Egalitarian Pairwise Kidney Exchange: Fast Algorithms via Linear Programming and Parametric Flow[*]

Jian Li
Tsinghua University
Beijing, China
lijian83@mail.tsinghua.edu.cn

Yicheng Liu
Tsinghua University
Beijing, China
liuyicheng1991@hotmail.com

Lingxiao Huang
Tsinghua University
Beijing, China
huanglx12@mails.tsinghua.edu.cn

Pingzhong Tang
Tsinghua University
Beijing, China
kenshinping@gmail.com

ABSTRACT

We revisit the pairwise kidney exchange problem established by Roth Sonmez and Unver [23]. Our goal, explained in terms of graph theory, is to find a maximum fractional matching on an undirected graph, that Lorenz-dominates any other fractional matching. The Lorenz-dominant fractional matching, which can be implemented as a lottery of integral matchings, is in some sense the fairest allocation and also enjoys the property of being incentive compatible. The original algorithm by Roth et al. runs in time exponential in the size of input. In this paper, we target at designing practically efficient polynomial time algorithms for finding the Lorenz-dominant fractional matching. We start with a conceptually very simple algorithm, coined the water-filling algorithm. The water-filling algorithm is natural and allows us to present a simpler constructive proof that there exists a unique Lorenz-dominant fractional matching. The algorithm can be readily realized by a series of linear programs, each having an exponential number of constraints, and thus can be solved by the ellipsoid algorithm in polynomial time. However, it is known that the ellipsoid algorithm is not efficient in practice. To make the algorithm practical, we propose the second implementation based on a parametric flow computation on a carefully constructed flow network. Notably, the evolution of the parametric flow simulates exactly the water-filling algorithm. The second implementation only consists of a maximum matching computation and a parametric flow computation, both of which admit very efficient algorithms in practice.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; J.4 [**Social and Behavioral Sciences**]: Economics; C.2.2 [**Computer-Communication Networks**]: Network protocols

General Terms

Economics, Theory, Algorithms

Keywords

Kidney Exchange, Parametric Flow, Lorenz-dominant

[*]This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300(301), the National Natural Science Foundation of China Grant 61033001, 61361136003, 61303077, 61202009 and a Tsinghua University Initiative Scientific Research Grant.

Appears in: *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), Lomuscio, Scerri, Bazzan, Huhns (eds.), May, 5–9, 2014, Paris, France.*

1. INTRODUCTION

Over the past decade, designing desirable matching markets on various domains has become a major topic of interests, attracting attentions from fields such as computer science, economics, operation research and medical science.

The design of matching markets presents at least three exciting challenges. The first is acquisition of domain knowledge. This means, as designer, one cannot come up with solutions that cannot be met in practice. Take for example the design of a kidney exchange market [20, 23], domain knowledge includes scale of surgeries that can be conducted simultaneously, types of preferences raised from patients, blood and tissue type information of the pool, cost of compatibility test for each individual as well as constraints imposed by law, etc. Unlike in the US where kidney exchange among three or more patient-donor pairs are logistically feasible; it is quite rare to conduct simultaneously surgeries at this scale in less developed countries such as China. Furthermore, unlike in the US where kidney exchange among three or more patient-donor pairs is legal, the "Regulations of human organ transplantation of China" explicitly requires the relation between recipient and donor must satisfy one of the three conditions: (1) Wife and Husband; (2) Blood-relatives who share an common ancestor up to three generations; (3) Provable relationship built during the treatment of decease. Under this regulation, it is almost impossible to validate the design of a market mechanism where three-way exchange is an option. As a result, the only option for setting up a kidney exchange system in China is pairwise exchange.

The second challenge regards economical incentives and fairness. To build a nationwide kidney exchange market, there are two levels of incentives at stake. At the patient level, is the market designed in a way that each patient benefits from reporting a truthful list of compatible kidneys (see e.g., [20, 23])? At the hospital level, is the market so designed that each hospital finds it most beneficial to report all its available resources [4]? In addition, it is not only desirable but sometimes also necessary to establish certain fairness standard for the market [23, 9].

The third challenge, namely the computational complexity of the market clearing algorithm, is particularly concerned with electronic markets. That is, given certain domain constraints and incentive constraints, whether it is possible to obtain an efficient algorithm that clears a potentially large scale market. It is well-known that, for the kidney exchange problem constrained by cycle length at most 3, the clearing problem is NP-Hard [1]. However, via a nice integer linear programming formulation, it is possible to clear the current US nationwide market using CPLEX and the algorithm has already been fielded and improved over the years [1].

As mentioned, we investigate the problem of pairwise kidney exchange. In particular, we are confined to the identical setting as in [23]. A general kidney exchange can be encoded as an undirected graph $\mathcal{G} = (V, E)$, where a vertex in V represents a patient-donor pair and each edge in E represents a possible pairwise kidney exchange, where the patient on one side of the edge receives a kidney from the donor on the other side, and vice versa. A *clearing algorithm* returns a maximum (cardinality) matching [1]. As is the case in Roth et al. [23], we allow for an algorithm to return a lottery over all possible maximum matchings. Under a lottery, a patient's utility can be naturally interpreted as the probability of being matched. With utility defined this way, a notion of distributive fairness, called *Lorenz-dominant utility*, can be defined as a utility profile u that, when compared with any other utility profile v, the sum of the smallest k individual utilities in u is no less than that of v, for all $1 \leq k \leq |V|$. The goal is to find the Lorenz-dominant utility profile and the corresponding lottery.

To this end, Roth et al. proposed the well-known *egalitarian mechanism* (also denoted as the RSU algorithm). The mechanism returns a lottery that is Lorenz-dominant. The mechanism is also proved to be *strategy-proof*: it is always in every patient's best interest to report truthfully the list of compatible kidneys. Hence, the RSU algorithm perfectly addresses the first two challenges mentioned above. However, it is not clear that their algorithm resolves the third issue, namely, computational complexity, as the RSU algorithm runs in exponential time on the size of the graph thus is impractical to be deployed directly in large market [2].

1.1 Our Contribution

In this paper, we take the egalitarian mechanism one step further by addressing the third challenge in a satisfactory way. In particular, we obtain a polynomial-time algorithm, that is also quite efficient in practice, for finding the Lorenz-dominant utility profile. Although our algorithm is closely tied with the RSU algorithm (will explain in Section 5), what we do is not to modify the RSU algorithm and design a polynomial time implementation out of it. Indeed, this direction was our initial attempt which did not seem to be easy and successful (at least for us). Instead, we start all over by proposing a more conceptually intuitive algorithm, coined the water-filling algorithm. The RSU algorithm can be derived as a particular implementation of it (not an efficient one though). The road-map of the paper and our contribution can be summarized as follows:

1. (Section 3) We first propose the water-filling algorithm for finding a Lorenz-dominant utility profile. The correctness of the algorithm can be shown using the representation of polymatroid and the submodularity of the associated rank function, This serves as a constructive proof of the existence of the Lorenz-dominant utility profile among all achievable utility profiles (the set of achievable utility profiles forms a polymatroid). The water-filling algorithm can be readily realized by solving a series of linear programs, each having an exponential number of linear constraints. Hence, in theory, we can apply the ellipsoid algorithm to solve the problem in polynomial time.

2. (Section 4) It is known that the ellipsoid algorithm is inefficient in practice. So we target at designing a practically efficient implementation of the water-filling algorithm. For this purpose, we construct a flow network based on the Gallai-Edmonds decomposition (which can be found in $O(n^3)$ time) [11] such that each feasible flow profile on the network corresponds to a feasible utility profile for the original problem. We adopt a parametric flow computation [12] on the constructed network. We show the evolution of the parametric flow exactly simulates the series of linear programs corresponding to the water-filling algorithm. Note that the parametric flow can be computed in $O(nm \log(n^2/m))$ time [12] (only a constant times of a single maximum flow computation).

3. (Section 5) We establish an explicit relationship between our parametric flow algorithm and the RSU algorithm. In particular, we show the evolution of the minimum cut in the parametric network exactly simulate the execution of the RSU algorithm. We can view this relationship as an alternative (and simpler) proof that the RSU algorithm actually returns the Lorenz-dominant utility profile (the original proof in [23] is rather involved).

4. (Section 6) We conduct extensive experiments in several simulated data sets and our results demonstrate the efficiency of our algorithm.

1.2 Other Related Work

Roth, Sonmez and Unver addressed the first two challenges mentioned previously, proposed several mechanisms tailored for different practical scenarios and fielded a nationwide kidney exchange market [20, 23, 21, 22]. In a follow-up work, Yilmaz gave a characterization of all egalitarian kidney exchange mechanisms for an extended setting including both exchanges and chains [30]. Our work is built and

[1]This implies patients' preferences are 0-1: a patient receives utility of 1 for being matched to any compatible kidney and 0 otherwise. 0-1 preference is justified to be a practical model of preference by Roth et al. [23]. For general preference, a clearing algorithm returns a maximum weighted matching for a weighted graph.

[2]In fact, the running time of the RSU algorithm is exponential in the number of odd components in the Gallai-Edmonds decomposition of $\mathcal{G}$. The number of odd components is $|V| - 2|\text{maximum matching}|$, which can be quite large in large networks.

improves upon their work of egalitarian pairwise kidney exchange [23], by addressing the third challenge — the time-complexity of clearing algorithm.

Since then, there has been a surge of research on theoretical and empirical evaluation of clearing algorithms for kidney exchange [13, 26, 1, 7]. More recently, chains, a sequence of exchange that starts with an altruistic donor, has been investigated [8, 3]. Furthermore, incentives issues of a kidney exchange market have been studied in various scenarios [2, 4, 29].

Finding a maximum matching is a central problem in combinatorial optimization. The first strongly polynomial time algorithm is discovered by Edmonds in his classic paper [10]. Since then, the problem has been studied extensively over several decades and several improvements over the time complexity have been obtained (see [25] for a list of such results). The current best results are an $O(\sqrt{|V|}|E|)$ algorithm in [17] and an $O(|V|^{\omega})$ algorithm in [18] where $O(|V|^{\omega})$ is the time required for multiplying two n by n matrix. Cunningham and Marsh [6] gave a linear programming characterization of the maximum matching polytope.

Technically, perhaps the most related work is [28], which shows the existence of a Lorenz-dominant vector among all integral or fractional points in a generalized polymatroid. However, Tamir's elegant proof does not directly suggest an practically efficient algorithm (for the fractional case). On the other hand, the proof of Theorem 1 is associated with the water-filling algorithm, based on which we can obtain an efficient implementation. In our opinion, our proof, although quite different from Tamir's, is as succinct. Our water-filling algorithm is also similar to the *simultaneous eating* algorithm proposed in [5]. However, their objective was not to find the Lorenz-dominate solution. Finally, we would like to mention that the notion of Lorenz-domination (or majorization) have been used in several resource allocation problems to characterize fairness, see e.g., [15, 27].

1.3 Problem Description

Consider an undirected graph $\mathcal{G}(V,E)$ ($|V|=n$, $|E|=m$), where each vertex stands for a patient-donor pair and each edge stands for a possible pairwise kidney exchange, where the patient on one side of the edge receives a kidney from the donor on the other side and vice versa. A matching μ is a collection of vertex disjoint edges. A maximum matching is a match with the largest possible cardinality. Let $\mathcal{U}$ be the set of all matchings over $\mathcal{G}$.

A *matching lottery* $\ell = (\ell_\mu)_{\mu\in\mathcal{U}}$ is a probability distribution over $\mathcal{U}$. For each matching $\mu \in \mathcal{U}$, $\ell_\mu \in [0,1]$ is the probability of choosing matching μ in lottery ℓ, and $\sum_{\mu\in\mathcal{U}} \ell_\mu = 1$. A matching lottery ℓ can be also viewed as *a fractional matching* which is defined to be a convex combination of several integral matchings. Let $\mathcal{L}$ be the set of matching lotteries. Given $\ell \in \mathcal{L}$, define the utility x_i^ℓ of vertex i to be the total probability that i is matched , i.e., $x_i^\ell = \sum_{\mu\in\mathcal{U}, i \text{ is matched in } \mu} \ell_\mu$. Define the utility profile induced by lottery ℓ to be the vector $x^\ell = (x_i^\ell)_{i\in V}$. Let $\mathcal{P} = \{x^\ell\}_{\ell\in\mathcal{L}}$ be the set of all feasible utility profiles.

Example 1.1. *Consider the graph G with four vertices u,v,w,z and three edges $(u,v),(v,w)$ and (w,z). $\mathcal{U}$ consists of five matchings: $\mu_1 = \{(u,v),(w,z)\}, \mu_2 = \{(u,v)\}, \mu_3 = \{(v,w)\}, \mu_4 = \{(w,z)\}$ and $\mu_5 = \emptyset$ (i.e., the empty matching). Define a matching lottery ℓ such that $\ell_{\mu_1} = 0.5$ and $\ell_{\mu_3} = 0.5$. Then, $x_u^\ell = \ell_{\mu_1} + \ell_{\mu_2} = 0.5$ and $x_v^\ell = \ell_{\mu_1} + \ell_{\mu_2} + \ell_{\mu_3} = 1$. The utility profile of ℓ is $(x_u^\ell, x_v^\ell, x_w^\ell, x_z^\ell) = (0.5, 1, 1, 0.5)$.*

For any utility profile $x \in \mathcal{P}$, sort individual utilities in increasing order. We use $x_{(i)}$ to denote the ith smallest entry in x (ties are broken in an arbitrary but fixed manner).

Definition 1. ***(Lorenz-domination)*** *A utility profile $x \in \mathcal{P}$ Lorenz-dominates [3] another utility profile $y \in \mathcal{P}$ if*

1. *For each $t \in [n]$, $\sum_{s=1}^t x_{(s)} \geq \sum_{s=1}^t y_{(s)}$.*
2. *For some $t \in [n]$, $\sum_{s=1}^t x_s > \sum_{s=1}^t y_{(s)}$.*

Definition 2. ***(Lorenz-dominant Utility Profile)*** *A utility profile is* Lorenz-dominant *if and only if it Lorenz-dominates every other utility profile in $\mathcal{P}$.*

The Lorenz-dominant utility profile corresponds to the fairest allocation in some sense among all possible allocations of the available resource. Moreover, it is not hard to see that the Lorenz-dominant condition implies Pareto efficiency. The mechanism is also proved to be *strategy-proof*: it is every patient's best strategy to report truthfully the list of compatible kidneys. The main goal of this paper is to design an efficient algorithm for finding the Lorenz-dominant utility profile in $\mathcal{P}$.

Example 1.2. *Clearly, the lottery ℓ defined in Example 1.1 does not produce the Lorenz-dominant utility profile. The Lorenz-dominant utility profile is in fact $(1,1,1,1)$, corresponding to the lottery in which μ_1 is assigned probability 1 and others 0. Consider another graph with vertices (u,v,w) and edges $(u,v),(v,w),(w,u)$ (i.e., a triangle). The Lorenz-dominant utility profile is $(2/3,2/3,2/3)$, corresponding to the lottery in which each nonempty matching is assigned probability $1/3$.*

2. PRELIMINARIES

We first recall basic definitions and properties on matroids and polymatroids. Our discussion will heavily rely on these combinatorial objects. For more information about the theory of matroids and polymatroid, see, e.g., [25].

Definition 3. ***(Matroid and The Rank Function)*** *A finite* matroid *$\mathcal{M}$ is a pair $(V,\mathcal{I})$, where V is a finite set (called the ground set) and $\mathcal{I}$ is a collection of subsets of V. Each element in $\mathcal{I}$ (a subset of V) is called an* independent set. *Moreover, $\mathcal{M} = (V,\mathcal{I})$ satisfies the following three properties: (1) $\emptyset \in \mathcal{I}$; (2) if $A \subseteq B$ and $B \in \mathcal{I}$, then $A \in \mathcal{I}$; (3) for all $A, B \in \mathcal{I}$ with $|A| > |B|$, there exists an element $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$.*

The rank function rank : $2^V \to \mathbb{Z}^+ \cup \{0\}$ *of the matroid $\mathcal{M}$ is defined to be* $\text{rank}(S) = \max_{I\subset S, I\in\mathcal{I}} |I|$ *for all $S \subseteq V$.*

The (inclusion-wise) maximal elements of $\mathcal{I}$ are called the bases. *From the definition, it is easy to see that all bases have the same cardinality. Obviously, for any base B,* $\text{rank}(B) = |B| = \text{rank}(V)$. □

Definition 4. ***(Submodular Function)*** *A set function $f : 2^V \to \mathbb{R}$ is a em submodular function if*

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T), \ \ \forall S, T \subseteq V.$$

[3]Such relation is also commonly termed as "super-majorization" in mathematics community [16]. We decide to inherit the terminology of [23] in this paper.

It is well-known that the rank function rank of a matroid $\mathcal{M}$ is submodular.

DEFINITION 5. ***(Polymatroid)*** *Let $f : 2^V \to \mathbb{R}$ be a submodular set function. The* polymatroid *associated with f is defined to be the following polyhedran (it is a polyhedran since it is defined by a set of linear inequalities)*

$$\mathcal{P}_f = \{x \in \mathbb{R}^V \mid x \geq \mathbf{0}, x(S) \leq f(S), \forall S \subseteq V\}$$

where we use $x(S)$ to denote $\sum_{v \in S} x_v$. $\square$

We now recall several well-known facts of matroids and polymatroids. For any $S \subset V$, we say vector $x = \{x_1, \ldots, x_{|V|}\}$ is the *characteristic vector* of S if $x_v = 1$ for $v \in S$ and $x_v = 0$ for $v \notin S$. We summarize them in the following lemma.

LEMMA 1. *Consider the polymatroid $\mathcal{P}_{\text{rank}}$ associated with the rank function* rank *of matroid $\mathcal{M}(V, \mathcal{I})$. Then, we have that $\mathcal{P}_{\text{rank}}$ coincides with the convex hull of the characteristic vectors of all independent sets of $\mathcal{M}$, i.e.,* ConvexHull$\{x_I \mid I \in \mathcal{I}\}$. *In other words, $\mathcal{P}_{\text{rank}}$ fully characterizes the matroid $\mathcal{M}$ (this is why it is called polymatroid) and each vertex of (a.k.a. extremal point) of $\mathcal{P}_{\text{rank}}$ corresponds to a characteristic vector of some independent set in $\mathcal{I}$. Any point in $\mathcal{P}_{\text{rank}}$ can be written as a convex combination of several characteristic vector of independent sets in $\mathcal{I}$.*

The following lemma about matchings is particularly useful for us (see e.g., [25], also proved in [23]).

LEMMA 2. *For an undirected graph $\mathcal{G}(V, E)$, define $\mathcal{I} = \{I \mid I$ is the set of vertices that are matched by some matching of $\mathcal{G}\}$. Then, $\mathcal{M}(V, \mathcal{I})$ is a matroid (called* matching matroid*). A base in $\mathcal{M}$ is the set of vertices that are matched by some maximum matching.* rank(S) *can be interpreted as the maximum number of vertices in S that can be matched in any matching.*

From now on, rank refers to the rank function of the matching matroid for $\mathcal{G}(V, E)$ and $\mathcal{P}_{\text{rank}}$ refers to the polymatroid associated with this particular rank function. Recall that a feasible utility profile can be understood as a fractional matching which is a convex combination of integral matchings. Therefore, $\mathcal{P}_{\text{rank}}$ is exactly the set of feasible utility profiles.

3. THE WATER-FILLING ALGORITHM VIA LINEAR PROGRAMMING

We first describe the so-called *water-filling algorithm* for finding the Lorenz-dominant utility profile in the polymatroid $\mathcal{P}_{\text{rank}}$ associated with the submodular function rank : $2^V \to \mathbb{R}$. The initial value of $x \in \mathcal{P}_{\text{rank}}$ is an all-zero vector. The algorithm starts by increasing uniformly the values of all coordinates of x until some constraint in $\mathcal{P}_{\text{rank}}$ is tight (i.e., the constraint $x(S) \leq \text{rank}(S)$ holds with equality). We fix those x_i's that participate in the tight constraint. Next, we continue increasing the rest of coordinates until we hit another tight constraint. We fix those tight x_i's and repeat the above process, until no coordinate can be further increased.

Linear programs : The above water-filling process can be readily implemented by solving a series of linear programs.

1. Step 1. Solve the linear program

   $$\mathsf{LP}_1 = \{\text{max. } \lambda_1, \text{ subject to } x_v = \lambda_1, \forall v \in V, \ x \in \mathcal{P}_{\text{rank}}\}.$$

   We call an element $v \in V$ *a tight element* if x_v participates in some tight constraint in $\mathcal{P}_{\text{rank}}$[4]. Let D_1 be the set of tight elements.

2. In general, at Step k, we solve the linear program

   $$\mathsf{LP}_k = \{\text{max. } \lambda_k, \text{ subject to } x_v = \lambda_j, \forall v \in D_j \forall j < k, \\ x_v = \lambda_k, \forall v \in V \setminus \cup_{j<k} D_j, \ x \in \mathcal{P}_{\text{rank}}\}.$$

   $D_k \leftarrow$ the set of elements that become tight in step k.

3. The algorithm return $x = \{x_v = \lambda_j \text{ for } v \in D_j\}_v$ if $\cup_{j=1}^{k} D_j = V$.

Polynomial time algorithms (in theory) Each linear program LP_i has an exponential number of constraints. So we can use the ellipsoid algorithm to solve such linear program (see e.g., [25]). To be able to apply the ellipsoid algorithm, we need a polynomial time *separation oracle* which, given a point $x \in \mathbb{R}^V$, can decide in polynomial time whether x is feasible, and in case x is infeasible, can return a violated constraint. In our problem, the separation oracle can be implemented using a submodular function minimization algorithm, which runs in polynomial time, (e.g., [14, 19]). Although the ellipsoid algorithm and the existing submodular function minimization algorithms runs in polynomial times in theory, they are known to be impractical for even medium-sized applications. An alternative way is to use the Cunningham-Marsh formulation of the matching polytope or solve a quadratic program. We defer the details to the full version of the paper. However, both of them also require the ellipsoid algorithm, which is inefficient in practice.

Proof of the Correctness We are ready to prove the first important theorem of the paper, that is the water-filling algorithm can indeed produce the Lorenz-dominant vector. Our proof is an alternative proof of the main result of [28] for polymatroid and can be seen as a constructive version of that proof[5]. Our proof, quite different from Tamir's, leverages the submodularity of the rank function of the matroid and maybe of interest in its own right.

THEOREM 1. *The vector x found by the water-filling algorithm is the* unique *Lorenz-dominant vector in $\mathcal{P}_{\text{rank}}$.*

Proof: We use the fact that x is Lorenz-dominant if and only if x is the minimizer of the function $G(y) \sum_i g(y_i)$ for *all* convex decreasing functions g over $\mathcal{P}_{\text{rank}}$ [16]. It suffices to show that for any $y \in \mathcal{P}_{\text{rank}}$ such that $y \neq x$, $\sum_v g(y_v)$ is not the minimum over $\mathcal{P}_{\text{rank}}$ for any convex g. In particular, we show that there are two coordinates i and j such that (1) $y_i < y_j$ and (2) the vector $y' = \{y_1, \ldots, y_i + \epsilon, \ldots, y_j -$

[4]In other words, increasing x_v would violate some constraint in $\mathcal{P}_{\text{rank}}$ when other x_us for $u \neq v$ are fixed. In many linear programming algorithms, we can easily detect such tight elements. Another way to test the tightness of an element is to solve a closely related linear program in which we fix other x_us and maximize x_v.

[5]In fact, Tamir [28] provided a constructive proof for the fact that there is a Lorenz-dominant vector among all *integral* vectors in a generalized polymatroid. It was mentioned that the fractional case is an extension of the integral case. This step is not fully constructive.

$\epsilon, \dots, y_n\}$ is in $\mathcal{P}_{\text{rank}}$ for some small $\epsilon > 0$. Note that from (1) and (2), we have that for any convex g and small $\epsilon > 0$,

$$\begin{aligned} G(y') - G(y) &= \sum_k g(y'_k) - \sum_k g(y_k) \\ &= g(y_i + \epsilon) + g(y_j - \epsilon) - g(y_i) - g(y_j) < 0. \end{aligned}$$

The last inequality is due to the the convexity of g.

Now, we show such coordinates i and j alway exist. Let $i = \arg\min_k \{y_k \mid y_k \neq x_k\}$. Suppose y_i does not participate in any tight constraint, i.e., $y(S) < f(S)$ for all $S \ni i$. Then, we can slightly increase y_i without violating any constraint and get a smaller $G(y)$ value. So y can not be a minimizer of G over $\mathcal{P}_{\text{rank}}$. Now, we consider the harder case where y_i participates in some tight constraints. Consider any two such constraints $y(S_1) = \text{rank}(S_1)$ and $y(S_2) = \text{rank}(S_2)$, where $S_1, S_2 \ni i$. Due to the submodularity of rank, we also have that $y(S_1 \cap S_2) = \text{rank}(S_1 \cap S_2)$ (and $y(S_1 \cup S_2) = \text{rank}(S_1 \cup S_2)$). In fact, this is because

$$\begin{aligned} y(S_1 \cap S_2) + y(S_1 \cup S_2) &\leq \text{rank}(S_1 \cap S_2) + \text{rank}(S_1 \cup S_2) \\ &\leq \text{rank}(S_1) + \text{rank}(S_2) = y(S_1) + y(S_2). \end{aligned}$$

Hence, the equality holds throughout in the chain. Let $T = \cap_k S_k$. Then, we can get that $y(T) = \text{rank}(T)$. Moreover, T is not empty since it contains i. If we can show that there exist $j \in T$ such that $y_j > y_i$, the proof is done since increasing y_i by ϵ and decreasing y_j by ϵ would maintain $y(S_k) = \text{rank}(S_k)$ for all k and does not violate other constraints for small enough $\epsilon > 0$.

To show the existence of such j, it is critical to observe that $x_i > y_i$. This is one property of our water-filling algorithm which can be seen as follows: Suppose $y_i > x_i$. During the process of water-filling, when coordinate i reaches value x_i, the value of each other coordinate i' is either less than or equal to $y_{i'}$ (because of the definition of i). This means coordinate i does not participate in any tight constraint when its value is x_i, rendering a contradiction. Since $y(T) = \text{rank}(T) \geq x(T)$ and $y_i < x_i$, there must be another coordinate j with $y_j > x_j$. Again by the definition of i, we have $y_j > y_i$.

The uniqueness is also clear from the above proof since any other vector in $\mathcal{P}_{\text{rank}}$ does not minimize G. This completes the proof of the theorem. ■

4. AN EFFICIENT ALGORITHM VIA PARAMETRIC FLOW

In this section, we show that, instead of using the ellipsoid algorithm, one can find the Lorenz-dominant utility profile using one maximum matching computation and one parametric flow computation, both of which admit polynomial time combinatorial algorithms that are very efficient in practice. We first show that we can construct a network flow instance, so that (roughly speaking) each feasible flow corresponds a feasible utility profile and vise versa. Then we conduct a parametric flow computation on this network to find the Lorenz-dominant utility profile.

We need the following Gallai-Edmonds Decomposition (GED) lemma [11] in the construction of the network flow instance. The GED lemma also plays a critical role in [23]. Suppose $\mathcal{U}_{\max} = \{\mu : \mu \text{ is a maximum cardinality matching}\}$. Firstly, we partition V into three parts V^E, V^M and V^M:

1. V^C: For each $v \in V^C$, there is some maximum matching where v is not matched, i.e., $V^C = \{i \in [n] : \exists \mu \in \mathcal{U}_{\max},\ s.t.\ \mu(ij) = 0\ \forall j \in [n]\}$.
2. V^M: It is the set of vertices that are not in V^C but have neighbor(s) in V^C, i.e., $V^M = \{i \in [n] \setminus V^C : \exists j \in V^C,\ s.t.\ (i,j) \in E\}$.
3. $V^E = V \setminus (V^M \cup V^C)$ is the set of the rest of vertices.

LEMMA 3. *(Gallai-Edmonds Decomposition) Let $\mu \in \mathcal{U}_{\max}$ be an arbitrary maximum cardinality matching.*

1. *For any vertex $i \in V^E$, there exists a vertex $j \in V^E$, such that $(i,j) \in \mu$. In fact, V^E consists of even connected components* [6]*.*
2. *For any vertex $i \in V^M$, there exists a vertex $j \in V^C$, such that $(i,j) \in \mu$;*
3. *V^C consists of several odd connected components. For any odd component $C \in V^C$, one of the below holds:*
   (a) *One and only one vertex $i \in C$ is matched with a vertex $j \in V^M$ while all remaining vertices in C are matched with each other in μ;*
   (b) *Arbitrary one vertex $i \in J$ remains unmatched in μ while all remaining vertices in C are matched with each other in μ.*

See Figure 1 for an example. It is well known that the Gallai-Edmonds decomposition can be computed efficiently.

LEMMA 4. *[10] Given a graph $G = (V,E)$ where $|V| = n$, there exists an algorithm that computes a maximum cardinality matching in $O(n^3)$ time. The Gallai-Edmonds decomposition can be computed from an maximum matching in an additional $O(n^2)$ time.*

We can use other algorithms to find the maximum cardinality matching, such as the Micali-Vazirani algorithm which runs in time $O(\sqrt{|V|}|E|)$ [17].

4.1 A Flow Network Construction based on GED

Given an undirected graph $\mathcal{G} = (V,E)$ where $|V| = n$, we first compute the Gallai-Edmonds decomposition which partitions the vertices into three disjoint sets V^E, V^M and V^C. Define $\mathcal{C} = \{C_1, \dots, C_k\}$ as the set of odd components in V^C. where $k = |\mathcal{C}|$. Then, we construct a bipartite graph $\mathcal{N} = (A, B, F)$. Here each node in A corresponds to a node in V. We use $a_i \in A$ to denote the node corresponding to $i \in V$. We use $A^{C_1}, ..., A^{C_k}$, to denote the k disjoint sets corresponding to $C_1, \dots, C_k$ respectively. Let $A^{EM} = \{a_i \mid i \in V^E \cup V^M\}$ $A^C = \{a_i \mid i \in V^C\}$. The construction of B is as follows. B can be partitioned into three parts $B^{EM} \cup B^M \cup B^C$. Each node in B^{EM} corresponds to a node in $V^E \cup V^M$. We use $b_i \in B^{EM}$ to denote the node corresponding to $i \in V^E \cup V^M$. Each node in B^M corresponds to a node V^M. We use $b'_i \in B^M$ to denote the node corresponding to $i \in V^M$. B^C consists of k disjoint sets $B^{C_1}, ..., B^{C_k}$, where B^{C_i} contains $|C_i| - 1$ vertices.

We now describe the construction of F.

[6]The number of vertices in a component is even.

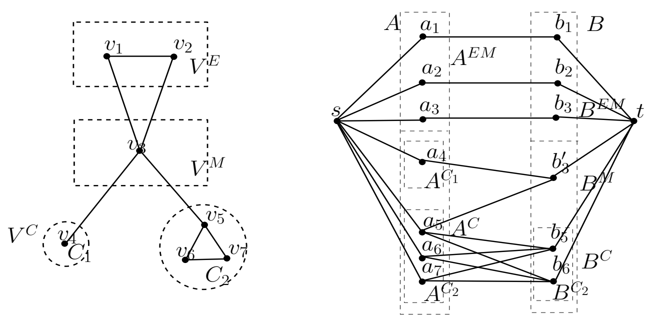

Figure 1: The left hand side illustrates the the Gallai-Edmonds decomposition of the graph and the right hand side is the corresponding flow network.

1. For each $i \in V^E \cup V^M$, we have edge $(a_i, b_i) \in F$ where $a_i \in A^{EM}$ and $b_i \in B^{EM}$.
2. For a vertex $a_i \in A^C$ and another vertex $b'_j \in B^M$, we have $(a_i, b_j) \in F$ if $(i, j) \in E$.
3. For each vertex $a \in A^{C_i}$ and each vertex $b \in B^{C_i}$ $(1 \leq i \leq k)$, we have $(a, b) \in F$. In other words, A^{C_i} and B^{C_i} form a complete bipartite graph.

Finally we add two distinguished vertices s and t into the graph $\mathcal{N}$. s is the source and is connected to all vertices of A. t is the sink and is connected to all vertices of B. All edges from A to B have infinity capacity and the rest of edges have unit capacity. This completes the the construction of the network. See Figure 1 for an illustration.

Definition 6. (flow profile) *Suppose f is a feasible flow in the network $\mathcal{N}$. We define the induced flow profile u^f to be a non-negative real value vector $u^f = (f(s, a_i))_{i \in [n]}$. We say a vector is a* feasible flow profile *if it is a flow profile induced by some feasible flow.*

The following theorem states the relation between the set of feasible flow profiles and the set of utility profiles, which plays an essential role in our algorithm. Due to space constraint, we omit the proof, which can be found in the full version of the paper.

Theorem 2. *The vector $u = \{u_1, \dots, u_n\}$ is a feasible flow profile for $\mathcal{N}$ iff u is a feasible utility profile for $\mathcal{G}$.*

It is convenient to view the above theorem in the form of the following corollary. For any $S \subseteq V$, we use $\mathcal{N}(S)$ to denote the network that is the same as $\mathcal{N}$ except that the set of edges going out of s is $\{(s, a_i) \mid i \in S\}$.

Corollary 2.1. *The polytope*

$$\{u \mid u \text{ is a feasible flow profile}\}$$

is the same as $\mathcal{P}_f$ (thus is a polymatroid). Moreover, the corresponding rank function $\text{rank}(S)$ *can be reinterpreted as the maximum amount of flow that can be pushed to the sink t in $\mathcal{N}(S)$.*

4.2 The Parametric Flow Algorithm

We briefly review the notion of parametric flow and show how to use a parametric flow computation to compute the Lorenz-dominant utility profile.

Parametric network : In the parametric network $\mathcal{N}_\lambda$, the edge capacities $c_\lambda(u, v)$ are linear functions of a real-valued parameter λ such that

1. $c_\lambda(s, a_i) = \lambda$ for all a_i. [7]
2. $c_\lambda(v, w)$ is constant for all $v \neq s$.

Let $\kappa(\lambda)$ be the value of maximum flow (also the minimum cut) on $\mathcal{N}_\lambda$. Consider the curve $\kappa(\lambda)$ on the λ-κ coordinate system. Imagine the process of gradually increasing λ from 0 to 1. Initially, all (s, a_i)s are saturated (i.e., $f(s, a_i) = c_\lambda(s, a_i)$) and the maximum flow is simply $|A|\lambda$ when λ is very small. This gives the first piece of the curve, $|A|\lambda$. As we continue increasing λ, at some point, not all (s, a_i)s can be saturated and the slope of κ decreases. In general, $\kappa(\lambda)$ is a piecewise-linear concave function with at most $n - 2$ breakpoints (see e.g., [12]).

In fact, all breakpoints and all maximum flows corresponding to these breakpoints can be found by a parametric flow computation in $O(nm \log(n^2/m))$ time [12]. Moreover, there is an important property of the algorithm in [12]:

P. As λ increases, $f(s, a_i)$ never decreases for all $a_i \in A$.

Simulating the Water-Filling Algorithm : With Property (P) and Corollary 2.1, the evolution of the flow profile u_f as we increase λ coincides with the evolution of the utility profile x as we increase λ during the execution of the water-filling algorithm. For example, in the beginning, we have all (s, a_i) are saturated in $\mathcal{N}_\lambda$ when λ is very small. This is the same as setting $x_i = \lambda \forall i \in V$ in the linear program LP_1. As we keep increasing λ, due to Corollary 2.1, we will hit a tight constraint $x(S) = \lambda|S| = f(S)$ where $f(S)$ is interpreted as in Corollary 2.1. So we reach the first break point and its value is exactly λ_1, the optimal value of LP_1. Due to Property (P), for all $i \in S$, $f(s, a_i)$ does not decrease or increase even we increase the capacity of (s, a_i) later. This is the same as fixing the value of all tight elements in D_1. The later steps can be argued in exactly the same way. Hence, at the end of this process ($\lambda = 1$), the flow profile returned by the parametric flow algorithm is exactly the same as the utility profile obtained by solving the set of linear programs, which is Lorenz-dominant by Theorem 1. We summarize our discussion in the following theorem.

Theorem 3. *There is an $O(n^3)$ algorithm for finding the Lorenz-dominant utility profile on an undirected graph with n vertices.*

Evolution of the Minimum s-t Cut : Let us gain a bit more insight of the process of increasing λ by considering the evolution of the minimum cut $(X_\lambda, \bar{X}_\lambda)$ where the source node $s \in X_\lambda$ and the sink node $t \in \bar{X}_\lambda$. In fact, the evolution of min-cut matches perfectly with the evolution of the tight elements: at step k, we can see that the set of node in A that join X_λ is exactly D_k, the set of new tight elements found by LP_k. More details can be found in the full version of the paper.

[7]In a general parametric network, the capacity of an edge going out s can be a general non-decreasing linear function of λ and the capacity of an edge entering t can be a general non-increasing linear function of λ.

5. RELATION WITH THE ROTH-SONMEZ-UNVER ALGORITHM

Roth, Sonmez and Unver [23] proposed an exponential time algorithm (denoted as the RSU algorithm from now on) to find the Lorenz-dominant utility profile. The running time of RSU is exponential in the number of odd components in the GED decomposition of $\mathcal{G}$. Due to the uniqueness of the Lorenz-dominant utility profile, our parametric flow algorithm obviously returns the same solution as the RSU algorithm. However, reaching this conclusion of course requires the correctness proof of RSU algorithm in [23], which is rather involved. In this subsection, we would like to make more explicit the relation between the RSU algorithm and our LP and parametric flow approaches. The purpose of this explicit relation is two fold: (1) The proof of Theorem 1 can then be seen as an alternative (but shorter) correctness proof of the RSU algorithm. (2) The parametric flow algorithm can be view as a polynomial time implementation of the RSU algorithm.

First, let us briefly review the RSU algorithm, which is also based on the GED decomposition. Let $\mathcal{C}$ be the set of odd components of V^C. For $\mathcal{J} \subseteq \mathcal{C}$ and $I \subseteq V^M$, define $C(\mathcal{J}, I) = \{i \in I : \exists j \in \mathcal{J}, \text{s.t. } (i,j) \in E\}$ which represents the neighbors of the set of odd components $\mathcal{J}$ in I. Define a real value function f through

$$f(\mathcal{J}, I) = \frac{|\bigcup_{J \in \mathcal{J}} J| - (|\mathcal{J}| - |C(\mathcal{J}, I)|)}{|\bigcup_{J \in \mathcal{J}} J|}$$

The RSU algorithm proceeds as follows:

1. Step 1: Let $D'_1 = \arg\min_{\mathcal{J} \subseteq \mathcal{C}} f(\mathcal{J}, V^M)$ and $V_1^M = C(D_1, V^M)$. If there are several sets minimizing f, we take their union.

2. In general, at step k, let $D'_k = \arg\min_{\mathcal{J} \subseteq \mathcal{C} \setminus \bigcup_{l=1}^{k-1} D'_l} f(\mathcal{J}, V^M \setminus \bigcup_{l=1}^{k-1} V_l^M)$ and $V_k^M = C(D'_k, V^M \setminus \sum_{l=1}^{k-1} V_l^M)$.

The utility profile u obtained is such that $u_i = 1$ for each $i \in V \setminus V^C$, and $u_i = f_k \triangleq f(D'_k, V_k^M)$ for each $i \in D'_k$.

Now, we establish the relation between the RSU algorithm and our water-filling algorithm. In particular, we can show the following lemma.

LEMMA 5. *We have that, for all k, $f_k = \lambda_k$ and $D'_k = D_k$ where λ_k is the kth breakpoint of the curve $\kappa(\lambda)$ (λ_k is also the optimal value of* LP_k*) and D_k is the set of tight elements found in step k.*

6. EXPERIMENTS

In this section, we describe our implementations of the water-filling algorithm using parametric flow and report our experimental results on several generated data sets. We compare our algorithm with (an improved version of) the RSU algorithm. Since both the RSU and our algorithm are exact algorithms, for all data sets, both return the same results. As expected, our algorithm runs significantly faster.

6.1 Implementation Details

We first make substantial improvements to both of the original RSU and our parametric flow algorithm. We then implement the improved algorithms for comparison.

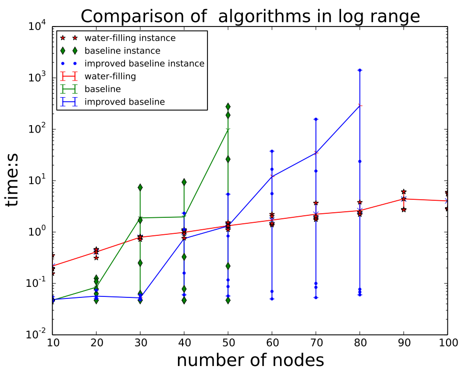

Figure 2: Comparison of mechanisms

6.1.1 Improve RSU

Recall that in GED decomposition, M is the set of vertices that are always in any maximum matching and are connected by those who are not, and C is the set of vertices that may not be in a maximum matching. For a typical graph, $|M|$ is much smaller than $|C|$, were $|C|$ is the number of odd components in C. $|M|$ is the number of the vertices in M. The original RSU algorithm exhaustively enumerates of all components in C. In our improved version, we show that one only needs to check all subsets of M. As a result, the time complexity of RSU is improved from $O(2^{|C|}|M|)$ to $O(2^{|M|}|C| \log |C|)$. We refer the reader to the full version of the paper for technical details of this part.

6.1.2 Simplify the flow network

In fact, the flow network constructed in Section 4 can further be simplified, in an equivalent way, as follows.

First, remove parts A^{EM} and B^{EM} and set final utility of each vertex in both sets to be 1. Second, for components that are not connected to M, calculate the utility of each vertex directly as follows: if the size of a component is s, then the utility of any vertex in this component is $\frac{s-1}{s}$. Third, replace nodes of the same component by a single node, and add up all the capacities and flows between this component and outside nodes accordingly. One can show the simplified network is equivalent to the one constructed in Section 4.

6.2 Experiment results

As mentioned, we implement the improved algorithms and compare them over several data sets. As is standard from the literature, we used a data generator by Saidman et al.[24]. Our code is written using both C++ and Python.

The result shows that if the data set is very small, the RSU is slightly faster than our water-filling algorithm. However when the data set is slightly larger, water-filling consistently beats the RSU with a large margin. The reason that RSU slightly outperforms water-filling in small data is the overhead introduced by the construction of the network required by the parametric flow implementation. However, this small overhead is outweighed by computational complexity once the graph gets large. In addition, we find the time complexity of the RSU is closely related to $|M|$, which may differ for two graphs with the same size but different structures.

Fig.2 shows the comparison between water-filling, the RSU algorithm and the improved RSU algorithms. We test 5 in-

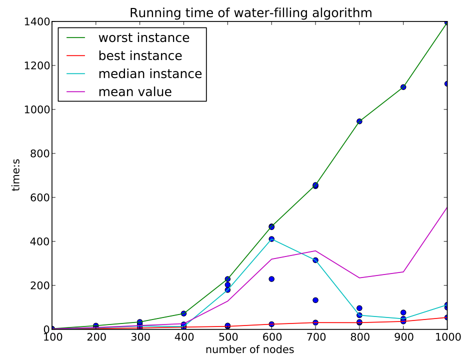

Figure 3: This figure shows how the running time grows as a function of the number of nodes

tances for each data size (20, 40, 60, 80, 100 nodes) and each algorithm. Those cases with more than 10 minutes for each in average are not plotted on the figure. As shown in the figure, our water-filling algorithm is also much stable than the other two algorithms. From the figure, it can be seen that our algorithm fluctuates between the orders of n^2 and n^3 as n grows, while the RSU algorithms are much slower and much more unstable.

Fig.3 shows the runtime of the water-filling algorithm on larger scale. The running time of water-filling now fluctuates drastically for different instances. Recall that the worst case complexity of our algorithm is $O(n^3)$. Clearly, the bound is not met in many instances. When the size of graph gets large, bad cases and good cases vary drastically in graph structure. In other words, the fluctuation in performance is due to the intrinsic complexity of the set of instances.

7. FUTURE WORK

One immediate next step is to extend our algorithms to a more general setting that accommodates chains and cycles. Note that the Lorenz-dominant allocation may not exist in more general settings. Another important extension is to consider the dynamic exchange network where nodes arrive and leave online. How to deal with issues such fairness, incentive and computation efficiency in the dynamic setting is an interesting but challenging future direction.

8. REFERENCES

[1] D. J. Abraham, A. Blum, and T. Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In *ACM EC*, pages 295–304, 2007.

[2] I. Ashlagi, F. A. Fischer, I. A. Kash, and A. D. Procaccia. Mix and match. In *ACM EC*, pages 305–314, 2010.

[3] I. Ashlagi, D. Gamarnik, M. A. Rees, and A. E. Roth. The need for (long) chains in kidney exchange. Working Paper 18202, National Bureau of Economic Research, July 2012.

[4] I. Ashlagi and A. Roth. Individual rationality and participation in large scale, multi-hospital kidney exchange. In *ACM EC*, pages 321–322.

[5] A. Bogomolnaia and H. Moulin. A new solution to the random assignment problem. *Journal of Economic Theory*, 100(2):295–328, 2001.

[6] W. Cunningham and A. Marsh. A primal algorithm for optimum matching. *Polyhedral Combinatorics*, pages 50–72, 1978.

[7] J. P. Dickerson, A. D. Procaccia, and T. Sandholm. Dynamic matching via weighted myopia with application to kidney exchange. In *AAAI*, 2012.

[8] J. P. Dickerson, A. D. Procaccia, and T. Sandholm. Optimizing kidney exchange with transplant chains: theory and reality. In *AAMAS*, pages 711–718, 2012.

[9] J. P. Dickerson, A. D. Procaccia, and T. Sandholm. Price of fairness in kidney exchange. In *AAMAS, to appear*, 2014.

[10] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[11] T. Gallai. Maximale systeme unabhängiger kanten. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 9:401–413, 1964.

[12] G. Gallo, M. Grigoriadis, and R. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

[13] S. Gentry, D. Segev, and R. Montgomery. A comparison of populations served by kidney paired donation and list paired donation. *American Journal of Transplant*, 5(8):1914–21, 2005.

[14] S. Iwata and J. Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1230–1237. Society for Industrial and Applied Mathematics, 2009.

[15] R. Latham. Lorenz-dominating income tax functions. *International Economic Review*, 29(1):185–198, 1988.

[16] A. Marshall, I. Olkin, and B. Arnold. *Inequalities: theory of majorization and its applications*. Springer, 2010.

[17] S. Micali and V. Vazirani. An o($\sqrt{|V|}|e|$) algoithm for finding maximum matching in general graphs. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 17–27. IEEE, 1980.

[18] M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 248–255. IEEE, 2004.

[19] J. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.

[20] A. Roth, T. Sonmez, and M. Unver. Kidney exchange. *Quarterly Journal of Economics*, 119:457–488, 2004.

[21] A. Roth, T. Sonmez, and U. Unver. A kidney exchange clearinghouse in new england. *American Economic Review*, pages 376–380, 2005.

[22] A. Roth, T. Sonmez, and U. Unver. Efficient kidney exchange: Coincidence of wants in a markets with compatibility-based preferences. *American Economic Review*, 2007.

[23] A. Roth, T. Sönmez, and M. Utku Ünver. Pairwise kidney exchange. *Journal of Economic Theory*, 125(2):151–188, 2005.

[24] S. L. Saidman, A. E. Roth, T. Sönmez, M. U. Ünver, and F. L. Delmonico. Increasing the opportunity of live kidney donation by matching for two-and three-way exchanges. *Transplantation*, 81(5):773–782, 2006.

[25] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[26] D. L. Segev, S. E. Gentry, D. S. Warren, B. Reeb, and R. A. Montgomery. Kidney paired donation and optimizing the use of live donor organs. *Journal of American Medical Association*, 2005.

[27] C. Sung. Achieving log-utility fairness in cdma systems via majorization theory. *Communications Letters, IEEE*, 13(9):625–627, 2009.

[28] A. Tamir. Least majorized elements and generalized polymatroids. *Mathematics of Operations Research*, 20(3):583–589, 1995.

[29] P. Toulis and D. C. Parkes. A random graph model of kidney exchanges: efficiency, individual-rationality and incentives. In *ACM EC*, pages 323–332, 2011.

[30] O. Yilmaz. Kidney exchange: An egalitarian mechanism. *Journal of Economic Theory*, 146(2):592–618, 2011.