Quasi-Polynomial Local Search for Restricted Max-Min Fair Allocation

Luká Poláek, Ola Svensson
Presented by Martin Böhm
Approximation and Online Algorithm Seminar, MFF UK

D(Max-min fair allocation**):** Given the set of $\mathcal{R}$ resources and $\mathcal{P}$ players, find an allocation of resources to the players such that the least satisfied player is satisfied as much as possible. Also known as the *Santa Claus problem.*

D(Restricted max-min fair allocation**):** In our setting, while resources have different values to all players (v_{ij} for the resource i-th player and j-th resource) there are only two possible values that can arise for a given player: $v_{ij} = 0$ or $v_{ij} = v_J$.

T(Main theorem**):** For any $\varepsilon \in (0, 1]$ we can find a $\frac{1}{4+\varepsilon}$-approximation algorithm for restricted max-min fair allocation in time $n^{O(\frac{1}{\varepsilon} log n)}$, where $n = |\mathcal{P}| + |\mathcal{R}|$.

O: Our algorithm tries to do allocation for a given T which is said to be the optimum solution. If T is not a feasible solution, it fails and we can use this to apply a binary search on the optimal value of T.

Therefore, for the rest of the talk, we assume T is fixed and it is the optimum solution of the restricted max-min fair allocation problem. For the simplicity of the argument, we also assume $T = 1$ through scaling of the values on input.

Configuration LP

D(Configuration LP**):** The configuration LP has exponentially many conditions, based on every admissible assignment of the resources to a player. More formally, it is the following linear program $CLP(T)$ parametrized by T:

$$\min 0,$$
$$\forall i \in \mathcal{P} : \sum_{C \in \mathcal{C}(i,T)} x_{i,C} \geq 1,$$
$$\forall j \in \mathcal{R} : \sum_{i,C : j \in C, C \in \mathcal{C}(i,T)} x_{i,C} \leq 1,$$
$$x \geq 0.$$

D: The dual of the Configuration LP can be stated as:

$$\max \sum_{i \in \mathcal{P}} y_i - \sum_{j \in \mathcal{R}} z_j,$$
$$\forall i \in \mathcal{P}, \forall C \in C(i,T) : y_i \leq \sum_{j \in C} z_j,$$
$$y, z \geq 0.$$

Local search

We will solve the problem by using a local search algorithm for a special kind of hypergraph matching. $\alpha > 4$ will be our approximation parameter.

D: The ground set of our hypergraph will be $\mathcal{P} \cup \mathcal{R}$. A hyperedge e in our setting will contain a player e_P along with an inclusion-minimal set of resources such that e_P has assigned value at least $1/\alpha$.

D(Thin, fat edges**):** A *fat* edge will be an edge such that it contains only one resource (of size at least $1/\alpha$. A *thin* edge is any non-fat edge.

D(Length**):** The *length* of a fat edge will be set as 0, of any thin edge as 1.

In our scenario, we will try to build an *alternating path tree* on our hypergraph until we hit a certain depth. Suppose that we have already created a partial matching M. The tree is contains two kinds of edges. The first group are the edges we have decided to try to add (denoted as A) and the other group are the edges in M which block the addition of the edges in A – these will be denoted B.

D(Addable edge**):** An edge e is *addable* to the alternating path tree if it is disjoint from any edge in $A \cup B$ and it contains a player that is already in our tree.

D(Blocking edge**):** An edge b in the matching M is *blocking* an edge e if it intersects the edge in a resource.

D(Layers**):** For the sake of the analysis, we decompose the edge sets A and B into classes based on the distance to the root, denoting them as A_0, B_0, A_1, B_1 etc. We also denote A_i^t (B_i^t) to be the subset of edges in A_i (B_i) which are thin, and similarly A_i^f (B_i^f) for fat edges.

Local search algorithm

Input: a partial matching M.

Output: an increased matching, provided T is a feasible solution.

```
1  Find an unmatched player, make it a root of the tree.
2  While an addable edge is within distance $2\log_{(\alpha-1)/3}(|\mathcal{P}|) + 1$:
3      Find an addable edge $e$ of minimum distance from the root.
4      Add $e$ to the tree.
5      If it has blocking edges, add them to the tree also.
6      While $e$ has no blocking edges:
7          Add $e$ to the matching $M$.
8          Remove its parent edge $b$ from the matching $M$.
9          Repeat the procedure for the grandparent of $e$.
10     Remove all edges that are in the same or greater distance as
   the last edge added to the matching.
```

Analysis

O: The algorithm does go through all fat edges first, selecting an edge of distance 1 only after the subtree of distance 0 has been traversed.

L(Runtime**):** For a desired approximation guarantee of $1/\alpha = 1/(4 + \varepsilon)$, the algorithm terminates in time $n^{O(\frac{1}{\varepsilon} \log n)}$.

P: Create a signature vector:

$$(-|A_0|, |B_0|, -|A_1|, |B_1|, \ldots, -|A_{2k}|, |B_{2k}|, \infty).$$

We prove that any operation decreases the lexicographic size of the vector. Since the length of the vector is bounded by the limit in the algorithm, we get the desired runtime bound.

L(Key lemma**):** Let $\alpha > 4$. Assuming that $CLP(T)$ is feasible, if there is no addable edge within distance $2D+1$ from the root for some integer D, then

$$\frac{(\alpha - 4)}{3} \sum_{i=1}^{D} |B_{2i}^t| < |B_{2D+2}^t|.$$

C(Correctness**):** If $\alpha > 4$ and $CLP(T)$ is feasible, there always is an addable edge within distance $2D + 1$ for $D = \log_{(\alpha-1)/3}(|\mathcal{P}|)$.

O: Every blocking edge can be mapped to exactly one vertex of $\mathcal{P}$.

O: The size of each thin edge is at most $2/\alpha$.

O: Any part of a B-edge that is not contained in any A-edge must be of size $1/\alpha$.