A Faster Deterministic Algorithm for Kidney Exchange via Representative Set

Kangyi Tian and **Mingyu Xiao***
School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
kangyitian947@gmail.com, myxiao@gmail.com

Abstract

The Kidney Exchange Problem is a prominent challenge in healthcare and economics, arising in the context of organ transplantation. It has been extensively studied in artificial intelligence and optimization. In a kidney exchange, a set of donor-recipient pairs and altruistic donors are considered, with the goal of identifying a sequence of exchanges—comprising cycles or chains starting from altruistic donors—such that each donor provides a kidney to the compatible recipient in the next donor-recipient pair. Due to constraints in medical resources, some limits are often imposed on the lengths of these cycles and chains. These exchanges create a network of transplants aimed at maximizing the total number, t, of successful transplants. Recently, this problem was deterministically solved in $O^*(14.34^t)$ time (IJCAI 2024). In this paper, we introduce the representative set technique for the Kidney Exchange Problem, showing that the problem can be deterministically solved in $O^*(6.855^t)$ time.

1 Introduction

Kidney disease has become an increasingly significant global issue. According to international consensus [Francis *et al.*, 2024], approximately 850 million people worldwide are estimated to suffer from kidney disease. Patients with kidney disease are typically treated through either *dialysis* or *kidney transplantation*. In terms of therapeutic outcomes, kidney transplantation is generally preferred over dialysis for treating kidney disease. However, there is a substantial gap between the number of kidney donors and recipients. Patients unable to find a compatible donor for a transplant are placed on a waiting list. Unfortunately, as the waiting list grows longer each year, many patients miss the optimal window for treatment and may even lose their lives while waiting.

Often, patients on the waiting list have family members or friends willing to donate a kidney, but the kidney may not be compatible. To address this challenge, Kidney Paired Donation (KPD), also known as Kidney Exchange, was introduced by Rapaport [Rapaport, 1986]. Since its introduction in 1986, KPD has been implemented in several countries and has helped many people receive compatible kidneys [Pahwa *et al.*, 2012; Dickerson *et al.*, 2016].

In Kidney Paired Donation, each participant is either an altruistic donor or a donor-recipient pair. To ensure that every recipient receives a compatible kidney, kidney donations are organized into *chain models* or *cycle models*. In chain models, an altruistic donor first donates a kidney to the next donor-recipient pair. Then, each subsequent donor-recipient pair in the chain receives a kidney from the previous pair and donates a kidney to the next pair, if one exists. In cycle models, each pair receives a kidney from the previous pair and donates a kidney to the next pair. It is important to note that in both models, a pair can choose to exit the program after receiving a kidney, as they have no legal obligation to donate. From a fairness perspective, we aim to avoid this, particularly in cycle models. To prevent this issue, all kidney transplant surgeries are ideally performed simultaneously. However, due to medical and logistical constraints, there can only be a limited number of surgeries at one time. As a result, the length of chains and cycles cannot be excessively long.[1]

To optimize the number of kidney exchanges, the central problem in Kidney Paired Donation can be framed as follows. First, a compatibility graph is constructed based on medical compatibility criteria. In this graph, each vertex represents either an altruistic donor or a donor-recipient pair, and a directed edge exists between two vertices if one can donate a compatible kidney to the other. We can assume that self-loops do not exist in the compatibility graph. Our computational task then becomes the problem of finding the largest vertex-disjoint packing of cycles and paths starting from an altruistic vertex in the compatibility graph, subject to restrictions on the length of paths and cycles. This problem is referred to as the Kidney Exchange Problem.

We now present the formal definition of the Kidney Exchange Problem. Let G be the compatibility graph, and B be the set of altruistic vertices. We are given two nonnegative integers, l_p and l_c. The *length* of a path or cycle is defined as the number of edges it contains. A path is *feasible* if it starts at an altruistic vertex in B and has a length of at most l_p, while

*Corresponding Author

[1]The length restriction for chains is generally less stringent than for cycles [Anderson *et al.*, 2015; Glorie *et al.*, 2014].

a cycle is *feasible* if it does not contain any vertices in B and its length does not exceed l_c. A path-cycle packing is a set of vertex-disjoint cycles and paths. A path-cycle packing is *feasible* if all cycles and paths in it are feasible. An instance of KEP is denoted by (G, B, l_p, l_c, t) .

Kidney Exchange Problem (KEP)
Input: A directed graph $G = (V, A)$ without self-loops, an altruistic vertex set $B \subseteq V$ without any edge entering a vertex in B, two nonnegative integers l_p and l_c, and a target integer t.
Output: A feasible path-cycle packing such that the total length of cycles and paths in it is at least t.

1.1 Related Work

The Kidney Exchange Problem (KEP) has been extensively studied since its introduction in [Rapaport, 1986]. In most studies, KEP is typically modeled as a packing problem involving vertex-disjoint chains and cycles. Due to variations in the types of packings and constraints on the lengths of chains and cycles, many variants have been proposed in recent years [Constantino *et al.*, 2013; Manlove and O'Malley, 2014; Glorie *et al.*, 2014; Anderson *et al.*, 2015].

Numerous practical algorithms have been developed to address the Kidney Exchange Problem. A traditional approach to solving KEP is through Integer Programming. Based on Integer Programming, [Manlove and O'Malley, 2014] designed an algorithm and deployed it in software, which has been used to identify kidney exchange pairs in real-world applications. More recently, a new method using graph neural networks was introduced to KEP by [Pimenta *et al.*, 2023].

Theoretical studies have shown that KEP is NP-hard, even for some very restricted versions [Krivelevich *et al.*, 2007; Abraham *et al.*, 2007; Biró *et al.*, 2009]. Due to the NP-hardness of KEP, approximation algorithms [Krivelevich *et al.*, 2007; Jia *et al.*, 2017], randomized algorithms [Lin *et al.*, 2019], and parameterized algorithms [Xiao and Wang, 2018; Maiti and Dey, 2022; Hébert-Johnson *et al.*, 2024] have been introduced to handle its computational complexity.

We focus on KEP parameterized by the number of covered recipients t. Although this problem has been studied for many years, whether it is fixed-parameter tractable (FPT) with respect to t remained unresolved until recently.

[Maiti and Dey, 2022] was the first work to show that KEP is FPT by employing Color-Coding. Their randomized algorithm runs in time $O^*(8^t)$ and can be derandomized to run in $O^*(161^t)$. More recently, using the technique of arithmetic circuit representation of polynomials [Williams, 2009], [Hébert-Johnson *et al.*, 2024] developed a randomized algorithm with an improved running time of $O^*(4^t)$. After derandomization, the running time becomes $O^*(14.34^t)$. Very recently, Banik et al. [Banik *et al.*, 2025] proposed a simple algorithm with a claimed runtime of $O^*(10.88^t)$. However, their proof is incomplete, and we observe that their algorithm in fact requires $O^*(10.88^{2t})$ time. The discrepancy arises from their use of the color-coding technique introduced for KEP in [Maiti and Dey, 2022], where they assign t colors to the vertices. This is insufficient, because a feasible solution may contain up to $2t - 2$ vertices—for instance, two disjoint cycles each of length $t - 1$. Thus, one must instead use roughly $2t$ colors, which increases the exponential dependency from 10.88^t to 10.88^{2t}. A similar issue explains why the randomized algorithm in [Hébert-Johnson *et al.*, 2024] runs in $O^*(4^t)$ time rather than $O^*(2^t)$.

1.2 Our Contribution and Method

The main contribution of this paper is to introduce the *representative set* technique to KEP, leading to a *deterministic algorithm with running time* $O^*(6.855^t)$. This improves upon the previous best deterministic bound of $O^*(14.34^t)$ [Hébert-Johnson *et al.*, 2024].

In fact, by applying our Lemma 3, we can directly improve the running time of Maiti and Dey's randomized algorithm to $O^*(4^t)$. See Theorem 3 in the full version of this paper [Tian and Xiao, 2026] for the detailed proof. However, straightforward derandomization leads to a time bound of $O^*(30^t)$. Achieving better deterministic performance requires more advanced techniques.

From a technical perspective, our approach begins with a dynamic programming (DP) algorithm that step-by-step constructs all feasible path-cycle packings of total length at most t. However, a direct implementation faces a major bottleneck: the number of intermediate packings can be exponential and is not bounded by a function of t. Consequently, this approach does not yield a truly parameterized algorithm.

To address this, we compress the set of feasible path-cycle packings at each DP state. The key insight is that we only care about the set of recipients covered by the paths and cycles, not their specific structure. Thus, one set of cycles and paths can be replaced with another as long as they cover the same vertices.

To enable this compression, we employ the *representative set* technique [Fomin *et al.*, 2016]. A representative set is a carefully selected subset that captures the essential combinatorial properties of the larger family. In our setting, it enables us to preserve optimality while significantly reducing the search space.

Although representative sets have been well studied [Fomin *et al.*, 2016], existing methods cannot be directly applied to KEP due to the presence of local constraints and problem-specific structural features. In particular, our dynamic programming must be carefully designed to ensure that no valid solutions are lost during the compression step. As a result, our correctness proof and running time analysis differ significantly from those of earlier applications of the representative set method.

2 Preliminaries

Let $G = (V, A)$ denote a directed graph. We will let $n = |V|$. The graph G will be used to denote the the compatibility graph in KEP. We will always use B to represent the set of all altruistic donors. All vertices in B are sources, i.e., no edge enters a vertex in B.

A *path* of length $k - 1$ is sequence of distinctive vertices $v_1 v_2 \cdots v_k$ such that $(v_i, v_{i+1}) \in A$ for all $i = 1, 2, \cdots, k -$

1. For a path $v_1v_2\cdots v_k$, if there is also an edge $(v_k, v_1) \in A$, we call $v_1v_2\cdots v_kv_1$ a *cycle* of length k. For a path $P = v_1v_2\cdots v_k$ (resp., a cycle $C = v_1v_2\cdots v_kv_1$), we let $V(P) = \{v_1, v_2, \cdots, v_k\}$ (resp., $V(C) = \{v_1, v_2, \cdots, v_k\}$). For a path-cycle packing $\mathcal{D}$, we define the *size* $|\mathcal{D}|$ as the number of cycles and paths in $\mathcal{D}$. We also define the *length* $c(\mathcal{D})$ as the sum of lengths of all paths and cycles in $\mathcal{D}$. Note that $c(\mathcal{D})$ may be smaller than the number of vertices appearing in $\mathcal{D}$. A singleton $\{D\}$ may be simply written as D.

Let $\mathcal{F}$ be a collection of sets and S be another set. we define $\mathcal{F} \uplus S = \{X \cup S | X \in \mathcal{F}, X \cap S = \emptyset\}$. Let $\mathcal{S}$ be a family of sets and the cardinality $|\mathcal{S}|$ denote the number of sets in $\mathcal{S}$. We say that a family is *p-family* if $|X| = p$ holds for all sets $X \in \mathcal{S}$. We use $O^*(f(k))$ to denote $f(k)n^{O(1)}$.

Representative Set. Representative set is one of the main techniques used in this paper. Usually, representative sets are defined using matroids [Fomin *et al.*, 2016]. However, in this paper, we only use the technique in a simpler situation. So, we define representative sets in a simpler way without using the concept of matroids.

Definition 1 (q-Representative Set)**.** *Given a ground set E and a p-family $\mathcal{S}$ of subsets of E. A subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is* q-representative *for $\mathcal{S}$ if the following holds: for every set $Y \subseteq E$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y. We write $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ to denote that $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is q-representative for $\mathcal{S}$.*

In the definition, we require that $\mathcal{S}$ to be a p-family. The requirement on the size seems not used in the definition. However, we specify the information because this is related to the running time and the size of the representative set.

[Fomin *et al.*, 2016] proposed the following lemmas to compute the q-representative sets efficiently. We will also use them.

Lemma 1 ([Fomin *et al.*, 2016])**.** *Suppose that $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ are p-families. If $\mathcal{X} \subseteq^q_{rep} \mathcal{Y}$ and $\mathcal{Y} \subseteq^q_{rep} \mathcal{Z}$, then $\mathcal{X} \subseteq^q_{rep} \mathcal{Z}$.*

Lemma 2 ([Fomin *et al.*, 2016])**.** *There is an algorithm that, given a p-family $\mathcal{S}$ of sets over a ground E of size n, an integer q, computes in time*

$$\mathcal{O}\big(|\mathcal{S}| \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n\big)$$

a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ such that $|\widehat{\mathcal{S}}| \leq x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}$ and $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$.

3 A Faster Deterministic Algorithm

In this section, we will introduce our algorithm. We are going to prove the following main result.

Theorem 1. *KEP can be deterministically solved in $O^*(6.855^t)$ time.*

In our main algorithm, we first handle the case when $l_p \geq t$ or $l_c \geq t$. For this case, we check whether there is a feasible path of length t or a feasible cycle of length at least t. If so, we find a solution covering at least t recipients and the problem is solved. If no, we can set that $l_p = \min(l_p, t-1)$ and $l_c = \min(l_c, t-1)$. Finding feasible paths and cycles of length at least t can be done by slightly modifying the known algorithms of k-Path and Long Directed Cycle in [Fomin *et al.*, 2016]. To find a feasible k-path, we only need to modify the algorithm for k-Path by restricting starting points of paths in B in the dynamic programming. For a feasible cycle of length at least t, we can use their algorithm to find the smallest cycles of length at least t. Thus, we can check whether there is a feasible path (resp., a feasible cycle) of length $\geq t$ in time $O(2.619^t n \log^2 n)$ (resp., $O(6.75^{t+o(t)}mn^2)$) according to [Fomin *et al.*, 2016].

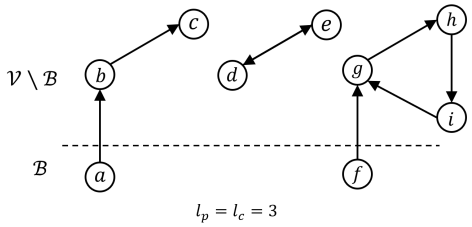

Figure 1: A compatibility graph

Next, we always assume that $l_p < t$ and $l_c < t$. The algorithm will contain three major parts. In Section 3.1, we first introduce the concept of *semi-feasible path-cycle packing*, which is an important concept used in the middle steps of our algorithm. In Section 3.2, we introduce a dynamic programming algorithm to compute semi-feasible path-cycle packings. This algorithm will solve KEP with $l_p < t$ and $l_c < t$ in $O^*(2^n)$ time. After that, in Section 3.3, we show how to apply representative sets to compress the number of semi-feasible path-cycle packings in the dynamic programming algorithm. As a result, we will obtain an efficient algorithm for KEP.

3.1 Semi-feasible Path-cycle Packing

Our idea is to use a dynamic programming algorithm to compute feasible path-cycle packings step by step. However, in some meddle steps, we need to store a partial of a feasible path or cycle. Thus, we come up with the concept of semi-feasible path-cycle packing.

A path-cycle packing $\mathcal{D} = \mathcal{P} \cup \mathcal{C} \cup \{D\}$ is *semi-feasible* if it satisfies the following conditions:

- $\mathcal{P}$ is a feasible path packing;
- $\mathcal{C}$ is a feasible cycle packing;
- D is either a feasible path or a feasible cycle or a path of length at most l_c such that $V(D) \cap B = \emptyset$.

We use the example in Figure 1 illustrate semi-feasible path-cycle packings. Let $P_1 = abc$, $P_2 = fgh$, $P_3 = igh$, $C_1 = ded$, and $C_2 = ghig$. Since $l_p = l_c = 3$, we know that P_1, P_2, C_1, and C_2 are all feasible. Then the path-cycle packing $\mathcal{D}_1 = \{P_1, C_1, P_2\}$ is both semi-feasible and feasible. The path-cycle packing $\mathcal{D}_2 = \{P_1\} \cup \{C_1\} \cup \{P_3\}$ is not a feasible packing since P_3 is not feasible. However, $\mathcal{D}_2$ is a semi-feasible path-cycle packing since P_3 is of length $\leq l_c$ and $V(P_3) \cap B = \emptyset$.

3.2 A Dynamic Programming Algorithm

In this section, we are going to use a dynamic programming method to solve KEP with $l_p < t$ and $l_c < t$ in $O^*(2^n)$

	Semi-feasible Packing $\mathcal{D} = \mathcal{P} \cup \mathcal{C} \cup \{D\}$			Vertex Subset $V(\mathcal{D})$
	$\mathcal{P}$	$\mathcal{C}$	D	
$V(\mathcal{D}) \in \mathcal{F}^{ab}_{1,1}$	$\emptyset$	$\emptyset$	ab	$\{a, b\}$
$V(\mathcal{D}) \in \mathcal{F}^{ac}_{2,2}$	$\emptyset$	$\emptyset$	abc	$\{a, b, c\}$
$V(\mathcal{D}) \in \mathcal{F}^{de}_{3,1}$	$\{abc\}$	$\emptyset$	de	$\{a, b, c, d, e\}$
$V(\mathcal{D}) \in \mathcal{F}^{ed}_{3,1}$	$\{abc\}$	$\emptyset$	ed	
$V(\mathcal{D}) \in \mathcal{F}^{fg}_{3,1}$	$\{abc\}$	$\emptyset$	fg	$\{a, b, c, f, g\}$
	$\emptyset$	$\{ded\}$	fg	$\{d, e, f, g\}$
$V(\mathcal{D}) \in \mathcal{F}^{dd}_{4,2}$	$\{abc\}$	$\emptyset$	ded	$\{a, b, c, d, e\}$
$V(\mathcal{D}) \in \mathcal{F}^{fg}_{5,1}$	$\{abc\}$	$\{ded\}$	fg	$\{a, b, c, d, e, f, g\}$

Table 1: Some illustrations of $\mathcal{F}^{vu}_{k,l}$ for the instance in Figure 1. Column 1 shows $\mathcal{F}^{vu}_{k,l}$ with different values of k, l, v and u; Columns 2,3 and 4 display the semi-feasible packing $\mathcal{D} = \mathcal{P} \cup \mathcal{C} \cup \{D\}$; Column 5 displays the corresponding vertex subset $V(\mathcal{D})$.

time. This running time bound is not new. Previously, [Xiao and Wang, 2018] showed that KEP can be solved in the same running time bound by using subset convolution. However, their approach cannot combine the representative set method to get the result in this paper. Next, we prove the following theorem.

Theorem 2. *There is a dynamic programming algorithm that can deterministically solve KEP with $l_p < t$ and $l_c < t$ in $O^*(2^n)$ time.*

We use KEPalg to denote the dynamic programming algorithm in Theorem 2. We also recall the assumption that $l_p < t$ and $l_c < t$. Under this assumption, we have the following property to bound the length of a feasible path-cycle packing.

Lemma 3. *Assume that $l_p < t$ and $l_c < t$. A KEP instance (G, B, l_p, l_c, t) is a yes-instance if and only if there exists a feasible path-cycle packing $\mathcal{D}$ such that $t \leq c(\mathcal{D}) \leq 2t$.*

Proof. If there is a feasible path-cycle packing $\mathcal{D}$ such that $t \leq c(\mathcal{D}) \leq 2t$, then the instance is a yes-instance by the definition of the problem.

Now we prove the other direction. If (G, B, l_p, l_c, t) is a yes-instance, then there exists a feasible cycle-path packing $\mathcal{D}$ such that $c(\mathcal{D}) \geq t$. We assume that $\mathcal{D}$ is the packing with the minimum $c(\mathcal{D})$ satisfying the above requirement. We are done if it also holds that $c(\mathcal{D}) \leq 2t$. Otherwise, we have that $c(D) > 2t$. Since $l_c < t$ and $l_p < t$, it holds that $c(D) < t$ for any path or cycle $D \in \mathcal{D}$. Let $\mathcal{D}' = \mathcal{D} \setminus D$. since $c(\mathcal{D}') = c(\mathcal{D}) - c(D) \geq t$, we obtain a new feasible path-cycle packing $\mathcal{D}'$ such that $t \leq c(\mathcal{D}') < c(\mathcal{D})$, which is a contradiction to the minimality of $\mathcal{D}$. So, we know that it holds $t \leq c(\mathcal{D}) \leq 2t$. □

This lemma says that there exists a path-cycle packing of length at most $2t$ if the input is a yes-instance. In the process of our dynamic programming, we will compute the vertex subsets of all possible semi-feasible packings with length at most $2t$. Finally, we answer the instance by checking whether there is a feasible path-cycle packing of length at least t among all semi-feasible packings we compute.

We can see that a semi-feasible path-cycle packing $\mathcal{D} = \mathcal{P} \cup \mathcal{C} \cup \{D\}$ is feasible if and only if D is a feasible path or cycle. To obtain the ultimate path-cycle packing, we will extend the packing by either increasing the length of the path D by 1 or moving D into $\mathcal{P} \cup \mathcal{C}$ and setting a new edge as D. This operation varies depending on the element D. If D is a path (but not a feasible path), we will add one edge to it such that it will either become a feasible cycle or a longer path. If D is a feasible cycle, we move D to $\mathcal{C}$ and take a new edge (disjoint with any element in $\mathcal{D}$) as D. If D is a feasible path, we can either increase the length of D by adding an edge at its end or move D to $\mathcal{P}$ and take a new edge (disjoint with any element in $\mathcal{D}$) as D.

For a semi-feasible packing $\mathcal{D} = \mathcal{P} \cup \mathcal{C} \cup D$, let $V(\mathcal{D}) = V(\mathcal{P}) \cup V(\mathcal{C}) \cup V(D)$. Let D^{vu}_l denote a path from v to u of length l if $v \neq u$ and a cycle containing v of length l if $u = v$. We define

$$\mathcal{F}^{vu}_{k,l} = \{V(\mathcal{D}) \mid \mathcal{D} = \mathcal{P} \cup \mathcal{C} \cup \{D^{vu}_l\} \wedge c(\mathcal{D}) = k\}.$$

Table 1 provides some illustrations of $\mathcal{F}^{vu}_{k,l}$ for the example in Figure 1. Our dynamic programming algorithm indeed stores the vertex sets of all possible semi-feasible path-cycle packings by computing $\mathcal{F}^{vu}_{k,l}$ for all $1 \leq k \leq 2t$, $v \in V$, $u \in V \setminus B$ and $1 \leq l \leq \max(l_p, l_c)$. The following Lemma 4 says that we can solve KEP by checking $\mathcal{F}^{vu}_{k,l}$.

Lemma 4. *A KEP instance (G, B, l_p, l_c, t) with $l_p, l_c < t$ is a yes-instance if and only if there exists a tuple (k, v, u, l) such that $t \leq k \leq 2t$, $\mathcal{F}^{vu}_{k,l} \neq \emptyset$, and one of the following two conditions holds:*

- *$v \in B$, $u \in V \setminus B$ and $l \leq l_p$;*
- *$v = u \in V \setminus B$ and $l \leq l_c$.*

Proof. We first prove the forward direction. By Lemma 3, we know that if (G, B, l_p, l_c, t) is a yes-instance, there must be a feasible path-cycle packing $\mathcal{D}$ such that $t \leq c(\mathcal{D}) \leq 2t$. Since feasible path-cycle packing is also semi-feasible, we know that there exists $\mathcal{F}^{vu}_{k,l}$ satisfying the above conditions that contains $V(\mathcal{D})$.

We consider the other direction. Now we have $\mathcal{D} = (\mathcal{P}, \mathcal{C}, D^{vu}_l)$ with $c(\mathcal{D}) \geq t$. Since it satisfies the above conditions, we know that D^{vu}_l is actually a feasible path or cycle. Thus $\mathcal{D}$ is a feasible path-cycle packing with $c(\mathcal{D}) \geq t$, which means (G, B, l_p, l_c, t) is a yes-instance. □

We have the following Lemma 5 to compute $\mathcal{F}^{vu}_{k,l}$.

Lemma 5. *There exists a dynamic programming algorithm that can compute $\mathcal{F}^{vu}_{k,l}$ for all $1 \leq k \leq 2t$, $v \in V$, $u \in V \setminus B$ and $l \leq \max(l_c, l_p)$ in time $O^*(2^n)$.*

Proof. Recall that our algorithm is a dynamic programming algorithm. We show how to compute $\mathcal{F}^{vu}_{k,l}$ for different values of k, l, v and u by either setting an initial value or giving a transition equation.

We distinguish five cases. In fact, if it is not of the first four cases, then there is no packing satisfying the condition of $\mathcal{F}^{vu}_{k,l}$ and we will simply let $\mathcal{F}^{vu}_{k,l} = \emptyset$. This is Case (v).

Case (i): $k = l = 1$, and $(v, u) \in A(G)$.

If $\mathcal{D} = \{\mathcal{P}, \mathcal{C}, D^{vu}_1\}$ is a semi-feasible path-cycle packing in $\mathcal{F}^{vu}_{1,1}$, we know that $\mathcal{P} = \mathcal{C} = \emptyset$. Notice that $v \neq u$ since there are no self-loops in G and $(v, u) \in G$. The only possible set in $\mathcal{F}^{vu}_{1,1}$ is $\{v, u\}$. Therefore, we let $\mathcal{F}^{vu}_{1,1} = \{\{v, u\}\}$.

Case (ii): $k > l = 1$, and $(v, u) \in A(G)$.

In this case, for any semi-feasible packing $\mathcal{D} = \{\mathcal{P}, \mathcal{C}, D_1^{vu}\}$ such that $V(\mathcal{D}) \in \mathcal{F}_{k,1}^{vu}$, it holds that $\mathcal{P} \cup \mathcal{C} \neq \emptyset$ since $k > l$. We assume that $D_{l'}^{v'u'}$ is an arbitrary feasible path or cycle in $\mathcal{P} \cup \mathcal{C}$ (let $v' = u'$ if $D_{l'}^{v'u'}$ is a cycle, otherwise $v' \neq u'$). Then the vertex subset of $\mathcal{D}' = \{\mathcal{P} \setminus \{D_{l'}^{v'u'}\}, \mathcal{C} \setminus \{D_{l'}^{v'u'}\}, D_{l'}^{v'u'}\}$ must be contained in $\mathcal{F}_{k-1,l'}^{v'u'}$. Notice that $v \neq u$ since $l = 1$. Thus, we can compute $\mathcal{F}_{k,1}^{vu}$ by enumerating all possible v', u' and l':

$$\mathcal{F}_{k,1}^{vu} = \bigcup_{v' \in B, u' \notin B,\ 1 \leq l' \leq l_p} \mathcal{F}_{k-1,l'}^{v'u'} \uplus \{v, u\} \ \cup \bigcup_{v' \notin B,\ 1 \leq l' \leq l_c} \mathcal{F}_{k-1,l'}^{v'v'} \uplus \{v, u\}. \tag{1}$$

Case (iii): $k \geq l > 1$, $v \neq u$, $l \leq l_p$ if $v \in B$, and $l \leq l_c$ if $v \notin B$.

In this case, for any semi-feasible packing $\mathcal{D} = \{\mathcal{P}, \mathcal{C}, D_l^{vu}\}$ such that $V(\mathcal{D}) \in \mathcal{F}_{k,l}^{vu}$, we have that $\mathcal{D}_l^{vu}$ is a path of length at least 2 if $v \neq u$ or a cycle if $v = u$. Let w be the predecessor vertex of u and D_{l-1}^{vw} be the path obtained by removing vertex u from D_l^{vu}. Then the vertex subset of the semi-feasible packing $\mathcal{D}' = \{\mathcal{P}, \mathcal{C}, D_{l-1}^{vw}\}$ must be contained in $\mathcal{F}_{k-1,l-1}^{vw}$. Thus, we can compute $\mathcal{F}_{k,l}^{vu}$ by enumerating all possible predecessor vertices w:

$$\mathcal{F}_{k,l}^{vu} = \bigcup_{w \neq v,\ (w,u) \in A(G)} \mathcal{F}_{k-1,l-1}^{vw} \uplus \{u\}. \tag{2}$$

Case (iv): $k \geq l > 1$, $v = u \notin B$, and $l \leq l_c$.

This case is similar to Case (iii) and we also handle it by enumerating all possible predecessors of vertices v:

$$\mathcal{F}_{k,l}^{vv} = \bigcup_{w \neq v,\ (w,v) \in A(G)} \mathcal{F}_{k-1,l-1}^{vw}. \tag{3}$$

Case (v): None of the above four cases holds. It is easy to verify that there is no corresponding packing satisfying the condition. We simply let $\mathcal{F}_{k,l}^{vu} = \emptyset$.

Next, we analyze the running time bound of the algorithm. Note that our running time bound is not directly obtained by evaluating the number of $\mathcal{F}_{k,l}^{vu}$ and the time to compute each $\mathcal{F}_{k,l}^{vu}$. We need a refined analysis. Note that each vertex subset in some $\mathcal{F}_{k,l}^{vu}$ can be computed in polynomial time. We only need to evaluate how many vertex subsets are contained in $\mathcal{F}_{k,l}^{vu}$. Let $num(k, l, v, u)$ denote the number of vertex subsets in $\mathcal{F}_{k,l}^{vu}$ for values of k, l, v and u. We have that

$$\begin{aligned} &\sum_{k=1}^{\min(2t,n)} \sum_{l=1}^{\max(l_p,l_c)} \sum_{v} \sum_{u} num(k, l, v, u) \\ \leq &\sum_{k=1}^{\min(2t,n)} \sum_{l=1}^{\max(l_p,l_c)} \sum_{v} \sum_{u} \binom{n}{k} \\ \leq &\, n^3 \sum_{k=1}^{n} \binom{n}{k} = O^*(2^n). \end{aligned}$$

The number of vertex subsets is bounded by $O^*(2^n)$ and the running time of the algorithm is bounded by $O^*(2^n)$. □

Now we are ready to prove Theorem 2.

Proof. (of Theorem 2). We prove Theorem 2 by showing that KEPalg can determine where there exists a feasible path-cycle packing of length at least t in time $O^*(2^n)$.

There are two mains parts in KEPalg. In the first part, we compute $\mathcal{F}_{k,l}^{vu}$ for all $1 \leq k \leq 2t$, $1 \leq l \leq \max(l_p, l_c)$, $v \in V$ and $u \in V \setminus B$. By Lemma 5, we know that the computation can be done in time $O^*(2^n)$. In the second part, we answer the instance by checking $\mathcal{F}_{k,l}^{vu}$. If there exists a tuple (k, l, v, u) such that $\mathcal{F}_{k,l}^{vu}$ is nonempty and either the two conditions in Lemma 4 holds, the answer of the instance is yes. The checking can be done without using more time than part 1.

The algorithm only answer the decision version of the problem. It can also output the corresponding feasible path-cycle packing by adding an additional data structure to store the corresponding path-cycle packings at each state. □

3.3 Representative Sets for Kidney Exchange

In this Section, our goal is to obtain a fast algorithm based on KEPalg in Theorem 2. The fast algorithm in this subsection is denoted by KEPalg-rep.

Observe that the exponential factor of the time complexity of KEPalg comes from the cardinality of $\mathcal{F}_{k,l}^{vu}$. Since each collection $\mathcal{F}_{k,l}^{vu}$ might contain at most $\binom{n}{k}$ vertex subsets, the time complexity would roughly be

$$\sum_{k=1}^{\min(2t,n)} \binom{n}{k} \leq \sum_{k=1}^{n} \binom{n}{k} = O(2^n).$$

Notice that we already bound the length of feasible path-cycle packings of KEP in Lemma 3. However, the cardinality of $\mathcal{F}_{k,l}^{vu}$ can only be bounded by $\binom{n}{k}$, which is not tight for our analysis above. This motivates us that we can improve the time complexity by compressing $\mathcal{F}_{k,l}^{vu}$, which might give us a time complexity bounded by t. For example, if we can compress $\mathcal{F}_{k,l}^{vu}$ to make it contain at most $\binom{2t}{k}$ subsets, then the time complexity would be reduced to $\sum_{k=1}^{2t} \binom{2t}{k} = O(4^t)$.

To compress $\mathcal{F}_{k,l}^{vu}$, our main technique is the efficient computation of representative sets in [Fomin *et al.*, 2016]. We first slightly modify the definition of states in our dynamic programming. Recall that D_l^{vu} denote a path from v to u of length l if $v \neq u$ and a cycle containing v of length l if $u = v$. At each state, We define a p-family $\mathcal{F}_{k,l,p}^{vu} = \{V(\mathcal{D}) \mid \mathcal{D} = \mathcal{P} \cup \mathcal{C} \cup D_l^{vu} \wedge c(\mathcal{D}) = k \wedge |V(\mathcal{D})| = p\}$. Note that we can compute $\mathcal{F}_{k,l,p}^{vu}$ for $1 \leq k, p \leq 2t$ similar to the computation of $\mathcal{F}_{k,l}^{vu}$ in Lemma 5.

Let $q = 2t - p$ and $\hat{\mathcal{F}}_{k,l,p}^{vu}$ denote a q-representative set of $\mathcal{F}_{k,l,p}^{vu}$. In our refined algorithm KEPalg-rep, we want to compute $\hat{\mathcal{F}}_{k,l,p}^{vu}$ for $1 \leq k, p \leq 2t$. For the convenience, we use another p-family $\mathcal{N}_{k,l,p}^{vu}$ as an intermediate variable to compute $\hat{\mathcal{F}}_{k,l,p}^{vu}$. To compute $\hat{\mathcal{F}}_{k,l,p}^{vu}$, we first compute $\mathcal{N}_{k,l,p}^{vu}$ by following the equation similar to Lemma 5 and then compute $\hat{\mathcal{F}}_{k,l,p}^{vu} \subseteq_{rep}^{2t-p} \mathcal{N}_{k,l,p}^{vu}$ by Lemma 2.

We recall the five cases in the proof of Lemma 5 and give the initialization and transition equation of $\mathcal{N}^{vu}_{k,l,p}$ and $\hat{\mathcal{F}}^{vu}_{k,l,p}$ directly as follows:

$$\hat{\mathcal{F}}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{N}^{vu}_{k,l,p}; \tag{4}$$

$$\mathcal{N}^{vu}_{k,l,p} = \begin{cases} \{\{v,u\}\} & \text{Case(i)} \\ \bigcup_{v'\in B, u'\in V\setminus B,\ 1\le l'\le l_p} \hat{\mathcal{F}}^{v'u'}_{k-1,l',p-2} \uplus \{v,u\} & \\ \cup \bigcup_{v'\in V\setminus B,\ 1\le l'\le l_c} \hat{\mathcal{F}}^{v'v'}_{k-1,l',p-2} \uplus \{v,u\} & \text{Case(ii)} \\ \bigcup_{w\neq v,\ (w,u)\in A} \hat{\mathcal{F}}^{vw}_{k-1,l-1,p-1} \uplus \{u\} & \text{Case(iii)} \\ \bigcup_{w\neq v,\ (w,v)\in A} \hat{\mathcal{F}}^{vw}_{k-1,l-1,p} & \text{Case(iv)} \\ \emptyset & \text{Case(v)} \end{cases} \tag{5}$$

The following Lemma 6 says that the computation in Equations (4) and (5) gives us a correct representative set of $\mathcal{F}^{vu}_{k,l,p}$.

Lemma 6. $\hat{\mathcal{F}}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{k,l,p}$.

Proof. We recall that the representative relation is transitive by Lemma 1. By Equation 4, we know that $\hat{\mathcal{F}}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{N}^{vu}_{k,l,p}$. If it holds that $\mathcal{N}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{k,l,p}$, then we are done because $\hat{\mathcal{F}}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{N}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{k,l,p}$. Therefore, we only need to prove that $\mathcal{N}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{k,l,p}$.

We prove $\mathcal{N}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{k,l,p}$ by induction.

For $k = 1$, by the initialization equation, it holds that $\mathcal{N}^{vu}_{k,l,p} = \mathcal{F}^{vu}_{k,l,p}$ and $\mathcal{N}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{k,l,p}$.

By induction, we assume that

$$\mathcal{N}^{vu}_{i,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{i,l,p} \text{ for } i = 1\cdots k-1\ (k>1).$$

Next, we prove that $\mathcal{N}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{k,l,p}$.

Without loss of generality, we take Case (iii) as an example and other cases can be handled similarly. For any vertex subset $X \in \mathcal{F}^{vu}_{k,l,p}$ in Case (iii), there must be some family $\mathcal{F}^{vw}_{k-1,l-1,p-1}$ containing $X \setminus \{u\}$.

Suppose that Y is disjoint with X and $|Y| \le 2t-p$. Then $Y \cup \{u\}$ is disjoint with $X \setminus \{u\}$ and $|Y \cup \{u\}| \le 2t-p+1$. By the induction, we have that

$$\hat{\mathcal{F}}^{vw}_{k-1,l-1,p-1} \subseteq^{2t-(p-1)}_{rep} \mathcal{F}^{vw}_{k-1,l-1,p-1}.$$

Thus, there must be a vertex subset $\hat{X} \in \hat{\mathcal{F}}^{vw}_{k-1,l-1,p-1}$ such that $\hat{X}$ is disjoint with $Y \cup \{u\}$. By Equation 5, we know that $\hat{X} \cup \{u\} \in \mathcal{N}^{vu}_{k,l,p}$. Let $\hat{X}' = \hat{X} \cup \{u\}$. Since $\hat{X}$ is disjoint with Y, we have that for any $X \in \mathcal{F}^{vu}_{k,l,p}$, if $|X \cap Y| = \emptyset$ and $|Y| \le 2t-p$, then there exists $\hat{X}' \in \mathcal{N}^{vu}_{k,l,p}$ such that $|\hat{X}' \cap Y| = \emptyset$.

Therefore, we have proven that $\mathcal{N}^{vu}_{k,l,p} \subseteq^{2t-p}_{rep} \mathcal{F}^{vu}_{k,l,p}$. □

The following Lemma 7 indicates that we can solve KEP by checking $\hat{\mathcal{F}}^{vu}_{k,l,p}$.

Lemma 7. *Suppose that we are given a KEP instance (G, B, l_p, l_c, t). If $l_p, l_c < t$, then (G, B, l_p, l_c, t) is a yes-instance if and only if there exists a tuple (k, l, p, v, u) such that $\hat{\mathcal{F}}^{vu}_{k,l,p} \neq \emptyset$ and either of the two conditions holds:*

- *$t \le k \le p \le 2t$, $v \in B$, $u \in V \setminus B$ and $l \le l_p$,*
- *$t \le k \le p \le 2t$, $v = u \in V \setminus B$ and $l \le l_c$.*

Proof. The proof is based on the proof of Lemma 4. We first prove the backward direction. By the definition, we have

$$\hat{\mathcal{F}}^{vu}_{k,l,p} \subseteq \mathcal{F}^{vu}_{k,l,p} \subseteq \mathcal{F}^{vu}_{k,l}.$$

This implies that if $\hat{\mathcal{F}}^{vu}_{k,l,p} \neq \emptyset$, then $\mathcal{F}^{vu}_{k,l} \neq \emptyset$. Note that if a tuple (k,l,p,v,u) satisfies either the conditions in Lemma 7, then the tuple (k,l,v,u) will naturally satisfies either the conditions in Lemma 4. Thus by Lemma 4, we have that (G, B, l_p, l_c, t) is a yes-instance.

Next, we consider the forward direction.

Suppose that (G, B, l_p, l_c, t) is a yes-instance. There exists a vertex subset with the smallest size p_0 in all $\mathcal{F}^{vu}_{k,l}$ satisfying either the two conditions in Lemma 4. We assume that $X_0 \in \mathcal{F}^{v_0u_0}_{k_0,l_0}$ without loss of generality. Since $X \in \mathcal{F}^{v_0u_0}_{k_0,l_0,p_0}$, we have that $\hat{\mathcal{F}}^{v_0u_0}_{k_0,l_0,p_0} \neq \emptyset$ by the definition of representative sets. Then it is enough to prove that $t \le p_0 \le 2t$. Note that $p_0 \ge k_0 \ge t$. Thus we only need to prove that $p_0 \le 2t$.

Let $\mathcal{D}_0$ be the corresponding semi-feasible path-cycle packing of X_0. If there are only cycles in $\mathcal{D}_0$, we have that $p_0 = k_0 \le 2t$. Otherwise, we can assume that there exists a path P in $\mathcal{D}_0$. Since there are no isolated vertices in $\mathcal{D}_0$, it holds that $p_0 \le 2k_0$. If $k_0 = t$, we have that $p_0 \le 2k_0 = 2t$. Thus, we only need to consider the case $k_0 > t$.

We construct another semi-feasible path-cycle packing D' by removing the endpoint of P (If P is a path of length 2, we remove it from $\mathcal{D}$ directly). Without loss of generality, we assume that $V(\mathcal{D}') \in \mathcal{F}^{v'u'}_{k',l'}$. Note that we have $k' = k_0 - 1 \ge t$. The tuple (k', l', v', u') with $\mathcal{F}^{v'u'}_{k',l'} \neq \emptyset$ satisfies either the two conditions in Lemma 4. However, we have that $|V(\mathcal{D}')| < |X_0|$, which is a contradiction to the minimality of X_0. So, we know that it holds $p_0 \le 2t$. □

We use the following Lemma 8 to compute $\mathcal{F}^{vu}_{k,l}$.

Lemma 8. *There exists a dynamic programming algorithm that can compute $\mathcal{N}^{vu}_{k,l,p}$ and $\hat{\mathcal{F}}^{vu}_{k,l,p}$ for all $1 \le k \le p \le 2t$, $v \in V$, $u \in V \setminus B$ and $l \le \max(l_c, l_p)$ in $O^*(6.855^t)$ time.*

Proof. First, notice that the bottleneck of the time complexity in Lemma 8 is the computation of $\hat{\mathcal{F}}^{vu}_{k,l,p}$, i.e., the computation of the representative set of $\mathcal{N}^{vu}_{k,l,p}$ by Lemma 2. Let

$$q = 2t - p \quad \text{and} \quad s_{p,q} = x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)}.$$

By Lemma 2, we have that $|\hat{\mathcal{F}}^{vu}_{k,l,p}| \le s_{p,q}$.

We set $x = \frac{p}{p+2q}$. To simplify our analysis, we first prove the following purely computational result

$$s_{p,q} \le e^2(p+1) \cdot s_{p+1,q-1} \cdot 2^{o(p+q)}. \tag{6}$$

Note that $(1+\frac{1}{t})^t < e$ for all $t > 0$. We have that

$$\begin{aligned}
\frac{s_{p,q}}{s_{p+1,q-1}} &= \frac{(\frac{p}{p+2q})^{-p}(1-\frac{p}{p+2q})^{-q}}{(\frac{p+1}{p+2q-1})^{-(p+1)}(1-\frac{p+1}{p+2q-1})^{-(q-1)}} \\
&= \big(1+\frac{1}{p+2q-1}\big)^{p+q} \cdot \frac{(p+1)^{p+1}}{p^p} \cdot \frac{(2q-2)^{q-1}}{(2q)^q} \\
&\le \big(1+\frac{1}{p+2q-1}\big)^{p+2q-1} \cdot (1+\frac{1}{p})^p (p+1) \\
&\le e^2(p+1).
\end{aligned}$$

Thus, we get (6).

By Equation (5), we have that

$$\begin{aligned}
|\mathcal{N}^{vu}_{k,l,p}| &\le max\{n^2 \cdot 2ts_{p-2,q+2},\ ns_{p-1,q+1}, ns_{p,q}\} \\
&\le 2e^4n^2t(p-1)p \cdot s_{p,q} \\
&\le 8e^4n^2t^3 \cdot s_{p,q}.
\end{aligned}$$

By Lemma 2, the time of computing all $\hat{\mathcal{F}}^{vu}_{k,l,p}$ can be bounded as follows:

$$\begin{aligned}
&\sum_{k=2}^{2t}\sum_{l=1}^{\max(l_c,l_p)}\sum_{v,u}\sum_{p=2}^{2t}|\mathcal{N}^{vu}_{k,l,p}| \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n \\
&\le \sum_{k=2}^{2t}\sum_{l=1}^{t}\sum_{v,u}\sum_{p=2}^{2t} 8e^4n^2t^3 \cdot s_{p,q} \cdot (1-x)^{-q} \cdot 2^{o(p+q)} \cdot \log n \\
&= \sum_{k=2}^{2t}\sum_{l=1}^{t}\sum_{v,u} 8e^4n^2t^3 \cdot 2^{o(p+q)} \cdot \log n \sum_{p=2}^{2t} s_{p,q} \cdot (1-x)^{-q} \\
&= 16e^4n^4t^5 \cdot 2^{o(p+q)} \cdot \log n \sum_{p=2}^{2t} s_{p,q} \cdot (1-x)^{-q} \\
&\le 32e^4n^4t^6 \cdot 2^{o(p+q)} \cdot \log n \max_{p=2}^{2t} s_{p,q} \cdot (1-x)^{-q}.
\end{aligned}$$

Recall that $x = \frac{p}{p+2q} = \frac{p}{4t-p}$. We get that

$$\begin{aligned}
&\max_{p=2}^{2t}\ s_{p,q} \cdot (1-x)^{-q} \\
&= \max_{p=2}^{2t}\ x^{-p}(1-x)^{-2q} \cdot 2^{o(p+q)} \\
&= 2^{o(p+q)} \max_{p=2}^{2t} (\frac{p}{p+2q})^{-p}(1-\frac{p}{p+2q})^{-2q} \\
&= 2^{o(t)} \max_{p=2}^{2t-1} (\frac{4t-p}{p})^{p}(\frac{4t-p}{4t-2p})^{4t-2p}.
\end{aligned}$$

We can assume that $p = \alpha t$ ($\alpha \in (0,2)$) and get

$$\begin{aligned}
&\max_{p=2}^{2t-1} (\frac{4t-p}{p})^{p}(\frac{4t-p}{4t-2p})^{4t-2p} \\
&= \max_{0<\alpha<2} [(\frac{4-\alpha}{\alpha})^{\alpha}(\frac{4-\alpha}{4-2\alpha})^{4-2\alpha}]^t.
\end{aligned}$$

To obtain the time complexity, our goal is to optimize function $f(\alpha) = (\frac{4-\alpha}{\alpha})^{\alpha}(\frac{4-\alpha}{4-2\alpha})^{4-2\alpha}$ when $0 < \alpha < 2$. By direct calculation, we know that $f(\alpha)$ get the maximum value when $\alpha_0 = 2 - \frac{2}{5}\sqrt{5}$ and $f(\alpha_0) < 6.855$. Therefore, we can compute $\mathcal{N}^{vu}_{k,l,p}$ and $\hat{\mathcal{F}}^{vu}_{k,l,p}$ in time $O^*(6.855^t)$. □

Finally, we are ready to prove Theorem 1.

Proof. [**Proof of Theorem 1**] We prove Theorem 1 by showing that the algorithm KEPALG-REP can determine whether there exists a feasible path-cycle packing of total length at least t in time $O^*(6.855^t)$. The algorithm KEPALG-REP consists of three main components:

Part 1: Handling Large Paths or Cycles. We first check whether $l_c \ge t$ or $l_p \ge t$. In this case, if there exists a feasible cycle (resp., path) of length at least t when $l_c \ge t$ (resp., $l_p \ge t$), we return YES and output the corresponding cycle or path. Otherwise, we safely set $l_c = \min(l_c, t-1)$ and $l_p = \min(l_p, t-1)$. To compute the feasible paths and cycles, we use the algorithm from [Fomin *et al.*, 2016], which runs in time $O^*(6.75^{t+o(t)})$.

Part 2: Computing Table Entries. Next, we compute the sets $\mathcal{N}^{vu}_{k,\ell,p}$ and $\hat{\mathcal{F}}^{vu}_{k,\ell,p}$ for all $1 \le k \le p \le 2t$, $1 \le \ell \le \max(l_p, l_c)$, $v \in V$, and $u \in V \setminus B$. According to Lemma 8, this step can be carried out in time $O^*(6.855^t)$.

Part 3: Answering the Instance. Finally, we determine whether the instance is YES by inspecting the entries $\hat{\mathcal{F}}^{vu}_{k,\ell,p}$. If there exists a tuple (k,ℓ,p,v,u) such that $\hat{\mathcal{F}}^{vu}_{k,\ell,p}$ is nonempty and at least one of the two conditions in Lemma 7 holds, we return YES; otherwise, we return NO. This check can be performed within the same time bounds as Part 2.

Similar to KEPALG, the algorithm KEPALG-REP can be extended to output a corresponding feasible path-cycle packing by maintaining an additional data structure during the computation. □

4 Conclusion and Discussion

In this paper, we study KEP parameterized by the number of covered recipients t and present a deterministic algorithm with running time $O^*(6.855^t)$. To achieve this result, we introduce the technique of representative sets into KEP. With modifications to the dynamic programming framework, our approach can be extended to various KEP variants [Krivelevich *et al.*, 2007; Biró *et al.*, 2009] as well as to other path- and cycle-related packing problems.

Several promising directions remain for further improving our algorithm. At present, our procedure computes vertex subsets for all semi-feasible path-cycle packings of length at most $2t$, as described in Lemmas 3 and 4. It may be possible to optimize this step by focusing solely on vertex subsets of semi-feasible packings of length at most t and then determining in polynomial time whether these can be extended to a full feasible path-cycle packing.

In the realm of randomized algorithms, the best known running time is $O^*(4^t)$. By applying Lemma 3, one can show that the randomized algorithm of Maiti and Dey [Maiti and Dey, 2022] also achieves a running time of $O^*(4^t)$. Whether this bound can be further improved remains an open and challenging problem.

Another intriguing avenue for future work is the development of even faster algorithms for KEP when l_p and l_c are small constants. This consideration is particularly significant in practical applications, where these parameters are often naturally limited in size.

Acknowledgments

We thank the anonymous reviewers for their valuable comments and suggestions that helped improve the quality of this paper. The work is supported by the National Natural Science Foundation of China, under the grants 62372095 and 62502078.

References

[Abraham *et al.*, 2007] David J. Abraham, Avrim Blum, and Tuomas Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In Jeffrey K. MacKie-Mason, David C. Parkes, and Paul Resnick, editors, *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, 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*, 112:663 – 668, 2015.

[Banik *et al.*, 2025] Aritra Banik, Sujoy Bhore, Palash Dey, and Abhishek Sahu. Kidney exchange: Faster parameterized algorithms and tighter lower bounds, 2025.

[Biró *et al.*, 2009] Péter Biró, David F. Manlove, and Romeo Rizzi. Maximum weight cycle packing in directed graphs, with application to kidney exchange programs. *Discret. Math. Algorithms Appl.*, 1(4):499–518, 2009.

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

[Dickerson *et al.*, 2016] John P. Dickerson, David F. Manlove, Benjamin Plaut, Tuomas Sandholm, and James Trimble. Position-indexed formulations for kidney exchange. In Vincent Conitzer, Dirk Bergemann, and Yiling Chen, editors, *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 25–42. ACM, 2016.

[Fomin *et al.*, 2016] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016.

[Francis *et al.*, 2024] Anna Francis, Meera N Harhay, Albert Ong, Sri Lekha Tummalapalli, Alberto Ortiz, Agnes B Fogo, Danilo Fliser, Prabir Roy-Chaudhury, Monica Fontana, Masaomi Nangaku, et al. Chronic kidney disease and the global public health agenda: an international consensus. *Nature Reviews Nephrology*, pages 1–13, 2024.

[Glorie *et al.*, 2014] Kristiaan M. Glorie, Joris van de Klundert, and Albert P. M. Wagelmans. Kidney exchange with long chains: An efficient pricing algorithm for clearing barter exchanges with branch-and-price. *Manuf. Serv. Oper. Manag.*, 16(4):498–512, 2014.

[Hébert-Johnson *et al.*, 2024] Úrsula Hébert-Johnson, Daniel Lokshtanov, Chinmay Sonar, and Vaishali Surianarayanan. Parameterized complexity of kidney exchange revisited. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 76–84. ijcai.org, 2024.

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

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

[Lin *et al.*, 2019] Mugang Lin, Jianxin Wang, Qilong Feng, and Bin Fu. Randomized parameterized algorithms for the kidney exchange problem. *Algorithms*, 12(2):50, 2019.

[Maiti and Dey, 2022] Arnab Maiti and Palash Dey. Parameterized algorithms for kidney exchange. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 405–411. ijcai.org, 2022.

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

[Pahwa *et al.*, 2012] Mrinal Pahwa, Yusuf Saifee, Vipin Tyagi, Sudhir Chadha, and Harsh Jauhari. Paired exchange kidney donation in india: A five-year single-center experience. *International urology and nephrology*, 44:1101–1105, 2012.

[Pimenta *et al.*, 2023] Pedro Foletto Pimenta, Pedro H. C. Avelar, and Luís C. Lamb. Graph neural networks with no supervision and heuristics for the kidney-exchange problem. In *35th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2023, Atlanta, GA, USA, November 6-8, 2023*, pages 258–262. IEEE, 2023.

[Rapaport, 1986] Felix T. Rapaport. The case for a living emotionally related international kidney donor exchange registry. *Transplantation proceedings*, 18 3) Suppl. 2:5–9, 1986.

[Tian and Xiao, 2026] Kangyi Tian and Mingyu Xiao. A faster deterministic algorithm for kidney exchange via representative set, 2026.

[Williams, 2009] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

[Xiao and Wang, 2018] Mingyu Xiao and Xuanbei Wang. Exact algorithms and complexity of kidney exchange. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence,*

IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, pages
555–561. ijcai.org, 2018.