the other will have utility zero. This scenario is bad for us since it does not allow any flexibility. However, in the "nice" case of Figure 4, two agents are satisfied by a large amount of utility-1 flows, so if we randomly assign each item, each agent only loses a small fraction of incoming flow with high probability. These examples suggest that some choices of partial assignments are bad for us. Our algorithm will ensure that the partial assignment found in each iteration is "nice" in the sense that it allows us to satisfy the removed agents in the end.

Finding a nice assignment: We use the almost feasible solutions computed by LP-rounding to produce nice assignments. Roughly speaking, we design a subroutine that takes as input a collection of trees where some vertices may lie on both green and blue paths and there is a tree for every

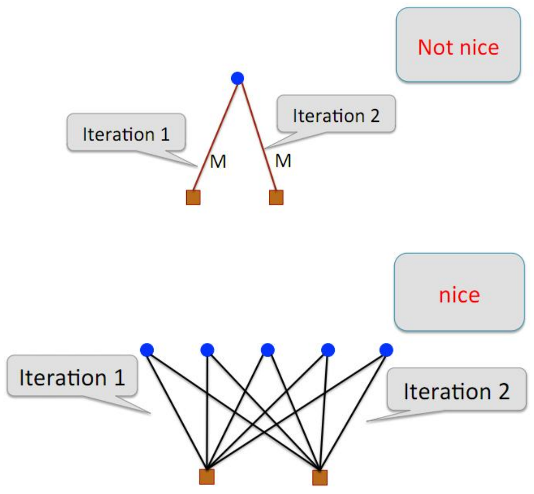

Figure 4: Nice assignments.

terminal. The subroutine will output a collection of *disjoint* trees for "almost" every terminal. For each such tree τ, let $A(\tau)$ be the light agent that connects to the terminal directly through an elementary blue path. These agents $A(\tau)$ are to be removed in the next iteration, and the partial assignment of items (defined by the trees) to these agents is guaranteed to be nice. Each terminal of tree τ can be satisfied by reassigning the private items along the blue path from agent $A(\tau)$. Assuming that we have such a subroutine, we now give a complete description of the iterations.

Description of Iteration i:

1. We are given an initial assignment π of private items.
2. Create a flow network and compute an almost feasible solution.
3. Find a nice assignment for a subset $\mathbf{A}'$ of agents.
4. Remove $\mathbf{A}'$ from the instance and compute a new assignment of private items π'.
5. Go back to step 1, using π' as a new assignment.

Remark: We did not cover details of the two main technical components of the paper, i.e. finding almost feasible solution and turning it into nice assignment. Please refer to [CCK09] for more detail on this. Also, the algorithm given in this note is an over-simplified description of the real algorithm, and many steps are not precisely stated here. For instance, agents $\mathbf{A}'$ that are removed from the problem instance in one iteration may be added back (and possibly removed again) later. The invariants we maintain ensure that the number of terminals goes down in each iteration, and we have a nice assignment to all agents that are not currently in the instance.

4 Conclusions

We conclude with a list of open problems:

1. We have discussed an $\tilde{O}(n^{\epsilon})$-approximation due to [CCK09] which runs in time $n^{O(1/\epsilon)}$, but the current lower bound remains at 2. Improving either the lower bound or upper bound would be interesting.
2. This paper uses an iterative rounding scheme to overcome the integrality gap of the LP. It is interesting to see if similar framework can be used to deal with other problems whose LP relaxations have large LP gap.

References

[AFS08] Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. In *APPROX-RANDOM*, pages 10–20, 2008.

[AS07] Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *STOC*, pages 114–121, 2007.

[BCG09] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *STOC*, pages 543–552, 2009.

[BD05] Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005.

[BS06] Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *STOC*, pages 31–40, 2006.

[CCK09] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *FOCS*, pages 107–116, 2009.

[Fei08] Uriel Feige. On allocations that maximize fairness. In *SODA*, pages 287–293, 2008.

[HSS10] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the lovasz local lemma. In *FOCS*, pages 397–406, 2010.