An Experimental Study of Fast Greedy Algorithms for Fair Allocation Problems

Le Dang Nguyen[1], Trung Thanh Nguyen[2]

[1] Faculty of Computer Science, Haiphong University, Haiphong, Vietnam
[2] ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam

Correspondence: Trung Thanh Nguyen (thanh.nguyentrung@phenikaa-uni.edu.vn)
Communication: received 25 February 2022, revised 12 April 2022, accepted 20 April 2022
DOI: 10.32913/mic-ict-research.v2022.n2.1032

***Abstract*: This paper is concerned with two salient allocation problems in fair division of indivisible goods, aiming at maximizing egalitarian and Nash product social welfare. These problems are computationally NP-hard, meaning that achieving polynomial time algorithms is impossible, unless P = NP. Approximation algorithms, which return near-optimal solution with a theoretical guarantee, have been widely used for tackling the problems. However, most of them are often of high computational complexity or not easy to implement. It is therefore of great interest to explore fast greedy methods that can quickly produce a good solution. This paper presents an empirical study of the performance of several such methods. Interestingly, the obtained results show that fair allocation problems can be practically approximated by greedy algorithms.**

Keywords: *Fair allocation, exact algorithm, greedy algorithm, mixed-integer linear program,* NP*-hard.*

I. Introduction

In this paper, we study the fair allocation problem, which has shown its growing interest during last decades, with a wide range of real-world applications [1]. In short, this is a combinatorial optimization problem which asks to allocate m *discrete* items amongst a set of n agents (or players) so as to meet a certain notion of fairness. It is assumed that every item is "*indivisible*" and "*non-sharable*", that is, i) it cannot be broken in pieces before allocating to agents, and ii) it cannot be shared by two or more agents. For example, laptops and cell-phones are indivisible items which agents might not want to share with others. An allocation of items to agents is simply a partition of the whole set of items into n disjoint subsets. There are up to n^m such partitions, making the solution space large enough so that an exhaustive search for an optimal solution is impossible.

It now remains to define what a fair allocation is, a concept that is of independent interest in the field of Economic and Social Choice Theory [2, 3]. In general, there are many different ways of defining fairness, depending on particular applications. The most common way is to either use a so-called Collective Utility Function (CUF), which is a function for aggregating individual agents' utilities in a fair manner, or to follow an orthogonal method relying on determining the fair share of agents. Since we are focusing on the first method in this paper, we refer the reader to the paper [4] and the references therein for more details of the second method. Suppose that every agent evaluates the value of items through a *utility function*, which maps each subset of items to a numerical value representing the utility of the agent for the subset. Then, one can define a max-min fair allocation to be the one that maximizes the lowest utility among agents' utilities. This is also known as the Rawls aggregation function [5]. Alternatively, one can also define a max-Nash allocation so as to maximize the product of individual agents' utilities [6]. A more general utility-based fair concept, called rank-weighted utilitarianism, can be found in [7].

Our focus is on the computational problem of computing max-min allocations and max-Nash allocations. This problem belongs to the class of combinatorial NP-hard optimization problems, for which exact algorithms with a polynomial runtime is unlikely to exist [8]. Indeed, computing a max-min allocation remains NP-hard even when there are two agents having identical utility functions, as this special case can be polynomially reduced from partition, a well-known NP-complete problem which asks if one can partition m positive integers into two subsets such that the smallest sum of the integers assigned to any subset is maximized [9]. Due to the hardness result, many attempts have been put on designing approximation algorithms which run in polynomial time and produce near-optimal solution with a provable approximation guarantee. In fact, an α-approximation algorithm ($\alpha \geq 1$) for a

maximization problem can return, for any given probem instance, a solution whose value is always at least $1/\alpha$ times the optimum. Obviously, the closer factor α to 1, the better the approximation. Bezáková and Dani [10] showed that max-min allocation is NP-hard to approximate to within a factor of 2. On the positive side, Asadpour and Saberi [11] used a linear program (LP) relaxation, known as the configuration LP, to obtain a $(\sqrt{n}\log^3 n)$-approximation algorithm, and this result was then improved by Bateni et al. [12] and Chakrabarty et al. [13], independently, to a $O(n^{\epsilon}\log n)$-approximation algorithm in $n^{O(1/\epsilon)})$ time, for any constant $\epsilon > 0$. For the problem of maximizing Nash product social welfare, Nguyen and Rothe [14] was the first to study its approximability and they gave the first $O(m)$-approximation algorithm, which holds even for general *sub-additive* utility functions[1]. Since this paper was published there have been a sequence of studies improving upon this approximation factor [15–17]. The first constant approximation factor was proposed by Cole and Gkatzelis in [18], where they developed a novel technique based on the so-called spending-restricted market equilibrium to attain a factor of 2.889. The state-of-the-art approximation factor of 1.45 is obtained in [17], using a local search based approach. In an opposite direction, Lee [19] proved that there is no approximation factor better than 1.00008, unless P = NP. Narrowing the gap $(1.00008; 1.45)$ remains an open problem.

Better approximation results can be achieved for several restricted setting where the number of agents is upper bounded by a constant, or where agents utility functions are of special structures. For example, there are a fully polynomial time approximation scheme (FPTAS) for a constant number of agents [8], and a PTAS for idential agents [14, 20]. Several other works have dealt with bi-value utility functions or scoring vector-based utility functions, such as [4, 21–24], just to mention a few.

While the aformentioned approximation algorithms can provide good approximate solutions in polynomial time, most of them are mainly of theoretical interest. In fact, these algorithms are complicated, not easy to implement, and slow in practice. This motivates a comprehensive empirical study of greedy algorithms with which are typically very easy to implement with low complexity. To the best of our knowledge, this paper is the first to present a full analysis of the effectiveness of several greedy methods for solving the fair allocation problem. We give a formal description of these algorithms, analyze their theoretical performance, and evaluate their effectiveness through various experiments. The experiment shows that the greedy algorithms are quite efficient in practice in the sense that they can provide good solution in a short amount of time, when compared to exact methods.

[1] A set function u is sub-additive if $u(S \cup T) \geq u(S) + u(T)$ holds for every subsets $S, T \subseteq \mathcal{R}$.

II. Problem model

In this section, we present basic notions of fair division, define our fairness criteria, and model the existence and optimization problems that we study. We consider the basic setting of the fair allocation problem which consists of

- a set of agents $\mathcal{A} = \{a_1, \ldots, a_n\}$,
- a set of items $\mathcal{R} = \{r_1, \ldots, r_m\}$, and
- a utility function $u_i : 2^{\mathcal{R}} \to \mathbb{Z}_+$ for each agent $a_i \in \mathcal{A}$,

where $\mathbb{Z}_+$ denotes the set of nonnegative integer numbers. Let $\mathcal{U} = \{u_1, \ldots, u_n\}$. We call $\mathcal{I} = \{\mathcal{A}, \mathcal{R}, \mathcal{U}\}$ an instance of the fair allocation problem. We assume that every utility function is additive, that is, $u_i(S) = \sum_{r \in S} u_i(\{r\})$ for every $S \subseteq \mathcal{R}$. For easy of presentation, we write $u_i(r)$ instead of $u_i(\{r\})$. Also, assume that the empty set has utility 0 for every agent. A (complete) allocation is a partition $\pi = (\pi_1, \ldots, \pi_n)$ of $\mathcal{R}$, where π_i is the subset assigned to agent a_i, and $\pi_i \cap \pi_j = \emptyset$ for $i \neq j$ and $\cup_{i=1}^{n} \pi_i = \mathcal{R}$. We call π an *incomplete allocation* if $\cup_{i=1}^{n} \pi_i \subset \mathcal{R}$. Also, we say that π is a partial allocation of π' if $\pi_i \subseteq \pi'_i$ for all $i \in \{1, 2, \ldots, n\}$. We focus on max-min allocation and max-Nash allocation, which are defined through the concept of *social welfare*.

An allocation π is said to be max-min (respectively, max-Nash) if it has maximum egalitarian social welfare (respectively, maximum Nash product social welfare). We denote the problems of computing such allocations as Max-Min and Max-Nash. In the next section we will briefly review the complexity as well as algorithmic results for these problems.

Notation: For a positive integer p, we denote $[p]$ as the set $\{1, 2, \ldots, p\}$ of positive integers. Also, $\mathbf{0}$ and $\mathbf{1}$ are vectors with all 0 and 1 coordinates, respectively. Let $\mathbf{e}_i$ denotes the vector with a 1 in the i-th coordinate and 0's elsewhere. Also, we $||.||$ to denote the size of a given set.

Example 1: Consider an instance with three agents and six items, and the agent utility functions are given in the Table 1. One can check that the maximum egalitarian social welfare of the instance is 5, by taking an allocation $\pi = (\{r_5, r_2\}, \{r_4, r_6\}, \{r_3, r_1\})$. If we want to maximize the Nash product social welfare, just take the allocation $\pi' = (\{r_5, r_1\}, \{r_4, r_6\}, \{r_3, r_2\})$, with $sw_N(\pi') = (5.5.7)^{1/3} > sw_N(\pi) = (6.5.5)^{1/3}$.

III. Exact methods

The first and naive method for exactly for solving Max-Min and Max-Nash is exhaustive search which enumerates all possible allocations of m item to n agents and

TABLE I
UTILITIES OF THE AGENTS FOR EXAMPLE 1.

	r_1	r_2	r_3	r_4	r_5	r_6
a_1	1	2	2	0	4	1
a_2	1	1	0	3	1	2
a_3	1	3	4	2	0	2

picks the one of maximum egalitarian social welfare or of maximum Nash product social welfare. As mentioned earlier, this method takes $O(n^m)$ time.

A better method is to model the problems as integer (linear or quadratic) programs which can be solved by commercial solvers such as IBM ILOG CPLEX or Gurobi. A straightforward integer programming (IP) formulation for MAX-MIN can is as follows.

$$\begin{aligned}
\text{(IP)} \quad \max \quad & \min_{i=1}^{n} \left\{ \sum\nolimits_{j=1}^{m} u_i(r_j) \cdot x_{ij} \right\} && (1)\\
\text{s.t.} \quad & \sum\nolimits_{i=1}^{n} x_{ij} = 1, \qquad j \in [m], && (2)\\
& x_{ij} \in \{0,1\}, \quad i \in [n], j \in [m], && (3)
\end{aligned}$$

where the binary variable $x_{ij} = 1$ if item r_j is allocated to agent a_i, and $x_{ij} = 0$ otherwise. IP is not a linear program due to the nonlinearity of the objective function, and thus cannot be solved directly by CPLEX. The idea is to put the objective as a constraint, using an additional (fractional) variable T. As such, IP can be reformulated as the following equivalent Mixed Integer Linear Program (MILP):

$$\begin{aligned}
\text{(MILP)} \quad \max_{T \in \mathbb{R}_+} \quad & T\\
\text{s.t} \quad & \sum\nolimits_{j=1}^{m} u_i(r_j) \cdot x_{ij} \geq T, \ i \in [n],\\
& (2) - (3),
\end{aligned}$$

An integer programming formulation of MAX-NASH is similar to IP, except that the objective is replaced by the product function of agent utilities, i.e.,

$$\left(\prod_{i=1}^{n} \left\{ \sum\nolimits_{j=1}^{m} u_i(r_j) \cdot x_{ij} \right\} \right)^{1/n}. \qquad (4)$$

Again (4) is nonlinear, thus we will not be able to use CPLEX for solving it directly. To address this issue, we propose an equivalent model, called Second-Order Cone Program (SOCP), using the technique introduced by Ben-Tal and Nemirovski [25], that can be handled by CPLEX. To do so, we introduce new variables $z_i = \sum_{j=1}^{m} u_i(r_j) \cdot x_{ij}$ and write

$$\begin{aligned}
\max \quad & \eta\\
\text{s.t} \quad & 0 \leq \eta \leq \left(\prod_{i=1}^{n} z_i \right)^{1/n}, && (5)\\
& (2) - (3).
\end{aligned}$$

Let $k = \lceil \log_2 n \rceil$, the constraint (5) can be represented as

$$\begin{aligned}
& 0 \leq \eta \leq \sqrt{y_1^{k-1} y_2^{k-1}},\\
& 0 \leq y_j^l \leq \sqrt{y_{2j-1}^{l-1} y_{2j}^{l-1}}, \ j \in [2^{k-1}], \ l \in [k-1],\\
& 0 \leq y_i^0 = z_i, \ i \in [n],\\
& 0 \leq y_j^0 = \eta, \ j = n+1, \ldots, 2^k,
\end{aligned}$$

where $\eta, \{y_j^l\}$, $j = 1, \ldots, 2^k, l = 0, \ldots, k-1$, are additional variables. The above formulation is mixed integer SOCP since any constraint of the form $\{u, v, w \geq 0: \ u \leq \sqrt{vw}\}$ is equivalent to $\{u, v, w \geq 0: \ 4u^2 + (v-w)^2 \leq (v+w)^2\}$.

IV. GREEDY METHODS

In this section we describe several greedy methods with low complexity, for solving MAX-MIN and MAX-NASH. Recall that m is assumed to be at least n, otherwise the problem becomes trivial.

1. LPT algorithm

A first and naive allocation strategy is to allocate items, one by one, to agents, until no item available. In fact, the algorithm performs in m steps, at j-th step item r_j is given to an agent of smallest utility among those who have nonzero utility for r_j (tie-breaking arbitrarily). There should be at least one such agent, otherwise we can remove this item r_j from $\mathcal{R}$ without affecting the optimal value. Clearly, the complexity of the algorithm is $O(mn) = O(m^2)$, as $n \leq m$. A formal description is given in Algorithm 1. The algorithm is also known as the LPT rule, which was originally used in the context of the minimum makespan problem [26]. It was shown that such an algorithm yields a $\frac{4n-2}{3n-1}$-approximation algorithm for MAX-MIN [27] and a 1.061-approximation algorithm for MAX-NASH [23], assuming that all agents have the same utility function.

Algorithm 1: LPT

```
Inputs: 𝒜, ℛ, 𝒰.
Outputs: An allocation π.
begin
    for j := 1 to m do
        Assign item r_j to an agent who is worst-off
          among those who have non-zero utility for the
          item (tie-breaking arbitrarily);
    end
    return π;
end
```

Example 2 (continued Example 1): Consider the instance given in Example 1, the LPT algorithm will return $\pi = (\{r_1, r_5\}, \{r_2, r_4, r_6\}, \{r_3\})$. This is neither a max-min allocation nor a max-Nash allocation.

It is natural to ask if the LPT algorithm can theoretically guarantee any bound on the quality of the solution found for non-identical utility functions. We answer this question affirmatively for both MAX-MIN and MAX-NASH.

Proposition 1: The LPT algorithm does not posses an $O(m)$-approximation algorithm for both MAX-MIN and MAX-NASH.

Proof: We prove that LPT cannot give an $\frac{\epsilon}{m}$-approximation for any fixed positive constant $\epsilon < 1$. The proof is by contradiction. Consider an instance with $n = 2$ agents $\{a_1, a_2\}$, $m = 4$ items $\{r_1, r_2, r_3, r_4\}$ and the utilities of agents for items are as follows: $u_1(r_1) = 2, u_1(r_2) = 1, u_1(r_3) = 1, u_1(r_4) = M$, $u_2(r_1) = 0, u_2(r_2) = 3, u_2(r_3) = M, u_2(r_4) = 1$, where M is some positive number $M > 2$. One can easily see that the allocation returns by LPT is $\pi = (\{r_1, r_3\}, \{r_2, r_4\})$, which has the egalitarian social welfare of 4, while the max-min allocation is $\pi^* = (\{r_1, r_4\}, \{r_2, r_3\})$, which has egalitarian social welfare of $2+M$. Suppose that LPT is an $\frac{\epsilon}{4}$-approximation, then it must hold that:

$$4 \geq \frac{\epsilon}{4}(2 + M) \geq \frac{\epsilon}{2} + M\frac{\epsilon}{4}.$$

However, this is not true for $M > \frac{16}{\epsilon}$.

Similarly, $\pi = (\{r_1, r_3\}, \{r_2, r_4\})$ has the Nash product social welfare of 4, and $\pi^* = (\{r_1, r_4\}, \{r_2, r_3\})$ has the Nash product social welfare of $\sqrt{(2 + M)(3 + M)} > 4$, for $M > 2$. If LPT is an $\frac{\epsilon}{4}$-approximation, then we have that

$$4 \geq \frac{\epsilon}{4}\sqrt{(2 + M)(3 + M)} \geq \frac{\epsilon}{4}\sqrt{5 + 5M + M^2},$$

which again is not true for $M > \frac{16}{\epsilon}$.

2. Graph matching greedy algorithm

In [14] the authors proposed a graph-matching greedy algorithm (GMG) for computing an allocation of maximum Nash product social welfare. Let $G = (V_G, E_G, w)$ be a weighted bipartite graph, where $V_G = \mathcal{A} \cup \mathcal{R}$ is vertex sets including agent-vertices and of item-vertices. Each $e = (a_i, r_j) \in E$ has a weight that equals to the utility of agent a_i for item r_j, i.e., $w(e) = u_i(r_j)$. The basic idea of GMBA is to find a matching of the graph G such that the product of weights of the edges is maximized. Then, for each pair of agent-item in the matching, allocate item to its matched agent, while the remaining unmatched items are assigned arbitrarily to agents. It was prove in [14] that such an algorithm can guarantees an $O(m)$-approximation. Recently, Garg et al. [28] have modified this algorithm by repeating the matching algorithm for the remaining items several times until no items available. A formal description is given in Algorithm 2.

Basically, Algorithm 2 works in rounds, in each round n items are chosen to be allocated, one item per agent. By adding dummy items of zero utility if needed, one can assume that m is a multiple of n, that is, $m = \ell n$, for some positive integer ℓ. Hence, the number of rounds in the algorithm is ℓ. We now explain in detail how items are allocated in each round, assuming that the objective is to maximizing the Nash product social welfare. The case of the egalitarian social welfare is slightly different and is discussed later.

In the first round of the algorithm, we find a max-prod matching of G (see Definition 1), denoted as $\mathcal{M}^1$, and allocate items to agents based on this matching to obtain an incomplete allocation π^1. By removing items that have been allocated and edges in the matching, we obtain a subgraph of G. In the next round, a straightforward idea is to repeat what we have done in the first round to the subgraph and update the allocation. However, note that agents have already received a subset of items in the previous round, thus it would be fairer if the matching is done in such a way that maximizing the product of agent utilities. To this end, we replace the weight function of G by the new one which adds a utility $u_i(\pi_i^1)$ to the weight of every edge incident to agent-vertex a_i, for all $i \in [n]$. We apply this replacement for all the ℓ rounds.

For the case of max-min allocations, in each round of Algorithm 2 we find a max-min matching instead of a max-product matching. That is, we find a matching such that the minimum of weights of the edges in the matching is maximized (see Definition 1).

Definition 1: A maximum matching $\mathcal{M}$ of a weighted bipartite graph G is called

- a *max-min matching* if $\min_{e \in \mathcal{M}} w(e) \geq \min_{e \in \mathcal{M}'} w(e)$ for any other maximum matching $\mathcal{M}'$ of G;
- a *max-product matching* if $\prod_{e \in \mathcal{M}} w(e) \geq \prod_{e \in \mathcal{M}'} w(e)$ for any other maximum matching $\mathcal{M}'$ of G.

Algorithm 2: Graph-matching based Greedy (GMBA)

Inputs: $\mathcal{A}, \mathcal{R}, \mathcal{U}$.
Outputs: An allocation π.
begin
  $k \leftarrow 1$; $G^k \leftarrow G$; $w^k \leftarrow w$;
  $\pi^k \leftarrow (\emptyset, \ldots, \emptyset)$;
  while $k > \ell$ **do**
    Find a max-product matching $\mathcal{M}^k$ of G^k

$$\mathcal{M}^k \leftarrow \{(a_1, r^k_{j_1}), \ldots, (a_n, r^k_{j_n})\}$$

    Allocate items to agents based on $\mathcal{M}^k$

$$\pi^k_i \leftarrow \pi^k_i \cup \{r^k_{j_i}\}, \quad \text{for } i \in [n]$$

    $k \leftarrow k + 1$;
    Define $G^k = (V_{G^k}, E_{G^k}, w)$

$$V_{G^k} \leftarrow V_{G^{k-1}} \setminus \{r^k_{j_1}, \ldots, r^k_{j_n}\}$$
$$E_{G^k} \leftarrow E_{G^{k-1}} \setminus \mathcal{M}^k$$

    $w^k((a_i, r_j)) \leftarrow u_i(\pi^{k-1} \cup \{r_j\}),\ \forall\ (a_i, r_j) \in E_{G^k}$.
  end
  return π^ℓ
end

It was proved that both the matching can be efficiently found in polynomial time [14], by reducing to the well-known problem of finding a maximum weight matching which can be solved by the Hungarian method [29]. Alternatively, one can formulate the problem as a binary linear program in which the constraint matrix (i.e., the matrix defining the system of linear constraints) is totally unimodular[2]. It is well-known that such a program can be efficiently solved in $O(m^{2.5}L)$ time[3] [30], where L is the number of binary bits encoding the problem instance. For the sake of completeness, we provide an ILP model for the maximum weighted matching problem in Appendix.

Lemma 1 ([14]): Given a weighted complete bipartite graph G, a max-min matching and a max-product matching of G can be found in $O(m+n)^3$ time and $O((m+n)^3 L)$, respectively, where L is the number of binary bits encoding the problem instance.

Example 3 (continued Example 1): Consider the instance given in Example 1, the LPT algorithm will return $\pi = (\{r_1, r_5\}, \{r_2, r_4, r_6\}, \{r_3\})$. This is neither a max-min allocation nor a max-Nash allocation.

The Algorithm 2 runs in $\ell = m/n$ steps, and at each step $k \in [\ell]$ we run Hungarian method (or solve a BLP) to find a maximum weighted matching on a graph G^k having n agent-vertices and $m - (k-1)n$ item-vertices. Then the number of vertices of G^k at step k is equal to $(\ell + 2 - k)n$. Therefore, the overall complexity of Algorithm 2 is

$$O(n^3 \sum_{k=1}^{\ell} (\ell + 2 - k)^3) = O(n^3 \ell^4) = O(m^4/n),$$

where the second equality is due to the equality $\sum_{i=1}^{p} i^3 = (\sum_{i=1}^{p} i)^2 = \frac{1}{4}p^2(p+1)^2$, whose proof can be derived using induction on p.

Theorem 1 ([28]): Algorithm 2 is an $O(n)$-approximation algorithm for MAX-NASH.

It is not very clear if Algorithm 2 can give also an $O(n)$-approximation algorithm for MAX-MIN.

3. Picking sequences

Picking sequence has been known as simple protocol for distributing items among agents without eliciting the agents' utility functions first. This can be also seen as a decentralized approach in which agents communicate with each other to reach a final allocation. In particular, agents are asked to choose items one after the other, according to a predefined order, also called *picking policy*. Formally, a picking policy is a sequence $\sigma = (\sigma_1 \ldots \sigma_m) \in \{1, \ldots, n\}^m$, where at each step, agent a_{σ_i} picks her most preferred item among those remaining. For a formal definition of an allocation induced by a picking policy we refer to the paper by Bouveret and Lang [31]. In this paper, we consider two common policy types:

- *balanced picking policy* $(12\ldots n12\ldots n\ldots)$: agents pick items in rounds. In the first round, agents take turn to pick item: agent 1 first, then agent 2, then agent 3, and so on. Agent n is the last agent who picks item. In the next rounds, we repeat this item picking procedure until all items are allocated.
- *fair picking policy* $(1\ldots nn\ldots 1\ldots)$: according to this policy the orders in which agents take turn to pick item are different between "odd rounds" and "even rounds". In fact, in the first round, the order is the same as the one in the balanced picking policy. However, it is reversed in the second round. That is, the agent n is the first to pick item, then agent $n-1$, and so on.

Example 4 (continued Example 1): Consider the instance given in Example 1, one can see that $\pi = (\{r_5, r_2\}, \{r_4, r_6\}, \{r_3, r_1\})$ will be returned by the balanced picking policy, while $\pi' = (\{r_5, r_1\}, \{r_4, r_6\}, \{r_3, r_2\})$ will be returned by the fair picking policy.

We denote by BPG and FPG the algorithm 3 when a balanced picking policy and a fair picking policy are used, respectively. Like the LPT algorithm, these algorithms

[2] An integer matrix A is totally unimodular if if every square submatrix has determinant 0, +1 or −1. Another sufficient condition for the totally unimodularity is that no more than 2 nonzeros are in each column.

[3] To be precisely, the running time of Lee and Sidford's algorithm [30] is $\tilde{O}((nnz(A) + d^2)\sqrt{d}L)$, where $nnz(A)$ is the number of non-zero elements in A, and d is the number of A's rows.

Algorithm 3: Picking-sequence based Greedy

Inputs: $\mathcal{A}, \mathcal{R}, \mathcal{U}$ and a policy $\sigma = (\sigma_1 \dots \sigma_m)$
Outputs: An allocation π
begin
  $\tilde{\mathcal{R}} \leftarrow \mathcal{R}$;
  for $j := 1$ to m **do**
    Ask agent σ_j to pick one item from $\tilde{\mathcal{R}}$
    Let r be the item picked by σ_j and $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \setminus \{r\}$
  end
  Let π be the obtained allocation;
  return π;
end

do not result in an $O(m)$-approximation, as shown in the proposition below.

Proposition 2: BPG and FPG algorithms do not posses an $O(m)$-approximation algorithm for both MAX-MIN and MAX-NASH.

Proof: We give a proof for FPG only, as the case of BPG can be treated similarly. The proof is by contradiction. Consider an instance with $n = 2$ agents $\{a_1, a_2\}$, $m = 2$ items $\{r_1, r_2\}$ and the utilities of agents for items are as follows: $u_1(r_1) = 1, u_1(r_2) = 1$, $u_2(r_1) = 1, u_2(r_2) = M$, where M is some positive number $M < 1$. By the FPG algorithm, agent a_1 is the first to pick item, yielding to an allocation π, which has the egalitarian social welfare of M, while the max-min allocation is $\pi^* = (\{r_2, r_1\})$, which has egalitarian social welfare of 1. Suppose that FPG is an $\frac{\epsilon}{2}$-approximation, then it must hold that:

$$M \geq \frac{\epsilon}{2} \cdot 1 = \frac{\epsilon}{2}.$$

However, this is not true for $M < \frac{\epsilon}{2}$.

Similarly, π has the Nash product social welfare of $\sqrt{M}$, and π^* has the Nash product social welfare of $1 > \sqrt{M}$, fro $M < 1$. If LPT is an $\frac{\epsilon}{2}$-approximation, then we have that

$$\sqrt{M} \geq \frac{\epsilon}{2} \cdot 1 = \frac{\epsilon}{2},$$

which again is not true for $M < \frac{\epsilon}{4}$.

V. EXPERIMENTS

In this section, we carry out comprehensive computational tests to evaluate the performance of the four greedy algorithm: LPT, GMG, FPG and BPG. In particular, we compare the solution returned by each greedy algorithm with that produced by the exact integer programming method (presented in Section III). We make use of several kinds of problem instances in the experiment, which are specified in the following.

For the evaluation purpose, we propose studying different instance sizes with a varying number of agents $n \in \{4, 8, 12, 16, 20\}$ and items $m = \{2n, 4n, 6n, 8n, 10n\}$. The largest instances are those of 20×200, which is a considerable size that can be solved by IBM-ILOG CPLEX 20.1. There are 25 combinations of n and m and for each combination we generate 10 instances and average the results. All the algorithms are implemented using Python on an INTEL Core i7, 1.5 GHz, with 16 GB of RAM. The best known solution for each one of instances is obtained by running the MILP model presented in Section III with IBM-ILOG CPLEX solver with a time limit of $T_{\max} = 300s$. The utilities of agents for items are uniformly distributed in the interval $[1, 100]$ as it is common in the literature for fair division [4, 32–34].

We propose studying additional sets of instances with scoring-vector based utilites and idenical utilities. The reason is that greedy algorithms have shown their good performance when applied to these special kinds of instances. It is worth noting that even under this restriction, both Max-Min and MAX-NASH remain NP-hard (see [21, 22]). On the other hand, fair allocation with scoring-vector based utilities has been attracted a considerable attention in the last decade. In this setting, it is given as input a (nonnegative) vector $s = (s_1, \dots, s_m)$ where $s_1 \geq \dots \geq s_m$. Each agent is asked to rank m items from the most preferred to the least preferred ones, and then assigns a utility of s_j to an item ranked at position j, for all $j \in [m]$. In our experiment, we focus on the most common scoring vector, the Borda vector $s = (m, m-1, \dots, 1)$. For example, by using the Borda vector, every agent has a utility of m for the top-ranked item and a utility of 1 for the least preferred item.

The tabulated results for each algorithm will be presented as the relative percentage deviation (RPD) from the solution obtained by the solver as follows:

$$\text{RPD}(\mathcal{I}) = \frac{\text{OPT}(\mathcal{I}) - A(\mathcal{I})}{\text{OPT}(\mathcal{I})} \times 100,$$

where $A(\mathcal{I})$ is the value returned by a given algorithm A and an instance $\mathcal{I}$, and $\text{OPT}(\mathcal{I})$ is the aforementioned 300 second CPLEX run on the same instance.

We present now the full results for all three types of instances, presented in Tables II-IV. The first observation is that GMG outperforms other three algorithms LPT, FPG and BPG for both the problems Max-Min and MAX-NASH, no matter what the instance type is used. While the LPT algorithm has a worse performance than the others in all cases, FPG and BPG give a quite similar performance (except the identical utilites), which is much better than that of LPT but slightly worse than that of GMG.

Tables II and III show the results for instances where utility functions are uniform randomly generated in the interval $[0, 100]$, and for the instances with the Borda scoring-vector-based utilities. It is observed that there are not much differences in the performance of the algorithms

TABLE II
AVERAGE RELATIVE PERCENTAGE DEVIATIONS FOR THE INTERVAL [0, 100]

		Max-Min				Max-Nash			
n	m	**LPT**	**GMG**	**FPG**	**BPG**	**LPT**	**GMG**	**FPG**	**BPG**
4	8	53.378	16.256	25.575	25.637	37.114	3.651	9.025	8.161
	16	39.221	12.132	15.658	16.171	38.13	2.312	5.137	3.991
	24	42.788	9.972	10.777	12.54	39.586	1.494	2.886	2.696
	32	41.587	8.281	9.084	7.259	38.412	0.893	2.021	2.33
	40	41.411	9.355	8.921	10.862	36.325	1.145	1.927	1.577
8	16	62.511	16.018	33.895	32.781	42.808	1.557	6.173	6.671
	32	53.426	10.571	16.455	19.641	43.38	1.304	3.243	4.822
	48	48.364	9.989	10.592	12.64	44.154	1.187	2.495	2.341
	64	48.285	7.748	9.945	9.455	44.036	0.922	1.858	1.829
	80	45.425	8.345	8.265	8.791	41.651	0.782	1.45	1.397
12	24	63.008	10.81	27.532	29.812	45.122	1.123	6.757	6.78
	48	53.662	7.652	17.43	15.201	48.772	1.162	3.564	2.884
	72	52.859	7.559	9.394	11.635	46.602	0.878	2.457	2.237
	96	50.918	6.249	9.486	8.804	46.022	0.71	1.784	1.393
	120	49.435	7.252	7.602	8.017	45.364	0.711	1.417	1.407
16	32	62.917	10.744	27.871	31.968	50.124	0.858	4.581	4.664
	64	54.707	7.164	15.504	14.564	47.53	0.671	2.406	2.229
	96	51.532	6.534	9.568	11.174	47.359	0.795	2.049	1.856
	128	50.357	5.175	10.086	8.993	47.217	0.528	1.381	1.405
	160	50.57	5.624	6.171	6.254	47.352	0.483	1.083	0.988
20	40	62.483	10.055	32.575	26.79	47.075	0.679	4.314	4.201
	80	57.116	6.582	13.627	13.685	48.345	0.622	2.345	2.508
	120	52.144	5.499	8.719	9.941	48.607	0.628	1.738	1.822
	160	52.171	4.937	7.278	8.283	48.154	0.572	1.55	1.404
	200	51.89	3.79	5.121	7.172	49.502	0.436	1.067	0.9
Average		**51.686**	**8.572**	**14.285**	**14.723**	**44.749**	**1.044**	**2.988**	**2.899**

between these two types of instances. For example, LPT gives average results of around 52% and 45% deviation from the optimal max-min solution and the optimal max-Nash solution, respectively. FPG and BPG give 12-14% for MAX-MIN and these are significantly reduced to less than 3% for MAX-NASH. The best performance is offered by GMG, which gives less than 10% for MAX-MIN and only around 1% for MAX-NASH. One can see that all the greedy algorithms achieve better results when applied to the MAX-NASH problem. This is not very surprising because while MAX-MIN aims to maximizing the utility of the worst-off agents, which may leave a large gap in the utilities between agents, MAX-MIN can well address this issue by simultaneously balancing agents utilities (see also Example 1 which shows a max-min allocation not being a max-Nash allocation).

Finally, we show the results of the identical utilities in Tables IV, which exhibits better performances compared to other two instance-types above. For the problem MAX-MIN, GMG is the best with 3.336%, which is approximately three and six times better than LPT and BPG, respectively, while FPG gives 4.178%, which almost matches the result of GMG. In the case of MAX-NASH, both GMG and FPG obtain very good results of less than 0.1%, while the results for LPT and BPG are 0.752% and 1.278%, respectively, which are also quite impressive when compared to the case of non-identical utilities. Note that the fact that LPT works well for identical utilities has been also theoretically studied in the literature [23, 27].

To get an overall picture of experimental results, Table

TABLE III
AVERAGE RELATIVE PERCENTAGE DEVIATIONS FOR THE BORDA UTILITY FUNCTIONS

		Max-Min				Max-Nash			
n	m	**LPT**	**GMG**	**FPG**	**BPG**	**LPT**	**GMG**	**FPG**	**BPG**
4	8	60.342	13.584	18.939	20.718	40.203	2.377	6.828	3.644
	16	44.802	7.343	10.215	14.565	42.097	1.379	2.968	3.459
	24	46.276	5.283	8.896	10.811	35.66	0.877	1.663	2.847
	32	37.348	4.813	6.27	6.901	35.892	0.775	1.984	1.85
	40	42.593	3.63	5.169	6.686	41.213	1.095	1.554	1.756
8	16	58.459	13.152	27.286	33.238	44.686	0.973	4.026	3.892
	32	50.28	7.544	14.782	14.56	43.658	1.024	3.262	2.486
	48	51.336	4.8	10.195	9.652	45.098	0.882	1.687	1.609
	64	48.201	5.299	7.223	7.211	44.302	0.827	1.702	1.827
	80	46.489	4.125	5.848	6.918	43.465	0.566	1.298	1.376
12	24	60.338	8.451	24.633	25.366	46.358	0.712	3.58	4.255
	48	53.928	5.455	12.522	12.514	47.124	0.932	2.902	2.369
	72	51.867	5.644	8.838	11.998	47.229	0.707	1.498	1.76
	96	52.668	4.042	6.66	7.726	45.297	0.571	1.485	1.472
	120	48.447	4.022	5.405	6.108	46.406	0.501	1.198	1.17
16	32	60.509	7.746	24.777	27.47	46.877	0.878	5.461	4.366
	64	54.402	4.86	15.319	17.302	47.553	0.679	2.306	2.477
	96	51.186	4.609	8.26	10.302	47.974	0.494	1.763	1.858
	128	51.716	4.167	6.52	7.641	46.707	0.529	1.448	1.33
	160	49.757	4.054	6.381	7.608	47.664	0.455	1.028	1.196
20	40	61.573	7.548	25.648	34.687	48.732	0.523	4.387	3.918
	80	55.005	4.209	15.34	14.411	48.847	0.542	1.942	2.024
	120	52.479	3.46	9.595	11.662	46.458	0.435	1.495	1.349
	160	51.436	3.466	6.496	6.579	47.697	0.444	1.206	1.145
	200	51.88	3.208	6.238	6.361	47.879	0.386	0.928	1.105
Average		**51.733**	**5.781**	**11.898**	**13.56**	**45.003**	**0.782**	**2.383**	**2.261**

V presents all three tested instance-types along with all four greedy algorithms. There are two global averages calculated in the table. The first one is the average results for each algorithm of all instance-types, i.e., an average relative percentage deviation from the 300s CPLEX solution for all instances. The second global average contains the three "typical" instance-types: general utilities in $[0, 100]$, borda scoring-vector-based utilities, and identical utilities in $[0, 100]$. From these averages one can remark that GMG manages to beat the other three algorithms LPT, FPG and BPG. However, to be fair, it is noticed that the latter three algorithm are more faster than GMG, in terms of worst-case running time. In particular, GMG uses, on average, less than $1/4$ second of CPU time, whereas FPG and BPG have been run for just around $1/10$ second. It is worth noting that exactly solving our problems by CPLEX takes up to 5 seconds for each instance.

Finally, our experimental results give some interesting observation about how to solve NP-hard problems Max-Min and MAX-NASH practically. While the first problem is theoretically hard to approximated to within a factor of 2, it can be well tackled in practice by, e.g., the greedy algorithm GMG, which can quickly produce a solution whose value is at least 90% that of the optimum. For the second problem, GMG can even give a much better ratio of 99%, compared to the theoretical ratio of 69% ($\approx 1/1.45$ given in [17]). On the other hand, simple greedy strategies such as FPG and BPG seems to be very good alternatives with the loss of no more than 3% in the objective values, for both the problems.

VI. CONCLUSION

We have studied empirically the four greedy algorithms, LPT, GMG, FPG and BPG, for solving two computationally

TABLE IV
AVERAGE RELATIVE PERCENTAGE DEVIATIONS FOR THE INTERVAL $[0, 100]$ AND IDENTICAL UTILITY FUNCTIONS

		Max-Min				Max-Nash			
n	m	**LPT**	**GMG**	**FPG**	**BPG**	**LPT**	**GMG**	**FPG**	**BPG**
4	8	21.605	13.649	13.649	33.454	1.701	0.659	0.659	3.084
	16	14.268	3.059	5.556	17.41	0.638	0.05	0.109	1.149
	24	9.982	1.465	2.985	12.783	0.328	0.014	0.035	0.457
	32	6.589	0.683	2.071	9.32	0.152	0.003	0.009	0.259
	40	5.323	0.409	1.245	8.032	0.114	0.001	0.01	0.174
8	16	25.157	9.365	9.365	38.935	3.008	0.572	0.572	4.846
	32	15.854	3.028	4.597	20.691	0.758	0.047	0.063	1.037
	48	9.603	1.831	2.375	14.001	0.252	0.01	0.007	0.435
	64	7.206	0.988	1.742	10.471	0.118	0.001	0.006	0.262
	80	5.498	0.48	1.294	8.793	0.136	0.001	0.005	0.168
12	24	28.776	11.456	10.42	45.524	2.916	0.233	0.233	4.778
	48	15.791	2.582	4.996	22.859	0.602	0.032	0.034	1.105
	72	10.125	1.269	2.659	15.672	0.283	0.005	0.014	0.504
	96	6.983	0.977	1.674	11.244	0.171	0.002	0.003	0.243
	120	6.287	0.526	1.144	9.137	0.1	0.001	0.003	0.166
16	32	26.993	11.006	11.006	44.936	2.481	0.369	0.369	4.899
	64	14.88	2.764	4.853	24.853	0.638	0.035	0.05	1.072
	96	10.126	1.089	1.966	15.75	0.336	0.004	0.011	0.477
	128	8.024	0.537	1.723	12.129	0.18	0.001	0.005	0.275
	160	6.318	0.473	1.166	9.524	0.086	0.001	0.001	0.167
20	40	28.681	10.023	10.023	45.073	2.611	0.155	0.155	4.454
	80	15.446	2.791	3.508	23.725	0.65	0.028	0.028	1.008
	120	11.546	1.675	2.257	16.221	0.292	0.002	0.008	0.49
	160	7.858	0.853	1.27	11.814	0.121	0.001	0.003	0.261
	200	5.919	0.409	0.899	9.432	0.131	0.001	0.002	0.183
Average		**12.994**	**3.336**	**4.178**	**19.671**	**0.752**	**0.089**	**0.095**	**1.278**

TABLE V
SUMMARY RESULTS FOR ALL INSTANCE TYPES. BOLD (ITALICS) FIGURES INDICATE BEST (WORST) RESULTS, RESPECTIVELY.

	Max-Min				Max-Nash			
Instance types	**LPT**	**GMG**	**FPG**	**BPG**	**LPT**	**GMG**	**FPG**	**BPG**
Interval $[0, 100]$	*51.686*	**8.572**	14.285	14.723	*44.749*	**1.044**	2.988	2.899
Borda utilities	*51.733*	**5.781**	11.898	13.56	*45.003*	**0.782**	2.383	2.261
Identical utilities	12.994	**3.336**	4.178	*19.671*	0.752	**0.089**	0.095	*1.278*
Average	*38.804*	**5.916**	10.12	15.985	30.168	**0.638**	1.822	2.146

NP-hard problems: egalitarian social welfare maximization and Nash product social welfare maximization. It has be proved that LPT and other two picking policy algorithms cannot give any theoretical bound on the quality of solution found. However, based on the simulation results, it is observed that all the studied greedy algorithms can quickly find near optimal solution of very good quality for various instance types, using a short amount of running time. For the future work, it would be very interesting to see if the graph-matching based greedy algorithms GMG can give an $O(n)$-approximation algorithm for MAX-MIN.

VII. Appendix

An ILP model for MWMP

For a bipartite graph $G = (V, E)$, let $\delta(v)$ be the set of edges incident to the vertex $v \in V$. An ILP formulation for MWMP is as follows.

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e x_e \\ \text{s.t} \quad & \sum_{e \in \delta(v)} x_e \leq 1, && v \in V, \\ & x_e \in \{0, 1\}, && e \in E, \end{aligned}$$

where $x_e = 1$ if edge e is in the matching $\mathcal{M}$, and $x_e = 0$, otherwise. Since G is complete, there are mn edges in total, and thus the number of variables x_e is mn. An equivalent matrix form of the above ILP as follows.

$$\begin{aligned} \max \quad & \mathbf{1}^T \cdot x \\ \text{s.t} \quad & Ax \leq \mathbf{1}, \\ & x \in \{0, 1\}^{mn}, \end{aligned}$$

where $x = (x_{ij})$, and $x_{ij} = 1$ if item r_j is assigned to agent a_i, and $x_{ij} = 0$, otherwise. $A \in \{0, 1\}^{(m+n)\times mn}$ is a block matrix of the form

$$\begin{bmatrix} A^{(1)}_{n\times m} & A^{(2)}_{n\times m} & \cdots & A^{(n)}_{n\times m} \\ I_{m\times m} & I_{m\times m} & \cdots & I_{m\times m} \end{bmatrix},$$

where $A^{(i)}_{n\times m}$ is the matrix of size $n \times m$, with all 1-entries in i-th row, and 0-entries in others, and $I_{m\times m}$ is identical matrix of size $m \times m$. It is not hard to see that each column of A contains exactly 2 nonzeros, making A totally unimodular. In addition, the total number of nonzeros of A is $2mn$, resulting in the complexity of $O((mn + (m+n)^2)\sqrt{m+n}L)$ when applying Lee and Sidford's algorithm on G.

Acknowledgement

We would like to thank the reviewers for very helpful comments and suggestions that helped us to improve the presentation of the manuscript. This work was supported by Vietnam National Foundation for Science and Technology Development under Project 102.01-2020.21.

References

[1] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. A. Padget, S. Phelps, J. A. R.-Aguilar, and P. Sousa, "Issues in multiagent resource allocation," *Informatica (Slovenia)*, vol. 30, no. 1, pp. 3–31, 2006.

[2] M. Fleurbaey and F. Maniquet, *A Theory of Fairness and Social Welfare*. Cambridge University Press, 2011.

[3] H. Moulin, *Handbook of Computational Social Choice*, F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, Eds. Cambridge University Press, 2016.

[4] S. Bouveret and M. Lemaître, "Characterizing conflicts in fair division of indivisible goods using a scale of criteria," *Auton. Agents Multi Agent Syst.*, vol. 30, no. 2, pp. 259–290, 2016.

[5] J. Rawls, *A Theory of Justice*. London: Oxford University Press, 1971.

[6] M. Kaneko and K. Nakamura, "The nash social welfare function," *Econometrica*, vol. 47, no. 2, pp. 423–435, 1979.

[7] H. Moulin, *Axioms of cooperative decision making*. Cambridge: Cambridge University Press, 1988.

[8] N. Nguyen, T. T. Nguyen, M. Roos, and J. Rothe, "Computational complexity and approximability of social welfare optimization in multiagent resource allocation," *Auton. Agents Multi Agent Syst.*, vol. 28, no. 2, pp. 256–289, 2014.

[9] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. New York: W. H. Freeman and Company, 1979.

[10] I. Bezáková and V. Dani, "Allocating indivisible goods," *SIGecom Exch.*, vol. 5, no. 3, pp. 11–18, 2005.

[11] A. Asadpour and A. Saberi, "An approximation algorithm for max-min fair allocation of indivisible goods," *SIAM J. Comput.*, vol. 39, no. 7, pp. 2970–2989, 2010.

[12] M. Bateni, M. Charikar, and V. Guruswami, "Maxmin allocation via degree lower-bounded arborescences," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, M. Mitzenmacher, Ed. ACM, 2009, pp. 543–552.

[13] D. Chakrabarty, J. Chuzhoy, and S. Khanna, "On allocating goods to maximize fairness," in *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*. IEEE Computer Society, 2009, pp. 107–116.

[14] T. T. Nguyen and J. Rothe, "Minimizing envy and maximizing average nash social welfare in the allocation of indivisible goods," *Discret. Appl. Math.*, vol. 179, pp. 54–68, 2014.

[15] N. Anari, S. O. Gharan, A. Saberi, and M. Singh, "Nash social welfare, matrix permanent, and stable polynomials," in *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, ser. LIPIcs, C. H. Papadimitriou, Ed., vol. 67. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 36:1–36:12.

[16] R. Cole, N. R. Devanur, V. Gkatzelis, K. Jain, T. Mai, V. V. Vazirani, and S. Yazdanbod, "Convex program duality, fisher markets, and nash social welfare," in *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, C. Daskalakis, M. Babaioff, and H. Moulin, Eds. ACM, 2017, pp. 459–460.

[17] S. Barman, S. K. Krishnamurthy, and R. Vaish, "Finding fair and efficient allocations," in *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, É. Tardos, E. Elkind, and R. Vohra, Eds. ACM, 2018, pp. 557–574.

[18] R. Cole and V. Gkatzelis, "Approximating the nash social welfare with indivisible items," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, R. A. Servedio and R. Rubinfeld, Eds. ACM, 2015, pp. 371–380.

[19] E. Lee, "Apx-hardness of maximizing nash social welfare with indivisible items," *Inf. Process. Lett.*, vol. 122, pp. 17–20, 2017.

[20] G. J. Woeginger, "A polynomial-time approximation scheme for maximizing the minimum machine completion time," *Oper. Res. Lett.*, vol. 20, no. 4, pp. 149–154, 1997.

[21] A. Darmann and J. Schauer, "Maximizing nash product social welfare in allocating indivisible goods," *Eur. J. Oper. Res.*, vol. 247, no. 2, pp. 548–559, 2015.
[22] D. Baumeister, S. Bouveret, J. Lang, N. Nguyen, T. T. Nguyen, J. Rothe, and A. Saffidine, "Positional scoring-based allocation of indivisible goods," *Auton. Agents Multi Agent Syst.*, vol. 31, no. 3, pp. 628–655, 2017.
[23] S. Barman, S. K. Krishnamurthy, and R. Vaish, "Greedy algorithms for maximizing nash social welfare," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018, pp. 7–13.
[24] G. Amanatidis, G. Birmpas, A. Filos-Ratsikas, A. Hollender, and A. A. Voudouris, "Maximum nash welfare and other stories about EFX," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, C. Bessiere, Ed. ijcai.org, 2020, pp. 24–30.
[25] A. Ben-Tal and A. Nemirovski, "On polyhedral approximations of the second-order cone," *Math. Oper. Res.*, vol. 26, no. 2, pp. 193–205, 2001.
[26] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal of Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.
[27] B. Y. Wu, "An analysis of the LPT algorithm for the max-min and the min-ratio partition problems," *Theor. Comput. Sci.*, vol. 349, no. 3, pp. 407–419, 2005.
[28] J. Garg, P. Kulkarni, and R. Kulkarni, "Approximating nash social welfare under submodular valuations through (un)matchings," in *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, S. Chawla, Ed. SIAM, 2020, pp. 2673–2687.
[29] E. A. Dinic and M. A. Kronrod, "An algorithm for the solution of the assignment problem," *Math. Dokl.*, vol. 10, no. 6, pp. 1324–1326, 1969.
[30] Y. T. Lee and A. Sidford, "Efficient inverse maintenance and faster algorithms for linear programming," in *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, V. Guruswami, Ed. IEEE Computer Society, 2015, pp. 230–249.
[31] S. Bouveret and J. Lang, "A general elicitation-free protocol for allocating indivisible goods," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, T. Walsh, Ed. IJCAI/AAAI, 2011, pp. 73–78.
[32] G. Amanatidis, E. Markakis, A. Nikzad, and A. Saberi, "Approximation algorithms for computing maximin share allocations," in *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, ser. Lecture Notes in Computer Science, M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, Eds., vol. 9134. Springer, 2015, pp. 39–51.
[33] I. Caragiannis, D. Kurokawa, H. Moulin, A. D. Procaccia, N. Shah, and J. Wang, "The unreasonable fairness of maximum nash welfare," in *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, V. Conitzer, D. Bergemann, and Y. Chen, Eds. ACM, 2016, pp. 305–322.
[34] H. Aziz, G. Rauchecker, G. Schryen, and T. Walsh, "Algorithms for max-min share fair allocation of indivisible chores," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, S. P. Singh and S. Markovitch, Eds. AAAI Press, 2017, pp. 335–341.

Trung Thanh Nguyen received the M.S. degree in mathematics from the Institute of Mathematics, Vietnam Academy of Science and Technology in 2007, and the Ph.D. degree in computer science from Heinrich Heine University Duesseldorf, Germany, in 2013. He is currently an Associate Professor with the Faculty of Computer Science, Phenikaa University, Vietnam.
Email: thanh.nguyentrung@phenikaa-uni.edu.vn

Le Dang Nguyen received the M.S. degree in computer science from VNU University of Engineering and Technology in 2005, and the Ph.D. degree in applied mathematics from VNU University of Science, in 2016. He is currently a senior lecturer with the Faculty of Computer Science, Haiphong University, Vietnam.
Email: nguyenld@dhhp.edu.vn