Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Exact Algorithms and Complexity of Kidney Exchange

Mingyu Xiao, Xuanbei Wang
School of Computer Science and Engineering,
University of Electronic Science and Technology of China, China
myxiao@gmail.com, wxb_uesct@163.com

Abstract

Kidney Exchange is an approach to donor kidney transplantation where patients with incompatible donors swap kidneys to receive a compatible kidney. Since it was first put forward in 1986, increasing amount of people have gotten a life-saving kidney with the popularity of Kidney Exchange, as patients have more opportunities to get saved in this way. This growth is making the problem of optimally matching patients to donors more difficult to solve. The central problem, indeed, is the NP-hard problem to find the largest vertex-disjoint packing of cycles and chains in a graph that represents the compatibility between patients and donors, where due to the human resource limitation we may have constraints on the maximum length of cycles and chains. This paper mainly contributes to algorithms from theory for this problem with and without length constraints (restricted and free versions). We give: 1. A single-exponential exact algorithm based on subset convolution for the two versions; 2. An FPT algorithm for the free version with parameter being the number of vertex "types" in the graph.

1 Introduction

There are two treatment options for kidney disease. One is dialysis, the other is *kidney transplantation*. The latter is more preferred than the former. A patient may be saved by getting a kidney from a (deceased) donor. Unfortunately, the demand for kidneys far outstrips supply. In general, the average waiting time ranges from 2 to 5 years at most centers and even longer in some geographical regions [Abraham *et al.*, 2007].

Kidney Paired Donation (*KPD*), also called *Kidney Exchange*, is put forward to complement the weakness of dead body organ donation via the deceased donor waiting list. KPD is an approach to living donor kidney transplantation where patients with incompatible donors swap kidneys to receive a compatible kidney. Since it was first introduced in [Rapaport, 1986], increasing amount of people have gotten a life-saving kidney with the popularity of KPD, as patients have more opportunities to get saved in this way. Because better blood type and age matching are correlated with lower lifetime mortality and longer lasting kidney transplants [Segev *et al.*, 2005], many compatible pairs are also participating in swaps to find better matched kidneys. How to swap to save more patients or to get more benefits? This arises the *kidney exchange* problem.

In kidney exchange, patients and donors participate in *cycles* and *chains*. A patient together with his/her donor is regarded as a *vertex*. In a cycle, a patient in a vertex receives the compatible kidney of the donor in the previous vertex. The number of vertices in a cycle is the *length* of the cycle. A donor, as soon as the partner patient has received a kidney, can technically exit the program without donating one, as he/she is not legally bound to do so [Roth *et al.*, 2005b; Segev *et al.*, 2005; Mak-Hau, 2017]. In order to avoid this, usually exchanges are carried out simultaneously. This ensures that no donor backs out after his/her paired patient has received a kidney but before he/she donated one of his/her own kidneys [Dickerson *et al.*, 2017]. As each transplant involves two surgeries, there is a limit as to how many exchanges can be performed at once, due to human resource and logistic reasons. Thus, most field kidney exchanges, including UNOS, allow only cycles with small size [Gruessner and Sutherland, 2005].

For chain models, a donor without a paired patient, called the *altruistic donor*, enters the pool. An altruistic donor is regarded as an *altruistic vertex*. A sequence of kidney exchanges that begins from an altruistic vertex and terminates at a normal vertex forms a *chain*. The altruistic donor donates his/her kidney to a patient whose paired donor donates his/her kidney to another patient and so on. The *length* of a chain is defined to be the number of non-altruistic vertices in the chain, that is, the number of participant patients in it. In real life, chains can be executed non-simultaneously [Anderson *et al.*, 2015]. To see why, consider the situation where a donor backs out of a chain after his/her paired patient received a kidney, but before his/her own donation. Unlike in the case of a broken cycle, no pair in the remaining tail of the planned chain is strictly worse off; that is, no donor was "used up" before his/her paired patient received a kidney. Thus, chains can be much longer than cycles.

We can form kidney exchange problems as graph optimization problems. A pair of patient and donor is regarded as a vertex, an altruistic donor is also regarded as a vertex, and

there is a directed arc from a vertex i to a vertex j if the donor in vertex i is compatible to the patient in vertex j. Thus, we get a directed graph called the *compatibility graph*. Chains in compatibility graph are paths beginning at an altruistic vertex. We want to save patients as more as possible. So it becomes the optimization problem of finding a vertex-disjoint packing of cycles and chains in the compatibility graph to cover maximum number of vertices. This problem is also known as the *clearing problem*. Note that in compatibility graph, self-loops are allowed, i.e., an arc may begin and end at the same vertex, since some compatible pairs of patient and donor may also want to participate in swaps to find better matched kidneys.

With different constraints, we can get different versions of the problem. Most constraints are about the length of cycles and chains in kidney exchanges. In many previous problem models, the number of participants in a cycle is usually bounded by 3 due to the limitation on human resources in real life [Roth *et al.*, 2005b; Abraham *et al.*, 2007; Biro *et al.*, 2009; Mak-Hau, 2017]. In recent years, with the progress of technology, more and more patients are allowed to participate in cycles and chains. Manlove and O'Malley [2012] demonstrated the benefit of allowing 4-cycle exchanges, when compared with just 2/3-cycle exchanges. A 9-cycle kidney exchange was performed successfully in 2015 at two San Francisco hospitals [Mak-Hau, 2017]. The bounds on the length of cycles and chains may be further increased in the future. We study a general problem with general bounds L_c and L_p on the length of cycles and chains.

(L_c, L_p)-Kidney Exchange ((L_c, L_p)-KEP)
Input: A directed graph $G = (V, A)$ with possible self-loops, an altruistic vertex set $B \subseteq V$, and two nonnegative integers L_c and L_p.
Output: A set of vertex-disjoint directed cycles and paths covering maximum number of non-altruistic vertices in the graph, where each cycle has length at most L_c and each path begins with an altruistic vertex and has length at most L_p.

In (L_c, L_p)-KEP, we can let L_c and L_p to be 0 to indicate no cycles/chains allowed in the exchange. We also independently consider the version without constraints on the lengths of cycles and chains. In this problem, we want to find out how many patients can be saved "theoretically". We say "theoretically" because we allow the length of cycles and chains at any large.

Free Version of Kidney Exchange (F-KEP)
Input: A directed graph $G = (V, A)$ with possible self-loops and an altruistic vertex set $B \subseteq V$.
Output: A set of vertex-disjoint directed cycles and paths covering maximum number of non-altruistic vertices in the graph, where each path begins with an altruistic vertex.

1.1 Related Work

Since the first introduction of the kidney exchange problem [Rapaport, 1986], many properties and variants have been developed in the following research [Roth *et al.*, 2005b; 2005a; Segev *et al.*, 2005]. Some versions only consider cycles when finding the packing [Constantino *et al.*, 2013; Klimentova *et al.*, 2014; Sönmez and Ünver, 2014], while others consider cycles and chains together [Manlove and O'Malley, 2012; Glorie *et al.*, 2014]. Different restrictions on the length of cycles and chains have also been studied. For example, the problem in [Manlove and O'Malley, 2012] has the same size limitation for cycles and chains. Some may consider looser restrictions on chains [Glorie *et al.*, 2014; Anderson *et al.*, 2015]. Different versions can be regarded as different packing problems in graphs. Some relations between barter exchange and set packing are discussed in [Jia *et al.*, 2017].

Piratical (heuristic) algorithms for real-life data are hot tops and have also been extensively studied. Most of the fast algorithms are based on integer programming [Manlove and O'Malley, 2014; Dickerson *et al.*, 2016; Glorie *et al.*, 2014; Li *et al.*, 2014]. Some other methods can also be found in the literature [Biro *et al.*, 2009; Klimentova *et al.*, 2014; Dickerson *et al.*, 2017]. Some approximation algorithms for different versions are developed by showing the relations to the set packing problem [Krivelevich *et al.*, 2007; Jia *et al.*, 2017]. On the other hand, the $\mathcal{NP}$-hardness of many variants of the kidney exchange problem has been established [Krivelevich *et al.*, 2007].

To make the problems tractable, Dickerson et al. [2017] introduced a way to represent the kidney exchange graph in a compression way by identifying vertices having the same neighborhood properties. In the compatibility graph, vertices are in the same "type" if they have the same in- and out-neighborhood. The number of different vertex types may not be very large since they are determined by the blood type, age and some other properties of the patients and donors. Sometimes, we can view it as a constant. Dickerson et al. [2017] show that the kidney exchange problem can be solved in polynomial time when both the length of cycles and chains and the number of different types are bounded by some constants. The concept of "vertex type" is interesting and we can also consider it in this paper.

1.2 Main Contributions

In this paper, we mainly consider exact algorithms for the two fundamental computational problems in kidney exchange: (L_c, L_p)-KEP and F-KEP. In parameterized complexity, we study *parameterized problems*. An instance of a parameterized problem consists of an instance I of the original (NP-hard) problem and a parameter k. A parameterized problem with parameter k is *fixed-parameter tractable* (FPT) if and only if it allows an algorithm with running time $f(k)poly(|I|)$, where $f(k)$ is a computable function on k only, and $poly(|I|)$ is a polynomial function on the input size. These kinds of algorithms are called FPT algorithms. Under some reasonable assumptions, some parameterized problems do not allow FPT algorithms, which are *W[1]-hard* [Downey and Fellows, 2013]. We will consider the kidney exchange problems with the parameter being the number of vertex types. The main contribution of this paper contains two algorithmic results:

1. We show that both of (L_c, L_p)-KEP and F-KEP can

be solved in $O(2^n n^3)$ time. Our algorithms use the techniques of dynamic programming and subset convolution. Note that the trivial search algorithm uses at least $n^n n^{O(1)}$ time and most known IP algorithms for kidney exchange problems in the literature have much worse running time bound. Our algorithms use only single exponential running time.

2. We show that F-KEP is FPT by taking the number of vertex types as the parameter, which also implies that F-KEP is polynomially solvable when the numbers of different types of patients and donors are bounded by a constant. The algorithm is based on a decomposition technique and an IP model.

To solve the kidney exchange problem, we mainly work on the compatibility graph solving a cycle/chain packing problem. The compatibility graph $G = (V, A)$ is a directed graph with possible self-loops. We will use $n = |V|$ and $m = |A|$ to denote the numbers of vertices and arcs in the compatibility graph. A set of vertices in G is called an *independent set* if there is no arc from any vertex to another in the set. A *directed complete graph* is defined as a graph where each pair of vertices have two arcs from each to other and each vertex also has a self-loop. A single vertex without a self-loop is an independent set and a single vertex with a self-loop is a directed complete graph.

2 Exact Algorithms Based on Subset Convolution

In this section, we design exact algorithms for both (L_c, L_p)-KEP and F-KEP based on dynamic programming and subset convolution. Most previous exact algorithms for variants of the kidney exchange problem are based on IP and the running time of them are not analyzed or take a trivial bound of $n^n n^{O(1)}$ at least. We show that (L_c, L_p)-KEP and F-KEP can be solved in $2^n n^{O(1)}$ time by a clever using of subset convolution. In fact, the framework of our algorithms for (L_c, L_p)-KEP and F-KEP are the same, which can even be directly used to design algorithms for more variants of the kidney exchange problem with the same running time bound.

Definition 1. (Packing Unit) *Given a graph $G = (V, A)$ with an altruistic vertex set $B \subseteq V$,*

1. *A directed cycle or a directed path starting from an altruistic vertex in G is called a* packing unit *of F-KEP.*
2. *A directed cycle of length at most L_c or a directed path starting from an altruistic vertex of length at most L_p is called a* packing unit *of (L_c, L_p)-KEP.*

Sometimes we may simply use the vertex set of a packing unit to denote it. Packing unit is a basic notation used in our algorithms. The definitions of packing units are different for the two problems (L_c, L_p)-KEP and F-KEP. This will be the only difference in our algorithms for the two problems. So we may simply say "packing unit" without specifying for which problems when it is suitable for both cases.

For a packing unit of a cycle, we call it a *cycle-packing unit*. For a packing unit of a path, we call it a *path-packing unit*. Note that if a packing unit S is a cycle-packing unit then S contains no altruistic vertex, and if a packing unit S is a path-packing unit then S contains exactly one altruistic vertex. So a vertex subset S can not be both a cycle-packing unit and a path-packing unit.

Our algorithm contains two main steps. In the first step, we use a dynamic programming technique to check each of the 2^n vertex subsets whether it is a packing unit or not. The information will be stored in a table $D(\cdot)$ of size 2^n. These can be done in $O(2^n n^3)$ time. The algorithm follows the classic dynamic programming algorithm for checking if a graph being a Hamilton graph or not. However, our algorithm needs to check all induced subgraphs of the input graph not only the whole input graph. In the second step, based on $D(\cdot)$, we find out all vertex subsets that can be partitioned into packing units and the one containing maximum number of non-altruistic vertices will be an optimal solution to our problem. We will show that the second step can also be done in $O(2^n n^3)$ time by using subset convolution. Thus, our problems can be solved in $O(2^n n^3)$ time in total.

Step one. The purpose of this step is to find out which vertex subsets are packing units. The information will be store in a table $D(\cdot)$ of size 2^n.

For each vertex subset S and two vertices $v_1, v_2 \in S$, we consider if there is a path in $G[S]$ which starts in v_1, visits all vertices in $S \setminus \{v_1, v_2\}$, and ends in v_2. We use a table $H(\cdot)$ to store the information. If it is a yes-case for the triple (S, v_1, v_2), then $H(S, v_1, v_2) = 1$; otherwise $H(S, v_1, v_2) = 0$. The size of H is bounded by $O(2^n n^2)$. We use a dynamic programming method to compute the value of $H(S, v_1, v_2)$ for each $S \subseteq V$ and each vertex pair $v_1, v_2 \in S$.

For vertex subsets S with size $|S| \leq 2$, it is trivial to compute the value $H(S, v_1, v_2)$. Next, we assume that S is a set containing at least three vertices. For a triple (S, v_1, v_2), let S^* be the set of vertices $u \in S \setminus \{v_1, v_2\}$ such that there is an arc from u to v_2. There is a path P in $G[S]$ which starts in v_1, visits all vertices in $S \setminus \{v_1, v_2\}$, and ends in v_2 if and only if there are a vertex $u \in S^*$ and a path P' in $G[S]$ which starts in v_1, visits all vertices in $S \setminus \{v_1, v_2, u\}$, and ends in u. So we get the following relation

$$H(S, v_1, v_2) = \bigvee_{u \in S^*} H(S \setminus \{v_2\}, v_1, u). \quad (1)$$

We compute $H(S, v_1, v_2)$ in order of increasing cardinality of S by using (1). For each triple (S, v_1, v_2), the value $H(S, v_1, v_2)$ can be computed by using at most $|S| - 2$ basic computations based on the values $H(S', v_1', v_2')$ for S' with cardinality $|S'| = |S| - 1$. The computation for each element in $H(\cdot)$ takes $O(n)$ basic steps. The size of the table $H(\cdot)$ is bounded by $O(2^n n^2)$. Thus, we can compute all the values in $H(\cdot)$ in $O(2^n n^3)$ time.

Next, we are ready to check if a vertex set S is a packing unit or not based on $H(\cdot)$. We first consider packing units for F-KEP. For (L_c, L_p)-KEP, we only need to add one more constraint on the length of cycles or paths.

We maintain another table $D(\cdot)$ of size 2^n. For each vertex subset $S \subseteq V$, if S is a packing unit then $D(S) = 1$; otherwise $D(S) = 0$. The special case $S = \emptyset$ will also be used and we let $D(\emptyset) = 1$. We can compute the values of

$D(S)$ for all $S \subseteq V$ in $O(2^n n)$ time based on table $H(\cdot)$. It is not hard to see follows.

When S contains no altruistic vertex in B, S can only be a cycle-packing unit if it is a packing unit. So, if $S \cap B = \emptyset$, we compute $D(S)$ by the following way: arbitrarily pick a vertex $v \in S$, let S' be set of vertices $u \in S$ having an arc from u to v, and let

$$D(S) = \bigvee_{u \in S'} H(S, v, u). \tag{2}$$

When S contains exactly one altruistic vertex v, S can only be a path-packing unit if it is a packing unit. So, If $|S \cap B| = 1$, we compute $D(S)$ by the following relation

$$D(S) = \bigvee_{u \in S \setminus \{v\}} H(S, v, u). \tag{3}$$

If $|S \cap B| \geq 2$, we simply let $D(S) = 0$ since it can not be a packing unit.

For (L_c, L_p)-KEP, we only have one more operation to check the size of S. Before executing (2) (resp., (3)), we check weather it holds $|S| \leq L_c$ (resp., $|S| \leq L_p$) or not. If not, we directly let $D(S) = 0$ without executing (2) or (3).

For each element in $D(\cdot)$, we use at most $|S| < n$ basic computations to calculate it. To compute all the values in $D(\cdot)$, we will use $O(2^n n)$ time.

Step two. Equipped with $D(\cdot)$, we will use subset convolution to find out which vertex subsets can be partitioned into disjoint packing units.

Definition 2. (Subset Convolution) *Let f be an integer function on subsets of a set V. The* subset convolution*, or for short the* convolution *of f, denoted by $f * f$, is a function assigning to any $S \subseteq V$ an integer as follows*

$$(f * f)(S) = \sum_{T \subseteq S} f(T) \cdot f(S \setminus T). \tag{4}$$

Theorem 1. [Björklund *et al.*, 2007] *Let V be a set of* size *n and f: $2^V \to \{0, 1\}$ be a function on the subsets of V. If the values of $f(S)$, for all $S \subseteq V$ are given, then the subset convolution $f * f$ can be computed in $O(2^n n^2)$ time.*

Subset convolution is a natural tool to solve our problems. An *indicator function* of a function f, denoted by $\widehat{f}$, is defined as: $\widehat{f}(S) = 0$ if $f(S) = 0$ and $\widehat{f}(S) = 1$ if $f(S) \geq 1$. We list the main steps of our algorithm as follows.

1. $f(S) \leftarrow D(S)$ for all $S \subseteq V$;
2. For i from 1 to $\lceil \log n \rceil$, do
   $g(S) \leftarrow (f * f)(S)$ for all $S \subseteq V$, and
   $f(S) \leftarrow f(S) \vee \widehat{g}(S)$ for all $S \subseteq V$.
3. The subset $S^* \subseteq V$ with $f(S^*) = 1$ containing the maximum number of non-altruistic vertices is the solution.

Lemma 1. *For any nonempty vertex subset $S \subseteq V$, if S can be partitioned into at most k disjoint nonempty packing units, then $f(S) = 1$ after $\lceil \log k \rceil$ iterations of Step 2 in the above algorithm.*

Proof. We prove the lemma by induction on the number $\lceil \log k \rceil$ of iterations of Step 2. Initially $f(S') = M(S')$ for all $S' \subseteq V$. Thus, the lemma holds for the first iteration. Assume that the lemma holds after $\lceil \log k \rceil - 1$ iterations. We prove that the lemma also holds after $\lceil \log k \rceil$ iterations.

Since S can be partitioned into at most k disjoint nonempty packing units, we know that there are two disjoint sets S_1 and S_2 such that $S = S_1 \cup S_2$ and S_1 (resp., S_2) can be partitioned into at most $\lfloor \frac{k}{2} \rfloor$ (resp., $\lceil \frac{k}{2} \rceil$) disjoint nonempty packing units. By the assumption, we know that $f(S_1) = 1$ and $f(S_2) = 1$ after $\lceil \log(\lceil \frac{k}{2} \rceil) \rceil \leq \lceil \log k \rceil - 1$ iterations. In the next iteration of Step 2, we know that $g(S) = (f * f)(S) > 0$ by (4). Thus, $\widehat{g}(S) = 1$ and then $f(S) = 1$. It holds for $f(S) = 1$ after $\lceil \log k \rceil$ iterations of Step 2. □

Lemma 1 guarantees the correctness of our algorithm. Note that any vertex subset can be partitioned into at most n disjoint nonempty packing units. By Lemma 1, we know that after $\lceil \log n \rceil$ iterations of Step 2, the subset $S^* \subseteq V$ with $f(S^*) = 1$ containing the maximum number of non-altruistic vertices is our solution. For the running time, we mainly compute $\lceil \log n \rceil$ times of subset convolutions. The running time is $O(2^n n^2 \lceil \log n \rceil)$.

3 Algorithms for Quotient Models

Dickerson et. al. [2017] introduced a way to class the vertices in a compatibility graph into different *types* according to some actual meaning and then get a contracted representation of the compatibility graph. Two vertices belong to the same type if and only if they have exactly the same in- and out-neighbourhood. In graph theory, a similar notation is known as *module*. A vertex subset $M \subseteq V$ forms a module of a directed graph if the vertices in M cannot be distinguished by any vertex out of M, i.e., for any vertex $x \in V \setminus M$, if there is an arc from x to a vertex in M (resp., an arc from a vertex in M to x), then there is an arc from x to each vertex in M (resp., an arc from each vertex in M to x). However, the concepts of "type" and "module" are not exactly the same. The concept of "type" has more requirements: any two vertices in the same type M' cannot be distinguished by any other vertex, not only vertices out of M'. Therefore, for a type M', the induced subgraph M' can only be a complete directed graph or an independent set. However, for a module M, the induced subgraph M can be of any form. Even the whole vertex set of a graph can be a (trivial) module. For the sake of the presentation, we will call types *pure modules*.

According to "if and only if" in the definition of type (or pure module), we know that any pure module is maximal (no proper subset is still a pure module) and no two pure modules are intersected. Therefore, the vertex set of a graph can be partitioned into several disjoint pure modules, which is called the *pure modular decomposition* of a graph. It is known that the pure modular decomposition of a directed graph is unique and it can be found in linear time [McConnell and de Montgolfier, 2005].

Definition 3. (Quotient Graphs) *The* quotient graph $Q = (V_Q, A_Q, w)$ *of a directed graph G based on a pure modular decomposition is a directed graph with vertex weight w :*

$V \to$ Z, where each vertex $v \in V_Q$ represents a pure module in G with the weight $w(v)$ being the number of vertices in the pure module and there is an arc from vertex a to vertex b in A_Q if and only if there is an arc from a vertex in the pure module corresponding to a to a vertex in the pure module corresponding to b in G.

Note that the quotient graph may also have self-loops and a pure module of a complete directed clique will have self-loops. Dickerson et. al. [2017] introduced an idea of using quotient graphs to represent big kidney exchange graphs. Furthermore, they showed that (L, L)-KEP can be solved in polynomial time when both L and the number of different pure modules are bounded by a constant by giving an $O((n+1)^{L \cdot \theta^L})$-time algorithm, where θ is the number of different pure modules in the pure modular decomposition of the compatibility graph. However, this algorithm is not FPT with parameter θ since the exponential part of the running time is still related to the input size n. Next, we will give an FPT algorithm for F-KEP with parameter θ.

Our algorithm mainly works on quotient graphs. We first apply the linear-time algorithm [McConnell and de Montgolfier, 2005] to compute the quotient graph $Q = (V_Q = V_I \cup V_C \cup V_B, A_Q, w)$ of the compatibility graph G, where we distinguish three kinds of vertices in V_Q: V_I is the set of vertices corresponding to the pure modules being independent sets, V_C is the set of vertices corresponding to the pure modules being complete directed cliques, and V_B is the set of vertices corresponding to altruistic vertices in G.

Given the quotient graph, we first build an integer programming (IP) model to find how many vertices in different pure modules can be included into the solution (the cycle/path packing). Then based on the IP solution, we construct a solution by using a decomposition technique.

3.1 Properties and IP Model

Before giving our IP, we introduce an important property used in the IP. We denote the vertices in V_I as $\{i\}_1^p$, the vertices in V_C as $\{i'\}_1^q$ and the vertices in V_B as $\{i^*\}_1^b$, where $p + q + b = n$. Vertex weights $w(i), w(i')$ and $w(i^*)$ may be simply written as $w_i, w_{i'}$ and w_{i^*}, respectively.

Definition 4. (Configuration) *A* configuration *of a quotient graph $Q = (V_Q = V_I \cup V_C \cup V_B, A_Q, w)$ of a compatibility graph G is an assignment of nonnegative integers to each vertices and arcs in Q, denoted by $\mathcal{A} = (\{x_i\}, \{x_{i'}\}, \{x_{i^*}\}, \{e_{st}\})$. A configuration is* feasible *if there is a cycle/path packing in G that contains exactly x_i (resp., $x_{i'}$ and x_{i^*}) vertices in the pure module i (resp., i' and i^*), and exactly e_{st} arcs from vertices in module s to vertices in module t (include self-loops).*

A cycle/path packing can uniquely determines a configuration: the value of each element in the configuration is the number of vertices or edges of this type in the packing. We say that this configuration is *corresponding* to this cycle/path packing. Note that some configuration may not be corresponding to any cycle/path packing.

Lemma 2. *A configuration $\mathcal{A} = (\{x_i\}, \{x_{i'}\}, \{x_{i^*}\}, \{e_{st}\})$ is feasible, if and only if it satisfies the following Constraints C1-C6:*

C1: $w_i \geq x_i \geq 0$, *for each i;*
$w_{i'} \geq x_{i'} \geq 0$, *for each i';*
$w_{i^*} \geq x_{i^*} \geq 0$, *for each i^*;*

C2: $x_j = \sum_i e_{ij} + \sum_{i'} e_{i'j} + \sum_{i^*} e_{i^*j}$ *for each j;*

C3: $x_j \geq \sum_i e_{ji} + \sum_{i'} e_{ji'}$ *for each j;*

C4: $x_{j'} = \sum_i e_{ij'} + \sum_{i'} e_{i'j'} + \sum_{i^*} e_{i^*j'}$ *for each j';*

C5: $x_{j'} \geq \sum_i e_{j'i} + \sum_{i'} e_{j'i'}$ *for each j';*

C6: $x_{i^*} = \sum_j e_{i^*j} + \sum_{j'} e_{i^*j'}$ *for each i^*.*

Constraint C1 says that the value for each vertex can not excess the weight of the vertex; Constraints C2 and C3 (resp., Constraints C4 and C5) say that for each vertex in V_I (resp., V_C), its value is equal to the sum of the values of all its in-arcs, and its value is an upper bound of the sum of the values of all its out-arcs; Constraint C6 says that for each vertex in V_B, its value is equal to the sum of the values of all its out-arcs. There are some different between Constraints C2-C3 and Constraints C4-C5, because vertices in pure modules of complete directed cliques have self-loops and can even form cycle containing only one vertex.

For the correctness of Lemma 2, we can easily observe that for any cycle/path packing, the configuration corresponding to it satisfies Constraints C1-C6. For the 'only if' part, we give an algorithm later to construct a cycle/path packing based on a configuration satisfying C1-C6.

Lemma 3. *Let $\mathcal{A}_1 = (\{x_i\}, \{x_{i'}\}, \{x_{i^*}\}, \{e_{st}\})$ be a feasible configuration. The configuration $\mathcal{A}_2 = (\{x_i\}, \{w_{i'}\}, \{x_{i^*}\}, \{e'_{st}\})$ is a still feasible configuration, where $e'_{st} = e_{st}$ if $s \neq t$ and $e'_{st} = w_{i'} - x_{i'}$ if $s = t = i'$.*

Proof. Let P_1 be the cycle/path packing corresponding to $\mathcal{A}_1$. For each i', we can add $w_{i'} - x_{i'}$ self-loops for vertices in the pure module i' into P_1 to form another cycle/path packing P_2. Then P_2 is a packing corresponding to $\mathcal{A}_2$. □

Lemma 4. *Let $\mathcal{A}_1 = (\{x_i\}, \{x_{i'}\}, \{x_{i^*}\}, \{e_{st}\})$ be a feasible configuration such that for any other feasible configuration $(\{y_i\}, \{y_{i'}\}, \{y_{i^*}\}, \{e'_{st}\})$, it holds*

$$\sum_i x_i \geq \sum_i y_i.$$

The configuration $\mathcal{A}_2 = (\{x_i\}, \{w_{i'}\}, \{x_{i^*}\}, \{e'_{st}\})$ is a feasible configuration corresponding to a cycle/path packing containing the maximum number of non-altruistic vertices in $V \setminus B$, where $e'_{st} = e_{st}$ if $s \neq t$ and $e'_{st} = w_{i'} - x_{i'}$ if $s = t = i'$.

Proof. First, $\mathcal{A}_2$ is a feasible configuration by Lemma 3. Second, for any feasible configuration $(\{y_i\}, \{y_{i'}\}, \{y_{i^*}\}, \{e''_{st}\})$, it holds that $\sum_i x_i \geq \sum_i y_i$ by the definition of $\mathcal{A}_1$ and $\sum_i w_{i'} \geq \sum_i y_{i'}$ by Lemma 2 and Constraint C1. Therefore, it holds that $\sum_i x_i + \sum_i w_{i'} \geq \sum_i y_i + \sum_i y_{i'}$ and the cycle/path packing corresponding to $\mathcal{A}_2$ contains the maximum number of non-altruistic vertices. □

Lemma 4 provides a way to find a feasible configuration corresponding to a cycle/path packing containing the maximum number of non-altruistic vertices. We only need to maximize $\sum_i x_i$ satisfying Constraints C1-C6. Then, we can build an integer programming below, which contains $|V_Q| + |A_Q|$ integer variables: $\{x_i\}, \{x_{i'}\}, \{x_{i^*}\}$ and $\{e_{st}\}$.

$$\begin{aligned} \max \quad & \sum_i x_i \qquad \text{(IP)} \\ \text{s.t.} \quad & C1\text{-}C6. \end{aligned}$$

Assume that $\{\{z_i\}, \{z_{i'}\}, \{z_{i^*}\}, \{e_{st}\}\}$ is an optimal solution to the above IP. By Lemma 4, we know that $(\{z_i\}, \{w_{i'}\}, \{z_{i^*}\}, \{e'_{st}\})$ is a feasible configuration corresponding to an optimal solution to F-KEP.

3.2 The Cycle/Path Decomposition

Next, we show that given an arbitrary feasible configuration, how to construct a cycle/path packing corresponding to it.

Assume that the given feasible configuration is $\mathcal{A} = (\{z_i\}, \{z_{i'}\}, \{z_{i^*}\}, \{e_{st}\})$. For each vertex $v \in V$ in G, we define $In(v)$ and $Out(v)$ as the sum of the values of all the corresponding pure module's in-arcs and out-arcs in the quotient graph Q. Each of the following two cases is called a *packing element* in the compatibility graph G under $\mathcal{A}$:
Case 1. a directed cycle C in G (including a self-loop) such that for any vertex or arc in C, the value in $\mathcal{A}$ for the corresponding pure module or arc type is positive;
Case 2. a directed path P in G started at an altruistic vertex and ended at a vertex v with $In(v) > Out(v)$ such that for any vertex or arc in P, the value in $\mathcal{A}$ for the corresponding pure module or arc type is positive.

Our decomposition algorithm is simple, which iteratively does the follows until all the values in the configuration $\mathcal{A}$ become zero: find a packing element and update the configuration $\mathcal{A}$ by decreasing the value by 1 for each vertex and arc corresponding to a vertex and arc in the packing element. All the packing elements found by the procedure will form a cycle/path packing.

Lemma 5. *Given a feasible configuration and any packing element in the graph. The configuration is still feasible after by decreasing the value by 1 for each vertex and arc corresponding to a vertex and arc in the packing element.*

The correctness of this lemma is based on the following observation: for any vertex v in the packing element except the beginning and ending vertices of a path, the update decreases both $In(v)$ and $Out(v)$ by 1, which will not violate any of C1-C6. For a beginning vertex v of a path, it holds that $Out(v) > 0$ before the update and then it holds that $Out(v) \geq 0$ after the update. For an ending vertex v of a path, it holds that $In(v) > Out(v)$ before the update and then it holds that $In(v) \geq Out(v)$ after the update. Lemma 5 guarantees that after each iteration of the update, the configuration is still feasible. Next, we still need to consider two questions: why proper packing elements always exist during the procedure and how much time needed to find a packing element if it exists. We have the following lemmas.

Lemma 6. *When the configuration is feasible, packing elements always exist and one can be found in linear time.*

We use a depth-first search (DFS) technique to find a packing element when the configuration satisfies C1-C6. The main procedure is as follows:
If there is a vertex $i^* \in V_B$ such that the value x_{i^*} of it is greater than 0, we start from i^*, search by DFS a directed path with all vertices and arcs on it having positive values, until meeting a vertex v with $In(v) > Out(v)$ or a vertex t having been visited. Then we will get a directed path or a directed cycle (started and ended both at t) in Q, which will be corresponding to a packing element of Case 1 or 2 in G;
If the value of any vertex in V_B is 0, we start from any vertex with value > 0, search by DFS a directed path with all vertices and arcs on it having positive values, until meeting a vertex having been visited. Then we will get a directed cycle in Q (corresponding to a packing element of Case 1 in G).

Note that when the configuration satisfies C1-C6, only vertices v in V_B may hold that $In(v) < Out(v)$ and for any vertex $u \in V_B$ it holds that $In(v) = 0$ since there is no arc from a vertex to an altruistic vertex in the graph. When the value of any vertex in V_B is 0, it will hold $In(v) = Out(v)$ for any vertex in the graph, because $In(v) \geq Out(v)$ holds for any vertex v and in global $\sum_{v \in V_Q} In(v) = \sum_{v \in V_Q} Out(v)$. Therefore, during the above DFS, once the directed path enters a vertex, the vertex should be a vertex with value > 0 and there always has an arc from it to another vertex with positive value. So for the latter case, the DFS can always find a proper cycle.

3.3 Running Time Analysis

The most time-consuming part is the IP. It is known that an IP with x variables can be solved in $2^{O(x)}x^x$ time [Dadush, 2012]. In our IP, the number of variables is the number of vertices and arcs in the quotient graph Q, which is bounded by θ^2 ($\theta = |V_Q|$). To construct a cycle/apth packing based on a feasible configuration can be done in $O(n(n+m))$ time, since there are at most n packing elements. In total, the algorithm takes $O(2^{O(\theta^2)}\theta^{2\theta^2} + n(n+m))$ time. It is FPT by taking θ as the parameter.

4 Conclusion

This paper contributes to complexity and algorithms with theoretically proved running-time bounds for the kidney exchange problem. We design a uniform single-exponential exact algorithm for the kidney exchange problem with any constraints on the maximum length of the cycles and chains and an FPT algorithm for the kidney exchange problem without length constraints on the cycles and chains. Both results significantly improve previous theoretical bounds. It will be interesting to further study the effectiveness of the two algorithms on realistic instances.

Acknowledgements

The work is supported by the National Natural Science Foundation of China, under grants 61772115 and 61370071.

References

[Abraham *et al.*, 2007] David J Abraham, Avrim Blum, and Tuomas Sandholm. Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 295–304. ACM, 2007.

[Anderson *et al.*, 2015] Ross Anderson, Itai Ashlagi, David Gamarnik, and Alvin E Roth. Finding long chains in kidney exchange using the traveling salesman problem. *Proceedings of the National Academy of Sciences*, 112(3):663–668, 2015.

[Biro *et al.*, 2009] Péter Biro, David F Manlove, and Romeo Rizzi. Maximum weight cycle packing in directed graphs, with application to kidney exchange programs. *Discrete Mathematics, Algorithms and Applications*, 1(04):499–517, 2009.

[Björklund *et al.*, 2007] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *Proceedings of the Annual Acm Symposium on Theory of Computing*, pages 67–74, 2007.

[Constantino *et al.*, 2013] Miguel Constantino, Xenia Klimentova, Ana Viana, and Abdur Rais. New insights on integer-programming models for the kidney exchange problem. *European Journal of Operational Research*, 231(1):57–68, 2013.

[Dadush, 2012] Daniel Nicolas Dadush. *Integer Programming, Lattice Algorithms, and Deterministic Volume Estimation*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2012.

[Dickerson *et al.*, 2016] John P. Dickerson, David F. Manlove, Benjamin Plaut, Tuomas Sandholm, and James Trimble. Position-indexed formulations for kidney exchange. In *ACM Conference on Economics and Computation*, pages 25–42, 2016.

[Dickerson *et al.*, 2017] John P Dickerson, Aleksandr M Kazachkov, Ariel D Procaccia, and Tuomas Sandholm. Small representations of big kidney exchange graphs. In *AAAI*, pages 487–493, 2017.

[Downey and Fellows, 2013] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[Glorie *et al.*, 2014] Kristiaan M Glorie, J Joris van de Klundert, and Albert PM Wagelmans. Kidney exchange with long chains: An efficient pricing algorithm for clearing barter exchanges with branch-and-price. *Manufacturing & Service Operations Management*, 16(4):498–512, 2014.

[Gruessner and Sutherland, 2005] Angelika C Gruessner and David ER Sutherland. Pancreas transplant outcomes for united states (US) and non-us cases as reported to the united network for organ sharing (UNOS) and the international pancreas transplant registry (IPTR) as of june 2004. *Clinical transplantation*, 19(4):433–455, 2005.

[Jia *et al.*, 2017] Zhipeng Jia, Pingzhong Tang, Ruosong Wang, and Hanrui Zhang. Efficient near-optimal algorithms for barter exchange. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 362–370, 2017.

[Klimentova *et al.*, 2014] Xenia Klimentova, Filipe Alvelos, and Ana Viana. A new branch-and-price approach for the kidney exchange problem. In *International Conference on Computational Science and Its Applications*, pages 237–252. Springer, 2014.

[Krivelevich *et al.*, 2007] Michael Krivelevich, Zeev Nutov, Mohammad R. Salavatipour, Jacques Verstraete Yuster, and Raphael Yuster. Approximation algorithms and hardness results for cycle packing problems. *ACM Transactions on Algorithms*, 3(4):48, 2007.

[Li *et al.*, 2014] Jian Li, Yicheng Liu, Lingxiao Huang, and Pingzhong Tang. Egalitarian pairwise kidney exchange: fast algorithms via linear programming and parametric flow. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 445–452, 2014.

[Mak-Hau, 2017] Vicky Mak-Hau. On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. *Journal of combinatorial optimization*, 33(1):35–59, 2017.

[Manlove and O'Malley, 2012] David F. Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the uk: Algorithms and experimentation. In Ralf Klasing, editor, *Experimental Algorithms*, pages 271–282, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Manlove and O'Malley, 2014] David F. Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the UK: algorithms and experimentation. *ACM Journal of Experimental Algorithmics*, 19(1), 2014.

[McConnell and de Montgolfier, 2005] Ross M. McConnell and Fabien de Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Applied Mathematics*, 145(2):198–209, 2005.

[Rapaport, 1986] F. T. Rapaport. The case for a living emotionally related international kidney donor exchange registry. *Transplantation Proceedings*, 18(2):5–9, 1986.

[Roth *et al.*, 2005a] Alvin E. Roth, Tayfun Snmez, and M. Utku nver. A kidney exchange clearinghouse in new england. *The American Economic Review*, 95(2):376–380, 2005.

[Roth *et al.*, 2005b] Alvin E. Roth, Tayfun Sönmez, and M. Utku Ünver. Pairwise kidney exchange. *J. Economic Theory*, 125(2):151–188, 2005.

[Segev *et al.*, 2005] Dorry L Segev, Sommer E Gentry, Daniel S Warren, Brigitte Reeb, and Robert A Montgomery. Kidney paired donation and optimizing the use of live donor organs. *JAMA*, 293(15):1883–1890, 2005.

[Sönmez and Ünver, 2014] Tayfun Sönmez and M. Utku Ünver. Altruistically unbalanced kidney exchange. *Journal of Economic Theory*, 152(1):105–129, 2014.