KidneyExchange.jl: A Julia package for solving the kidney exchange problem with branch-and-price

Ayse N Arslan, Jérémy Omer, Fulin Yan

KidneyExchange.jl: A Julia package for solving the kidney exchange problem with branch-and-price

Ayşe N. Arslan
Centre Inria de l'Universite de Bordeaux, F-33405 Talence, France
Univ. Bordeaux, CNRS, INRIA, Bordeaux INP, IMB, UMR 5251, F-33400 Talence, France
ayse-nur.arslan@inria.fr

Jérémy Omer
Univ Rennes, INSA Rennes, CNRS, IRMAR - UMR 6625, 35000 Rennes, France
jeremy.omer@insa-rennes.fr

Fulin Yan
Univ Rennes, INSA Rennes, CNRS, IRMAR - UMR 6625, 35000 Rennes, France
fulin.yan@insa-rennes.fr

October 26, 2022

Abstract

The *kidney exchange problem* (KEP) is an increasingly important healthcare management problem in most European and North American countries which consists of matching incompatible patient-donor pairs in a centralized system. Despite the significant progress in the exact solution of KEP instances in recent years, larger instances still pose a challenge especially when altruistic donors are taken into account. In this article, we present a branch-and-price algorithm for the exact solution of KEP in the presence of altruistic donors. This algorithm is based on a disaggregated cycle and chains formulation where subproblems are managed through graph copies. We additionally, present a branch-and-price algorithm based on the position-indexed chain-edge formulation, as well as two compact formulations. We formalize and analyze the complexity of the resulting pricing problems and identify the conditions under which they can be solved using polynomial-time algorithms. We propose several algorithmic improvements for the branch-and-price algorithms as well as for the pricing problems. We extensively test all of our implementations using a benchmark made up of different types of instances with the largest instances considered having 10000 pairs and 1000 altruists. Our numerical results show that the proposed algorithm can be up to two orders of magnitude faster compared to the state-of-the-art. All models and algorithms presented in the paper are gathered in an open-access Julia package, KidneyExchange.jl.

1 Introduction

The *kidney exchange problem* (KEP) is an increasingly important healthcare management problem in most European and North American countries. A kidney transplant is the preferred treatment for many patients that are suffering from a chronic kidney disease. Although transplants are traditionally performed from a deceased donor, in many countries, the patients are allowed to find a living donor, typically a relative or a friend. Unfortunately, a patient may be incompatible with the person that they designate as their donor. The kidney exchange problem arises from this

context: incompatible patient-donor pairs are put in a centralized system, called a *kidney exchange program*, with a view to find matches between the donors and patients of different pairs.

Transplants in a kidney exchange can be performed in a cyclic manner with each patient receiving a transplant from the donor of the previous pair. In an increasing number of countries, the law also allows the participation of *altruistic donors*, individuals who are willing to donate a kidney without being attached to a patient. Each patient-donor pair and each altruist can participate in at most one exchange. One of the particularities of the kidney exchange problem is that the number of patient-donor pairs involved in an exchange is limited. Indeed, the donor of a pair may change their mind once the patient receives a donation, jeopardizing the transplants of patients down the line. In order to avoid this complicated situation, transplant operations in an exchange are typically performed simultaneously. Since, transplant operations require multiple operating rooms and an important number of medical staff, this practically restricts the number of pairs that can be in a cyclic exchange to 3 or 4, which is the current state of practice in many countries. This restriction is less strict in the presence of altruistic donors but is still present in practice, with exchanges involving altruists limited to 5 or 6 patients.

The operations research community has long been interested in the optimal solution of the kidney exchange problem and its variants, which led to the development of various mathematical programming formulations and associated solution algorithms. In this work, we concentrate on the optimal solution of the deterministic kidney exchange problem in the existence of altruistic donors. We implement two branch-and-price algorithms based on two different extended formulations previously proposed in the literature (Riascos-Álvarez et al. (2020),John P. Dickerson, Manlove, et al. (2016)) where subproblems are managed through graph copies. We analyze the resulting pricing problems in detail and identify conditions under which they can be solved through polynomial-time algorithms as well as propose numerical improvements to their solution. We propose several improvements to the branch-and-price algorithm specific to the kidney exchange problem. We also adapt existing compact formulations from the literature for use with feedback vertex sets (FVS) proposed by Riascos-Álvarez et al. (2020). Our work results in an open source Julia package that integrates state-of-the-art methodology and formulations from the literature, regrouping them on the same easy-to-deploy platform for the first time. Our numerical results show that our implementation of the branch-and-price algorithm can be up to two orders of magnitude faster than those proposed in the literature.

The remainder of this article is organized as follows. In Section 2, we formalize the kidney exchange problem and present a literature review. In Section 3, we present and adapt the existing compact mixed-integer programming formulations. In Section 4, we present the branch-and-price algorithm and provide implementation details. In Section 5, we briefly describe the benchmark and our open source Julia package, and we present our numerical results. Conclusions and future research directions are presented in Section 6.

2 Problem description and literature review

2.1 Preliminary notations

We first present a few general notations that will be used throughout the article, starting with the problem definition. A (directed) *walk* p in a directed graph $G = (V, A)$ is a subgraph of G consisting of a sequence of nodes and arcs $i_1, a_1, i_2, \ldots, a_{r-1}, i_r$ such that for all $1 \leq k \leq r-1$, $a_k = (i_k, i_{k+1}) \in A$. Given that we will consider only simple graphs, a walk will be more concisely denoted as $p = (i_1, \ldots, i_r)$ without ambiguity. If p is such that $i_1 = i_r$, then it is a *closed walk*. A *chain* is a walk without any repetition of nodes. Although such a walk is more classically referred to as a path, a simple path or an elementary path in the graph theory and network-flow literature (see e.g. Ahuja et al. 1988; Taccari 2016), we use the term *chain* to remain coherent with the

kidney exchange literature. A *cycle* is then a chain such that $i_1 = i_r$, *i.e.*, a closed walk without any repetition of nodes other than the first one. A walk p is neither a chain nor a cycle if and only if it contains a *subtour*, *i.e.*, a cycle that is strictly contained in p.

We will also denote as $[\![a\,;b]\!]$ the integer interval $\{a, a+1, \ldots, b\}$ for any $a, b \in \mathbb{Z}, a \leq b$.

2.2 Problem statement and definitions

An instance of the KEP is characterized by a simple arc-weighted directed graph $G = (V, A)$, and two integers K and L. The set of vertices V is partitioned into $P \cup D$, where P represents the patient-donor pairs and D represents the altruistic donors. Each arc $(u, v) \in A$, $u \in V$, $v \in P$, means that the donor in u and the recipient in v are compatible for a transplant. We denote as w_a the utility of a transplant represented by arc $a \in A$. It is very common to consider the total number of transplants in the kidney exchange literature, *i.e.*, $w_a = 1$ for $a \in A$. The problem then consists in finding disjoint cycles of length at most K and chains starting from altruistic donors of length at most L in order to maximize the total utility of the arcs covered by these cycles and chains.

For a more concise presentation, any cycle (resp. chain) with length at most K (resp. at most L) will be called a *feasible cycle* (resp. a *feasible chain*). We denote the set of all feasible cycles and chains by $\mathcal{C}_K$ and $\mathcal{C}_L$, respectively. For $c \in \mathcal{C}_K$ (resp.$c \in \mathcal{C}_L$) we denote by $V(c)$ the set of vertices, and $A(c)$ the set of arcs involved in c. Finally, for any vertex $v \in V$, $\mathcal{C}_K(v)$ (resp. $\mathcal{C}_L(v)$) is the set of cycles in $\mathcal{C}_K$ (resp. the set of chains in $\mathcal{C}_L$) containing v. We remark that for $v \in D$ the set $\mathcal{C}_K(v)$ is empty.

2.3 Literature review

Due to the arbitrary bounds on the lengths of cycles and chains, the kidney exchange problem is NP-Hard (Abraham et al. (2007)). Despite this discouraging complexity result, the operations research community has long been interested in the optimal solution of this problem, proposing various compact and extended formulations for the cycles-only and cycles-and-chains variants of the problem. Compact formulations can be solved directly using an off-the-shelf optimization solver, whereas extended formulations require either cut- or column-generation algorithms in their solution. Below, we present a literature review for the cycles-only and cycles-and-chains variants of the problem, independently. For the sake of conciseness, we do not present the considerable body of literature dealing with variants of the problem under uncertainty (see, for instance, John P. Dickerson, Procaccia, et al. (2019)).

Abraham et al. (2007) propose two extended formulations for the cycles-only variant of the problem: edge and cycle formulation. The edge formulation uses flow variables to form cycles and imposes the size limitation with an exponential number of constraints, while the cycle formulation uses an exponential number of cycle selection variables (one variable corresponding to each element of $\mathcal{C}_K$). The first requires cut-generation in its solution whereas the second requires column-generation. Authors report being able to solve instances involving up to 10,000 nodes to optimality using the branch-and-price algorithm with the cycle formulation. They solve the pricing problem using a combination of depth-first-search and heuristic algorithms. In Constantino et al. (2013) two compact mixed-integer programming formulations are proposed. One of these formulations, the extended-edge (EE) formulation, is based on the idea of creating one copy of the graph for each node and constructing cycles involving this node in each copy through the use of flow variables. Later, in Klimentova et al. (2014), Dantzig-Wolfe decomposition is applied to this formulation, which results in a branch-and-price algorithm with many pricing problems (one per node). These pricing problems permit searching for exchange cycles involving each node independently. In Riascos-Álvarez et al. 2020, this idea is further extended, using the concept of feedback vertex sets (FVS). The authors create a copy only for the nodes in an FVS instead of all nodes. To solve the resulting pricing problems, Klimentova et al. (2014) apply a combination of a random-walk

heuristic and Ford's shortest path algorithm, whereas Riascos-Álvarez et al. 2020 employ a combination of decision diagrams, depth-first search and mixed-integer programming. Lam and Mak-Hau (2020) propose to combine the branch-and-price algorithm of Klimentova et al. (2014) with valid inequalities, giving rise to a branch-and-price-and-cut algorithm. They solve the pricing problems by solving elementary longest path problems on a position-expanded directed acyclic graph through a label-setting algorithm when $K = 3$, and by exhaustive enumeration otherwise.

Similar to the cycles-only variant, when the possibility of constructing chains is introduced, two main approaches emerge in the literature: using flow variables to form chains starting from altruistic donors or using an exponential number of chain selection variables. In the flow-based models, cycle-flow and chain-flow variables are then distinguished. However, when $L \geq K + 1$, in the absence of cycle-breaking constraints, it would be possible to construct infeasible cycles (of length $\geq K+1$) using the chain-flow variables. As such, flow-based formulations for the cycles-and-chains variant have been inspired by different modeling techniques used for the traveling salesman and elementary shortest path problems where sub-tour elimination constraints are necessary. In the chain selection-based models, this difficulty emerges in the pricing problem.

In Anderson et al. (2015), the authors build on the formulations of Abraham et al. (2007) to take the existence of unlimited chains into account, inspired by the prize-collecting traveling salesman problem. They propose to solve the resulting formulations using cut generation algorithms. In Mak-Hau (2017) authors propose to adapt the extended-edge formulation of Constantino et al. (2013) to the case of unlimited chains inspired by the Miller-Tucker-Zemlin inequalities for the traveling salesman problem. Their formulation can also be adapted to the case where the chain size is limited. John P. Dickerson, Manlove, et al. (2016) propose compact formulations by adding a position index to variables of the extended-edge formulation. The resulting formulation, called the position-indexed edge formulation (PIEF), gives a stronger relaxation than the extended-edge formulation for the cycles-only variant. They call the resulting formulation the hybrid position-indexed edge formulation (HPIEF) for the cycles-and-chains variant. They additionally combine position-indexed variables for chains with cycle selection variables, giving rise to what authors call the position-indexed chain-edge formulation (PICEF). We remark that PICEF can be used as a compact formulation by enumerating the feasible cycles in a pre-processing step, otherwise it needs to be solved using a branch-and-price algorithm generating cycles through the solution of pricing problems. Additionally, this formulation reduces to the cycle formulation of Abraham et al. 2007 when $L = 0$. Finally, Glorie et al. (2014), Plaut et al. (2016a), and Riascos-Álvarez et al. (2020) propose branch-and-price algorithms that generate both cycles and chains through the solution of pricing problems. We remark that algorithms used in the solution of the pricing problems in both Glorie et al. (2014) and Plaut et al. (2016a) were later shown to be incorrect, *i.e.*, these algorithms may falsely conclude that there are no columns to be added to the master problem. As such, the work of Riascos-Álvarez et al. (2020) is the only correct branch-and-price algorithm in the literature for the cycles-and-chains variant. The authors propose a three-phase approach to the solution of pricing problems: a (restricted) decision diagram-based approach, followed by a mixed-integer programming model solved through cut generation, and finally a depth-first search heuristic, followed by the direct solution of a mixed-integer program. In the first phase, they create one decision diagram per altruistic donor and per node of an FVS. The former generates chains starting from each altruistic donor, whereas the latter generates cycles starting from each element of the FVS. In the second phase, they solve a relaxed longest-path formulation, which is augmented by cutting planes when necessary. This phase eventually proves that no positive reduced cost chain exists. Finally, in the third phase, positive reduced cost cycles missed in the first two phases are sought, first by depth-first search with a time-limit, and finally by the solution of a flow-based formulation limiting the number of arcs that can be selected to K.

While there exists an important body of literature on the exact solution of the KEP, especially its cycles-only variant, the cycles-and-chains variant of the problem is still under investigation with

the work of Riascos-Álvarez et al. (2020) being the only correct exact algorithm to date. The confusion surrounding the solution of subproblems in the existence of altruistic donors, and the associated theoretical and practical difficulty, needs to be clarified. Further, the numerical results in the literature focus mostly on instances from a single type of instance generator (known as the Saidman generator) and the developed algorithms are often inaccessible to researchers. In this work, we address most of these shortcomings. Our contributions include:

- Implementing two branch-and-price algorithms that can solve the cycles-and-chains variant of the kidney exchange problem to exact optimality. Both our implementations gather numerous improvements tailored for the KEP which are detailed in the article. One of these algorithms applies to the position-indexed chain edge formulation (PICEF) of John P. Dickerson, Manlove, et al. (2016) and the other is based on the disaggregated cycles and chains formulation DCF of Riascos-Álvarez et al. (2020). To the best of our knowledge, these are the first publicly available implementations of the column generation algorithm for the KEP.

- Integrating the notion of FVS to existing compact formulations, *i.e.*, the extended-edge (EE) formulation (Constantino et al. (2013)) and the position-indexed edge formulation (PIEF) (John P. Dickerson, Manlove, et al. (2016)) for the cycles-only variant. We then combine these two formulations with position-indexed chain flow variables in the presence of altruistic donors in order to obtain two compact formulations for the cycles-and-chains variant of the problem.

- Gathering state-of-the-art formulations, algorithms, and instance generation schemes on the same easy-to-deploy platform for the first time. Our open source code can be used with different solvers, and facilitates development and numerical testing of new formulations/algorithms.

- Assessing the numerical performance of the two branch-and-price algorithms, as well as the compact formulations on a large benchmark with instances coming from different generators (including the Saidman generator). The largest instances tested include 10,000 pairs and 1,000 altruists. This is, to the best of our knowledge, the first time such large instances are solved in the literature for the cycles-and-chains variant (the seminal work of Abraham et al. (2007) considered the cycles-only variant with $K = 3$), and the first publicly available implementation to do so.

3 Compact integer programming formulations

In this section, we first adapt the extended-edge (Constantino et al. 2013) and position-indexed edge (John P. Dickerson, Manlove, et al. 2016) formulations for the cycles-only variant to the case where graph copies are created based on a feedback vertex set (FVS). An FVS is a set of vertices, that we denote by $S \subseteq P$ in the following, whose removal from G leads to an acyclic graph. Any cycle in G passes through at least one vertex $s \in S$, and it therefore suffices to consider only the elements of an FVS in creating the graph copies. We remark that the cycles-only formulations we present here reduce to those presented in the literature when $S = P$.

To calculate an FVS, we start by removing the vertices with in-degree or out-degree equal to zero from consideration. We then consider the remaining vertices in P in order of decreasing total degree. When considering a vertex s, we first add it in S and then we create a graph copy $G^s = (V^s, A^s)$ with nodes V^s and arcs A^s. To determine V^s, we use the pairwise distances $d(s, v)$ and $d(v, s)$ for $v \in P$. If $d(s, v) + d(v, s) > K$, vertex v cannot be part of a cycle with length at most K including s, so we do not include v in V^s. The set of arcs, A^s, is then equal to $(V^s \times V^s) \cap A$. This set can further be reduced using the pairwise distances. More specifically, if $d(s, i) + d(j, s) + 1 > K$, then arc (i, j) can be removed from A^s. Once a vertex is considered in order, we reduce the in-degree of each of its forward neighbors as well as the out-degree of each of its backward neighbors and we

remove it from G along with its incoming and outgoing arcs. The process continues until there are no vertices with positive in- and out-degree.

Let $x^s_{u,v}$ take value 1 if arc $(u,v) \in A^s$ is selected in graph copy $s \in S$ and 0 otherwise. The extended-edge formulation based on FVS, S, is then written as:

$$\max \quad \sum_{s \in S} \sum_{(u,v) \in A^s} w_{u,v} x^s_{u,v} \tag{1}$$

$$\text{s.t.} \quad \sum_{s \in S} \sum_{v|(u,v) \in A^s} x^s_{u,v} \leq 1 \qquad \forall u \in P \tag{2}$$

$$\sum_{v|(v,u) \in A^s} x^s_{v,u} = \sum_{v|(u,v) \in A^s} x^s_{u,v} \qquad \forall s \in S, u \in V^s \tag{3}$$

$$\sum_{(u,v) \in A^s} x^s_{u,v} \leq K \qquad \forall s \in S \tag{4}$$

$$\sum_{v|(u,v) \in A^s} x^s_{u,v} \leq \sum_{v|(s,v) \in A^s} x^s_{s,v} \qquad \forall s \in S, u \in V^s \tag{5}$$

$$x^s_{u,v} \in \{0,1\} \qquad \forall s \in S, (u,v) \in A^s. \tag{6}$$

Here, constraints (2) impose that each node $u \in P$ can be part of at most one cycle across all copies $s \in S$, (3) are flow-balance constraints and ensure that the flow in each copy $s \in S$ forms a cycle, (4) impose the cycle-size limit K and ensure that only feasible cycles are formed, and finally (5) break symmetries by generating cycles starting from s in graph copy $s \in S$. The objective function maximizes the total utility.

Similarly, let $x^s_{u,v,l}$ take value 1 if arc $(u,v) \in A^s$ is selected in position $l = 1, \ldots, K$ in copy $s \in S$ and 0 otherwise. The position-indexed formulation based on FVS, S, is then written as:

$$\max \quad \sum_{s \in S} \sum_{(u,v) \in A^s} \sum_{l=1}^{K} w_{u,v} x^s_{u,v,l} \tag{7}$$

$$\text{s.t.} \quad \sum_{s \in S} \sum_{v|(u,v) \in A^s} \sum_{l=1}^{K} x^s_{u,v,l} \leq 1 \qquad \forall u \in P \tag{8}$$

$$\sum_{v|(v,u) \in A^s} x^s_{v,u,l} = \sum_{v|(u,v) \in A^s} x^s_{u,v,(l+1)} \qquad s \in S, \forall u \in V^s \setminus s, l = 1, \ldots, K-1 \tag{9}$$

$$\sum_{l=2}^{K} \sum_{v|(v,s) \in A^s} x^s_{v,s,l} = \sum_{v|(s,v) \in A^s} x^s_{s,v,1} \qquad s \in S \tag{10}$$

$$x^s_{u,v,l} \in \{0,1\} \qquad \forall s \in S, (u,v) \in A^s, l = 1, \ldots, K. \tag{11}$$

Here, constraints (8) impose that each node $u \in P$ can be part of at most one cycle across all copies $s \in S$, and (9) and (10) are flow-balance constraints and ensure that cycles are formed starting from s in graph copy $s \in S$. The objective function maximizes the total utility.

In order to form chains, we use the position-indexed chain-flow variables $y_{u,v,l}$ that take value 1 if arc $(u,v) \in A$ is selected in position $l = 1, \ldots, L$ of a chain and 0 otherwise. We write:

$$\sum_{l=1}^{L} \sum_{v|(u,v) \in A} y_{u,v,l} \leq 1 \qquad \forall u \in V \tag{12}$$

$$\sum_{v|(v,u) \in A} y_{v,u,l} \geq \sum_{v|(u,v) \in A} y_{u,v,(l+1)} \qquad \forall u \in P, l = 1, \ldots, L-1 \tag{13}$$

$$\sum_{v|(u,v)\in A} y_{u,v,1} = 0 \qquad \forall u \in P \tag{14}$$

$$\sum_{v|(u,v)\in A} y_{u,v,l} = 0 \qquad \forall u \in D, l = 2, \ldots, L \tag{15}$$

$$y_{u,v,l} \in \{0,1\} \qquad \forall (u,v) \in A, l = 1, \ldots, L. \tag{16}$$

Here, constraints (12) impose that each node is involved in at most one chain, (13) are flow balance constraints and ensure that a chain is formed, (14)-(15) impose that only altruistic donors can initiate a chain and therefore appear in position $l = 1$ and only in position $l = 1$.

We then combine (12)-(16) with the FVS-based cycles-only formulations presented above. The first compact formulation we obtain is novel, to the best of our knowledge, and is written as:

$$\max \quad \sum_{s\in S} \sum_{(u,v)\in A^s} w_{u,v} x^s_{u,v} + \sum_{(u,v)\in A} \sum_{l=1}^{L} w_{u,v} y_{u,v,l} \tag{17}$$

$$\text{s.t.} \quad (3)-(6), (13)-(16) \tag{18}$$

$$\sum_{s\in S} \sum_{v|(u,v)\in A^s} x^s_{u,v} + \sum_{l=1}^{L} \sum_{v|(u,v)\in A} y_{u,v,l} \leq 1 \qquad \forall u \in V. \tag{19}$$

The second compact formulation we obtain is an adaptation of the HPIEF to the case where the graph copies are generated from an FVS. This formulation is written as:

$$\max \quad \sum_{s\in S} \sum_{(u,v)\in A^s} \sum_{l=1}^{K} w_{u,v} x^s_{u,v,l} + \sum_{(u,v)\in A} \sum_{l=1}^{L} w_{u,v} y_{u,v,l} \tag{20}$$

$$\text{s.t.} \quad (9)-(11), (13)-(16) \tag{21}$$

$$\sum_{s\in S} \sum_{v|(u,v)\in A^s} \sum_{l=1}^{K} x^s_{u,v,l} + \sum_{l=1}^{L} \sum_{v|(u,v)\in A} y_{u,v,l} \leq 1 \qquad \forall u \in V. \tag{22}$$

In order to further reduce symmetries in the case of integer weights, the objective function of the two models can be modified by multiplying variables $x^s_{u,v}$ and $x^s_{u,v,l}$ by $(1+\ell(s)/(w^{\max} * |P|^2))$ where $w^{\max} = \max_{(u,v)\in A} w_{u,v}$ and $\ell(s)$ is the index (position) of s in S. This modification prioritizes the use of arc (u,v) in forming cycles rather than chains. Between cycles that can be obtained in multiple graph copies, those that are obtained from a larger-indexed copy are preferred.

It is additionally worth noting that (12)-(16) can be combined with any existing cycle-based formulation, making sure that every node is involved in at most one cycle or chain, in order to obtain a valid formulation for the cycles-and-chains variant of the problem. For instance, the PICEF is obtained by combining (12)-(16) with the cycle formulation (see Appendix).

4 Branch-and-price algorithm

In this section, we provide the most important details behind our implementation of the branch-and-price algorithm. We consider the disaggregated cycle and chain formulation DCF proposed by Riascos-Álvarez et al. (2020) as well as the position-indexed chain-edge formulation (PICEF) proposed by John P. Dickerson, Manlove, et al. (2016). In both formulations, graph copies $\{G^s = (V^s, A^s)\}$ for $s \in S$, of G, based on an FVS, are used to decompose the pricing problem. These copies are obtained as described in Section 3. We additionally create a graph copy G^s for each altruistic donor $s \in D$ when using DCF in a similar manner. That is, starting from each altruistic

donor $s \in D$, we calculate the distance $d(s,v)$ for $v \in P$, and we include only those vertices with $d(s,v) \leq L$ in V^s. The set of arcs A^s is then constituted as $(V^s \times V^s) \cap A$.

Denoting as $\mathcal{C}_K^s$ (resp. $\mathcal{C}_L^s$) the set of feasible cycles (resp. chains) in G^s for $s \in S$ (resp. $s \in D$), $\mathcal{C}_K^s(v)$ (resp. $\mathcal{C}_L^s(v)$) the set of cycles in $\mathcal{C}_K^s$ (resp. chains in $\mathcal{C}_K^s$) including node $v \in V^s$ and $w(c)$ the utility of cycle $c \in \mathcal{C}_K^s$ (resp. chain $c \in \mathcal{C}_L^s$), DCF can be formulated as follows:

$$(\text{DCF}): \begin{cases} \max & \displaystyle\sum_{s\in S}\sum_{c\in\mathcal{C}_K^s} w(c)x_c + \sum_{s\in D}\sum_{c\in\mathcal{C}_L^s} w(c)x_c & \\ \text{s.t.} & \displaystyle\sum_{s\in S}\sum_{c\in\mathcal{C}_K^s(v)} x_c + \sum_{s\in D}\sum_{c\in\mathcal{C}_L^s(v)} x_c \leq 1 & \forall v \in V \\ & x_c \in \{0,1\} & \forall c \in (\cup_{s\in S}\mathcal{C}_K^s) \cup (\cup_{s\in D}\mathcal{C}_L^s). \end{cases} \tag{23}$$

The linear relaxation of DCF, DCF^{LP}, is obtained by dropping the integrality restrictions on variables x_c. We use a column generation algorithm to solve DCF^{LP}. Accordingly, at iteration $i \in \mathbb{Z}_+$ of the algorithm, we consider $\hat{\mathcal{C}}_K^{s,i} \subseteq \mathcal{C}_K^s$ for $s \in S$ and $\hat{\mathcal{C}}_L^{s,i} \subseteq \mathcal{C}_L^s$ for $s \in D$. In replacing $\mathcal{C}_K^s$ and $\mathcal{C}_L^s$ by $\hat{\mathcal{C}}_K^{s,i}$ and $\hat{\mathcal{C}}_L^{s,i}$, respectively, we obtain a restriction of DCF^{LP}, that we denote as RMP. Let the optimal solution of RMP at iteration i be denoted by $\hat{x}^i$, and let λ^i denote a vector of optimal dual variables associated to constraints (23), with λ_v^i denoting the dual variable associated to vertex $v \in V$. The reduced cost of a cycle (resp. chain) $c \in \mathcal{C}_K^s$ (resp. $c \in \mathcal{C}_L^s$) is given as $w(c) - \sum_{v\in V(c)} \lambda_v$. The pricing problem for graph copy G^s for $s \in S \cup D$ is then the decision problem that either identifies a positive reduced cost cycle or chain in G^s or concludes that none exists. We present our approach to the solution of these pricing problems in detail in Section 4.1.

If any positive reduced cost cycle or chain is detected through the solution of these pricing problems then $\hat{\mathcal{C}}_K^{s,i}$ and $\hat{\mathcal{C}}_L^{s,i}$ are augmented with these new columns. Otherwise, $\hat{x}^i$ is an optimal solution of RMP which can be extended to an optimal solution of DCF^{LP} by setting $x_c = \hat{x}_c^i$ for $c \in \hat{\mathcal{C}}_K^{s,i}$ and $c \in \hat{\mathcal{C}}_L^{s,i}$ and $x_c = 0$ for $c \in \mathcal{C}_K^s \setminus \hat{\mathcal{C}}_K^{s,i}$ and $c \in \mathcal{C}_L^s \setminus \hat{\mathcal{C}}_L^{s,i}$. We remark that, as there are $|S| + |D|$ pricing problems, an implementation choice needs to be made in terms of whether or not to solve all subproblems at each iteration as well as in which order to explore the subproblems. We present our approach to managing this algorithmic choice in detail in Sections 4.3 and 4.3.2. Let z^{LP} be the optimal value of DCF^{LP} and z^{lb} the objective value of the best integer solution found. If $z^{\text{lb}} = z^{\text{LP}}$ (which is true, for instance, if the optimal solution of DCF^{LP} is integer), then DCF is solved to optimality. Otherwise, it is necessary to branch. Following the conclusions of Riascos-Álvarez et al. (2020), we chose to branch on arcs in A instead of branching on the cycles and chains or on the arcs of the graph copies $G^s, s \in S \cup D$. As such, we identify an arc $(u,v) \in A$ such that

$$0 < \hat{f}_{u,v} := \sum_{s\in S}\sum_{c\in\mathcal{C}_K^s|(u,v)\in A^s(c)} \hat{x}_c + \sum_{s\in D}\sum_{c\in\mathcal{C}_L^s|(u,v)\in A^s(c)} \hat{x}_c < 1.$$

We branch on the arc (u,v) such that $\hat{f}_{u,v}$ is closest to 0.5. Branching is performed by adding the following constraint in DCF.

$$\sum_{s\in S}\sum_{c\in\mathcal{C}_K^s|(u,v)\in A^s(c)} x_c + \sum_{s\in D}\sum_{c\in\mathcal{C}_L^s|(u,v)\in A^s(c)} x_c \leq 0 \qquad (\geq 1) \tag{24}$$

Denoting μ, the dual variable associated to this branching constraint, the effect on the pricing problem associated to $s \in S \cup D$ is the addition of the term $\mu\ \mathbb{I}_c^s(u,v)$ in the reduced cost of a cycle (resp. chain), where $\mathbb{I}_c^s(u,v)$ is an indicator function that takes value 1 if $(u,v) \in A^s(c)$, and 0 otherwise for given $c \in \mathcal{C}_K^s \cup \mathcal{C}_L^s$. In Section 4.1, pricing problems are presented by taking the effect of these branching constraints into account.

In the case of PICEF, the branch-and-price algorithm changes slightly since only cycles are generated through the solution of subproblems. In particular, all cycle selection variables x_c for

$s \in S, c \in \mathcal{C}_K^s$ may be integer in a fractional solution. In this case, there is necessarily $(u,v) \in A$ and $l \in [\![1\,;L]\!]$ such that $0 < y_{u,v,l} < 1$. We then branch by imposing that either arc (u,v) is part of a solution, $\sum_{l=1}^{L} y_{u,v,l} \geq 1$, or not, $\sum_{l=1}^{L} y_{u,v,l} \leq 0$. This branching choice does not affect the reduced cost of cycles.

4.1 The pricing problem

In the remainder of the section, we will call a feasible cycle or chain with positive reduced cost a *positive cycle* or a *positive chain* unless stated otherwise.

The purpose of the pricing problem is to find a positive cycle or chain or prove that none exists. It has been shown (see for instance Glorie et al. 2014; Plaut et al. 2016a) that finding a positive cycle or showing that none exists can be done in polynomial time. The proof relies on the study of a dynamic programming algorithm similar to the Bellman-Ford algorithm. In the same articles, the authors have adapted this algorithm to the search for positive chains, but it has later been demonstrated by Plaut et al. (2016b) that the resulting algorithms are incorrect. Indeed, for $K = 0$ and $L = |V|$, finding a positive chain starting from an altruistic donor or proving that none exists amounts to solving an *elementary longest path problem*, which is known to be an NP-hard problem in the presence of positive cost cycles (see e.g. Taccari 2016). However, in the kidney exchange problem, K and L are not arbitrary, with the current practice typically limiting K to less than 4 and L to less than 6. One consequence of this is that adaptations of the Bellman-Ford algorithm are still able to solve the pricing problem in many cases.

At each iteration of the column generation algorithm, we search for a positive cycle in each copy $G^s, s \in S$, and a positive chain in each copy $G^s, s \in D$. By construction of the copies, there is no positive cycle or chain in G if none is found in $G^s, \forall s \in S \cup D$. In this search, the reduced cost of a chain or cycle c is denoted by $\bar{w}(c)$. As discussed in Glorie et al. 2014, $\bar{w}(c)$ can be expressed as the sum of weights on the arcs $\bar{w}_{u,v}, \forall (u,v) \in A(c)$ if branching is performed as described in (24). For instance, at the root node, we have

$$\bar{w}(c) = \sum_{(u,v)\in A(c)} w_{u,v} - \sum_{v \in V(c)} \lambda_v$$

regardless of whether c is a cycle or chain. Setting $\bar{w}_{u,v} = w_{u,v} - \lambda_v, \forall (u,v) \in A$, we obtain $\bar{w}(c) = \sum_{(u,v)\in A(c)} \bar{w}_{u,v}$ for any cycle c. If c is a chain, the dual cost of the initial altruistic node must be added to the reduced cost, so we may set $\bar{w}_{u,v} = w_{u,v} - \lambda_u - \lambda_v$ for all $(u,v) \in A$ such that $u \in D$. At any other node i of the branching tree, we let $\overline{A}^i$ be the set of arcs on which branching was performed. We then write $\bar{w}_{u,v} = w_{u,v} - \lambda_v - \mu_{u,v}$, for $(u,v) \in \overline{A}^i$ where $\mu_{u,v}$ is the dual variable associated to the branching constraint (u,v). The reduced cost of arcs $(u,v) \in A \setminus \overline{A}^i$ remain unchanged.

In the remainder, we first show that a polynomial-time dynamic programming algorithm can be used to identify positive columns in a large variety of cases, even when chains are allowed. When no positive chain or cycle is found by this polynomial algorithm, we state conditions under which no such cycle or chain exists and we describe an alternative procedure when these conditions do not hold. Given that most of the branch-and-price solution time is spent in searching for positive columns, we also provide some details on the algorithmic improvements that lead to significant reduction in computational time.

4.1.1 Dynamic programming search for positive cycles

The search for a positive cycle in graph copy G^s, for $s \in S$, is described by the pseudo-code in Algorithm 1. It is similar to the K first iterations of the classical Bellman-Ford algorithm for maximum cost walks starting from vertex $s \in S$. The algorithm computes walks $p_k(v)$ from s to

v with exactly k arcs and their weights $\bar{w}_k(v)$ for all $v \in V^s, k \in [\![0\,;K]\!]$. The walk obtained by adding arc $(v_r, v) \in A^s$ to any walk $p = (v_1, \ldots, v_r)$ is denoted as $p|v$. For each $k \in [\![0\,;K]\!]$, the algorithm stores in Q_k the list of vertices v such that $\bar{w}_k(v) > -\infty$ to avoid considering vertices that cannot be reached with exactly k arcs from s.

initialization: $\bar{w}_0(s) := 0$, $\bar{w}_0(v) := -\infty, \forall v \in V^s \setminus \{s\}$
$Q_0 := \{s\}$, $p_0(s) := (s)$
for $k \in [\![1\,;K]\!]$ **do**
  $\bar{w}_k(v) := -\infty, \forall v \in V^s$, $Q_k := \emptyset$,
  for $u \in Q_{k-1}$ **do**
    for $(u, v) \in A^s$ **do**
      if $\bar{w}_{k-1}(u) + \bar{w}_{uv} > \bar{w}_k(v)$ **then**
        $\bar{w}_k(v) \leftarrow \bar{w}_{k-1}(u) + \bar{w}_{uv}$
        $p_k(v) \leftarrow p_{k-1}(u)|v$
        $Q_k \leftarrow Q_k \cup \{v\}$
$k^* = \text{argmax}_{k \in [\![1;K]\!]} \bar{w}_k(s)$
if $\bar{w}_{k^*}(s) > 0$ **then**
  return $p_{k^*}(s)$
else
  return ()

Algorithm 1: Dynamic programming search for a positive cycle in G^s

Let $\mathcal{P}_k^s$ be the set of walks of G^s with length k starting from s, and among those let $\mathcal{P}_k^s(v)$ be the walks from s to v, $v \in V^s$. With these notations, it is straightforward to show that the values computed by Algorithm 1 satisfy, for all $v \in V^s$, the classical Bellman-Ford recurrence relation:

$$\max_{p \in \mathcal{P}_k^s(v)} \{\bar{w}(p)\} = \max_{u|(u,v) \in A^s} \left\{ \bar{w}_{uv} + \max_{p' \in \mathcal{P}_{k-1}^s(u)} \{\bar{w}(p')\} \right\}.$$

As a consequence, we can directly describe the behavior of the algorithm by the following property.

Property 1. *At the end of any iteration $k \in [\![1\,;K]\!]$ of the* for *loop of Algorithm 1, Q_k is the set of vertices that can be reached from s with exactly k arcs, and for all $v \in Q_k$, $\bar{w}_k(v)$ is the maximum reduced cost of a walk with length k from s to v in G^s. More formally, we have*

- $Q_k = \{v \in V : \mathcal{P}_k^s(v) \neq \emptyset\}$;
- $\forall v \in V, \bar{w}_k(v) = \max\{\bar{w}(p) : p \in \mathcal{P}_k^s(v)\}$.

As a result of Property 1, Algorithm 1 calculates the maximum reduced cost walk of length k from s to s for $k \in [\![1\,;K]\!]$. Since the longest walk problem is a relaxation of the (elementary) longest path problem, this directly guarantees that Algorithm 1 returns an empty walk if and only if there is no positive cycle going through s in G^s. When the algorithm returns a positive walk, this walk may contain a subtour if its length is greater than or equal to four. In this case, the contained subtour necessarily has positive length, and constitutes a positive cycle that can be added to the restricted master problem.

4.1.2 Dynamic programming search for positive chains

The search for a positive chain starting from vertex $s \in D$ is described by the pseudo-code in Algorithm 2. It starts with the same Bellman-Ford iterations as in Algorithm 1, and returns the first positive chain found, if any.

initialization: $\bar{w}_0(s) := 0$, $\bar{w}_0(v) := -\infty, \forall v \in V^s \setminus \{s\}$
$Q_0 := \{s\}$, $p_0(s) := (s)$
for $l \in [\![1\,;L]\!]$ **do**
  $\bar{w}_l(v) := -\infty, \forall v \in V^s$, $Q_l := \emptyset$,
  for $u \in Q_{l-1}$ **do**
    for $(u,v) \in A^s$ **do**
      if $\bar{w}_{l-1}(u) + \bar{w}_{uv} > \bar{w}_l(v)$ **then**
        $\bar{w}_l(v) \leftarrow \bar{w}_{l-1}(u) + \bar{w}_{uv}$
        $p_l(v) \leftarrow p_{l-1}(u)|v$
        $Q_l \leftarrow Q_l \cup \{v\}$
for $l \in [\![1\,;L]\!]$ **do**
  for $v \in V^s$ **do**
    if $\bar{w}_l(v) > 0$ *and* $p_l(v)$ *is a chain* **then**
      return $p_l(v)$
return ()

Algorithm 2: Dynamic programming search for a positive chain in G^s

Property 1 also applies to Algorithm 2. This means that the algorithm computes a maximum reduced cost walk with l arcs from s to each vertex $v \in V^s$ for $l \in [\![1\,;L]\!]$. The algorithm returns such a walk only if it has positive reduced cost and it can verify that it is a chain. Therefore, if Algorithm 2 returns a non-empty walk then G^s contains a positive chain that can be added to RMP. However, when the algorithm returns empty, it may be due to two reasons: (i) there are no positive walks, or (ii) all the positive walks found by the algorithm contain a subtour and therefore are not chains. In the first case, we may conclude that there are no positive chains in G^s since the longest walk problem is a relaxation of the (elementary) longest path problem, resulting in the following property.

Property 2. *If $\bar{w}_l(v) \leq 0$, for all $l \in [\![1\,;L]\!]$ and all $v \in V^s$ when Algorithm 2 returns, then there exists no positive chain in G^s.*

Despite this positive result, G^s may still contain a positive chain starting from s while Algorithm 2 fails to find any, even when G^s contains no positive cycle with length at most K. Indeed, Algorithm 2 may return empty in the case where all positive walks contain an infeasible subtour (i.e., a positive cycle of length $\geq K+1$) while failing to identify the existence of a positive chain. We illustrate this behavior on an example that was proposed by Plaut et al. (2016b) with $K = 3$ and $L = 5$ which is reproduced in Figure 1. The example relies on the existence of a positive cycle of length $K+1 = 4$, $(p_1, p_2, p_3, p_4, p_1)$, which *conceals* the only positive chain, $(s, p_5, p_2, p_3, p_4, p_1)$, whose length is $L = 5$. On this example, Algorithm 2 will identify the walk $(s, p_1, p_2, p_3, p_4, p_1)$ as the longest walk from s to p_1, and will return empty since all identified walks are either of non-positive weight or contain an infeasible subtour. We remark that, this would not be an issue if K was larger, in which case the positive cycle $(p_1, p_2, p_3, p_4, p_1)$ could be returned to the master problem, or if L was smaller, in which case no walk from s to $v \in V^s \setminus \{s\}$ could contain an infeasible cycle.

As a matter of fact, it is possible to find a positive cycle or chain or prove that none exists in polynomial time through executions of Algorithms 1 and 2 when $L \leq K+1$. We formalize this result in the following proposition.

Proposition 1. *If $L \leq K+1$ and G^s does not contain any positive cycle, G^s contains a positive chain if and only if Algorithm 2 returns a non-empty walk when applied to G^s.*

Proof. It is clear that G^s contains a positive chain if Algorithm 2 returns a non-empty walk. To

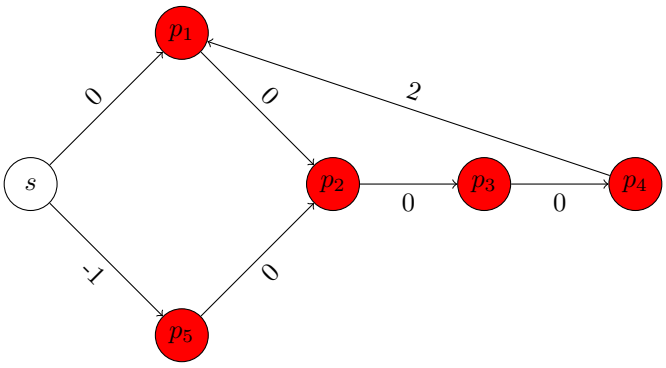


Figure 1: Reproduction of the counterexample of Plaut et al. (2016b)

prove the reverse implication, observe that vertex s corresponds to an altruistic donor, hence has no incoming arc. As a consequence, any subtour of a walk starting from s with length at most L has length at most $L-1$. If $L-1 \leq K$ and G^s does not contain any positive cycle, then any such subtour has a nonpositive reduced cost. As a result, there exists a longest walk from s to $v \in V^s \setminus \{s\}$ that does not contain a subtour and is therefore a chain. Hence, G^s contains a positive chain starting from s only if it contains a positive walk starting from s. Property 2 then yields the result. □

Corollary 1. *The graph G^s contains a positive chain with length at most $K+1$ if and only if G^s does not contain any positive cycle and Algorithm 2 returns a non-empty walk when applied to G^s.*

As a conclusion, we are now able to state a set of necessary conditions for Algorithms 1 and 2 to conclude incorrectly that there are no positive cycles and chains to be added to RMP.

Proposition 2. *The execution of Algorithm 1 for $G^s, \forall s \in S$, and of Algorithm 2 for $G^s, \forall s \in D$, is not sufficient to solve the pricing problem if all of the following conditions are satisfied simultaneously:*

1. *G does not contain any positive cycle;*
2. *for all $s \in D$, G does not contain any positive chain starting from s with length at most $K+1$;*
3. *for all $s \in D$, Algorithm 2 returns an empty chain when applied to G^s;*
4. *there exists $s \in D$ such that Algorithm 2 finds a positive walk that includes a subtour,* i.e.*, there exists $v \in V^s$ and $l \in [\![K+2\,;L]\!]$ such that $\bar{w}_l(v) > 0$.*

The above conditions are all met in the counterexample proposed by Plaut et al. (2016b), but they seem to be difficult to reproduce in the context of the column generation algorithm using realistic instances of the KEP. We remark, for instance, that the graph of the counterexample (Figure 1) does not contain any cycle with length at most 3. Moreover, the reduced costs given in the example are possible only if arcs do not all have the same weight, so this instance could not be obtained in the context of maximizing the number of transplants. Despite this, to prove the optimality of a solution of RMP, we need to be able to certify that no positive column exists when the conditions of Proposition 2 are not met. We propose to solve an IP formulation in this case. We present one such formulation in the next section.

4.1.3 Integer programming formulation of the pricing problem

As stated above, the execution of Algorithm 1 is sufficient to identify a positive cycle if G contains one. Moreover, the execution of Algorithm 2 is sufficient to prove that no feasible positive chain starting from $s \in D$ exists if the algorithm does not identify any positive walk starting from s. As a consequence, it is only when the above conditions are not satisfied that we need another solution approach to search for a positive chain in G^s, for $s \in D$.

The problem that needs to be solved in order to identify positive chains is a variant of the elementary shortest path problem (ESPP) in the presence of negative cycles. Various integer programming formulations of the ESPP have been studied and compared in Taccari (2016). In their numerical analysis, the best results were obtained with a strengthened version of the classical Dantzig-Fulkerson-Johnson formulation based on subtour elimination. This model requires a fixed sink node, so we add an artificial node t to G^s, for $s \in D$, together with a set of incoming arcs $\{(u,t) : \forall u \in V^s\}$. The resulting graph is denoted by $\hat{G}^s = (V^s \cup \{t\}, \hat{A}^s)$ where $\hat{A}^s = A^s \cup \{(u,t) : \forall u \in V^s\}$. The model then involves a set of arc variables $y_{u,v}$ for $(u,v) \in \hat{A}^s$. We adapt the formulation of Taccari (ibid.) to the search for positive chains in $\hat{G}^s$ as follows:

$$\max \quad \sum_{(u,v)\in A^s} \bar{w}_{u,v} y_{u,v} \tag{25}$$

$$\text{s.t.} \quad \sum_{v|(v,u)\in A^s} y_{v,u} - \sum_{v|(u,v)\in A^s} y_{u,v} \geq 0 \qquad \forall u \in V^s \setminus \{s\} \tag{26}$$

$$\sum_{v|(s,v)\in A^s} y_{s,v} = 1 \tag{27}$$

$$\sum_{v\in V^s} y_{v,t} = 1 \tag{28}$$

$$\sum_{(u,v)\in A^s} y_{u,v} \leq L \tag{29}$$

$$\sum_{(u',v')\in \hat{A}^s | u'\in U, v'\notin U} y_{u',v'} \geq \sum_{v|(u,v)\in A^s} y_{u,v}, \qquad \forall U \subseteq V^s \setminus \{s\}, \forall u \in U \tag{30}$$

$$y_{u,v} \in \{0,1\} \qquad \forall (u,v) \in A^s. \tag{31}$$

Constraints (26)-(29) are flow constraints and they bound the length of the walk to L. Constraints (30) are called generalized cutset inequalities (GCS). They eliminate subtours by ensuring that for all subsets of $V^s \setminus \{s\}$, the number of arcs outgoing from the set is larger than the number of arcs outgoing from any vertex of the set. There is an exponential number of GCS inequalities, so they are added in a cut generation procedure. The specificity of our application is that when solving this integer program, we know that the graph does not contain any positive cycle (with length $\leq K$), and probably a very small number of positive cycles with length at most $L-1 > K$. So we separate GCS inequalities only for integer solutions of the relaxed problem. Given that the cardinality of the support of such a solution is bounded by $L+1$, separation can be done quickly by searching for connected components using Tarjan's algorithm (Tarjan 1976). If the optimal value of this model is non-positive for $s \in D$ then G does not contain any positive chains.

4.1.4 Algorithmic improvements

It has been observed, in the literature, that the solution of the pricing problem takes the most significant amount of time when solving the kidney exchange problem using the column generation algorithm (see for instance Riascos-Álvarez et al. 2020). As such, significant numerical gains can

be obtained by improving the implementation of the pricing algorithms used. In this section, we provide several insights on the algorithmic choices that had most impact on our implementation.

We first remark that several immediate improvements over the presented versions of Algorithms 1 and 2 can be obtained by storing only the predecessors of the vertices in the computed walks, and stopping the algorithm as soon as a positive cycle is found. Moreover, in the last iteration of the algorithm, it suffices to update the distance labels only for the arcs going to the source.

Further, a significant improvement can be obtained by adding to the lists of reachable vertices only those vertices that can be involved in a positive cycle or chain. More specifically, denoting the maximum arc reduced cost by $\bar{w}^{\max} = \max_{a \in A} \bar{w}_a$, the maximum cost of a cycle going through node $v \in V^s$ is $\bar{w}_k(v) + (K-k)\bar{w}^{\max}$ which is an upper bound obtained at iteration $k \leq K$ of Algorithm 1. So vertex v is added to Q^k at step 8 of Algorithm 1 only if $\bar{w}_k(v) + (K-k)\bar{w}^{\max} > 0$. Algorithm 2 can be improved similarly.

Finally, in most of the instances considered in the literature, the arc weights are based solely on the destination node and not the source node. This reflects, for instance, the maximization of the number of transplants, in which case the arc weight is equal to one, independent of the donor. It can also reflect situations where different weights are assigned to the patient-donor pairs depending on the level of urgency of the transplant expected by the patient (and any donor is equally acceptable). Denoting the weight of vertex $v \in V$ by w_v, the weight of each arc $(u, v) \in A$ is then given by $w_{u,v} = w_v$. When treating the root node of the branch-and-price tree, this means that the reduced cost of an arc may also be written as the sum of independent terms depending on one extremity of the arc. More specifically, for all $(u, v) \in A$, we have:

$$\bar{w}_{u,v} = \begin{cases} w_v - \lambda_v, & \text{if } u \in P \\ w_v - \lambda_v - \lambda_u, & \text{if } u \in D. \end{cases}$$

We remark then that, at any iteration k of Algorithms 1 and 2, and for any vertex $v \in P$, and $u_1, u_2 \in V \setminus D$ predecessors of v, we have

$$\bar{w}_{k-1}(u_1) \geq \bar{w}_{k-1}(u_2) \implies \bar{w}_{k-1}(u_1) + \bar{w}_{u_1,v} \geq \bar{w}_{k-1}(u_2) + \bar{w}_{u_2,v}.$$

As a consequence, if we sort Q_{k-1} by decreasing values of $\bar{w}_{k-1}(\cdot)$, we need to treat at most one arc entering each vertex $v \in P$. This means that Algorithms 1 and 2 can be executed in $\mathcal{O}(K\,|V|)$ and $\mathcal{O}(L\,|V|)$, respectively, instead of $\mathcal{O}(K\,|A|)$ and $\mathcal{O}(L\,|A|)$. This observation is unfortunately only valid at the root node of the branch-and-price tree since the dual variables introduced by branching on arcs destroy the above cost structure. However, the numerical improvement obtained is still significant since most calls to Algorithms 1 and 2 are done at the root node (see 4.2 for details on the quality of the linear relaxation).

4.2 Heuristic search for a feasible solution and branching rules

As already observed by Riascos-Álvarez et al. (2020), DCF has a very strong linear relaxation. Indeed, in our numerical tests, we found the dual bound provided by DCF^{LP} to be exact for almost all of the instances tested. On the other hand, optimal solutions provided by DCF^{LP} are not always integer. As such, it might be important to search for integer solutions to prove that the integrality gap is indeed zero in order to avoid unnecessary branching.

To this end, when the column generation algorithm converges with a fractional optimal solution, we solve the integer programming counterpart of RMP, RMP^{IP}, to find an integer feasible solution. In most cases, RMP^{IP} and RMP have the same optimal value, so it is sufficient to solve RMP^{IP} in order to prove optimality. However, in some other cases, there is a gap between the values of RMP^{IP} and RMP, and branching is performed. Unfortunately, branching may prove inefficient in improving the incumbent solution: RMP has many alternative optimal solutions and branching on

a single arc is often not sufficient to eliminate all fractional solutions. As such, it might be necessary to explore multiple nodes down the branching tree before an integer solution can be found.

The most promising strategy then seems to be spending an extra computational effort in searching for a feasible solution at the root node. We thus implemented a heuristic algorithm that generates new cycles and chains in the hope that they will be selected to create an integer solution with an improved value. To do so, we randomly choose and remove half of the cycles and chains from RMP and solve it by column generation a second time. Every new column is then added to RMP^{IP} before solving it again. This strategy works particularly well for the kidney exchange problem because there are many feasible solutions with the same value. It is often possible to find an integer solution with the same value as the bound provided by DCF^{LP} through combinations of new columns. When this strategy does not prove successful it is necessary to branch as described at the beginning of the section.

4.3 Column generation strategies

The goal of the pricing problem is to find at least one positive cycle or chain in each graph copy $G^s, s \in S \cup D$. As mentioned in Section 4.1.4, when executing Algorithms 1 and 2, we always return the first positive cycle or chain found during the dynamic programming search. Alternatively, one can retain all positive cycles/chains obtained from a copy and return one with the most positive reduced cost.

The latter strategy has been tested, but it was significantly less efficient for two reasons. First, it requires that every iteration is completed in Algorithms 1 and 2 instead of terminating them early when possible. Second, it favors long cycles and chains, especially in the first iterations of the column generation algorithm, even though it is possible to obtain optimal solutions including short cycles and chains.

In the remainder of the section, we discuss how to take advantage of the packing constraints (23) to accelerate the search of cycles and chains in some graph copies. We then describe a strategy that ignores graph copies where no positive cycle or chain was found in a previous column generation iteration.

4.3.1 Vertex-disjoint columns

The only constraints of DCF, the packing constraints (23), impose that for each vertex $v \in V$, at most one cycle or chain involving this vertex will be selected in any feasible solution. Similar constraints in crew scheduling problems, imposing that each task must be assigned at most once, lead Desaulniers et al. (2002) to generate *task disjoint columns* at each iteration of the column generation algorithm. In this context, a column is a rotation consisting of a sequence of tasks that can be assigned to an employee. Two columns or rotations are task-disjoint when the rotations have no task in common.

In the kidney exchange problem, a column is a cycle or a chain, so two columns are said to be vertex-disjoint when the corresponding cycles/chains do not have any vertex in common. In order to impose the generation of vertex-disjoint columns in the context of sequential treatment of graph copies by Algorithms 1 and 2, it suffices to remove the vertices that are already covered by the walks found for subsequent copies. More specifically, if some positive cycle or chain c is found in graph G^s for some $s \in S \cup D$, then every vertex of c can be removed from the graph copies that still need to be treated. In particular, if there is $v \in V(c)$ such that $v \in S$, then G^v does not even need to be treated.

Enforcing the generation of vertex-disjoint columns may yield smaller pricing solution time by reducing the size and number of graph copies to be treated as outlined above. On the other hand, the ability to find an optimal solution by solving RMP^{IP} will depend on the size and diversity of the pool of columns used in RMP^{IP}. Imposing a pure vertex-disjoint column generation strategy

unfortunately produces a smaller pool of columns. As such, there is a trade-off between the ability to quickly solve pricing problems and the ability to produce integer solutions at the root node. In our implementation, we manage this trade-off by generating no more than κ columns covering the same vertex at each iteration.

Finally, the sets S and D are randomly shuffled before each iteration of the column generation algorithm in order to promote diversity in the pool of columns generated. Without this operation, imposing vertex-disjoint column generation may often concentrate on the same graph copies while ignoring others.

4.3.2 A tabu list of graph copies

The last strategy that we implemented in our branch-and-price algorithm consists in ignoring a graph copy $G^s, s \in S \cup D$, if no positive cycle or chain was found in it in a previous column generation iteration. It may be numerically advantageous to do so for two reasons. First, the dual solution of RMP does not change much from one iteration to another, and when it does, it usually leads to larger dual values hence smaller values of $\bar{w}_{u,v}$ for $(u, v) \in A$ because more vertices are covered at each iteration. As a consequence, it is unlikely that a positive cycle or chain is found in G^s if none was found at a previous iteration. Further, the executions of Algorithms 1 and 2 that end up returning empty are the most time-consuming ones.

In practice, if Algorithm 1 or 2 returns empty when executed on $G^s, s \in S \cup D$, we store s in a *tabu list* L. At each iteration of the column generation algorithm, we then search for a positive cycle or chain in G^s for $s \in (S \cup D) \setminus L$. Eventually, we obtain $L = S \cup D$ at which point we empty L and execute Algorithm 1 or 2 once more on each copy. If no positive column is found then we proceed to certifying the optimality of the current RMP solution. To do so, we need to guarantee that Algorithm 2 did not falsely conclude that no positive chain could be found. As such, we verify the conditions of Proposition 2 and if they are satisfied, we solve the integer programming formulation of the pricing problem presented in Section 4.1.3. We remark that this IP formulation does not need to be solved up to this point and is only solved if a certain number of conditions are satisfied simultaneously.

5 Numerical experiments

In this section, we present the numerical results obtained on a benchmark using our branch-and-price implementations as well as the compact MIP formulations presented in Section 3.

All our experiments are conducted using a 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 2.5 GHz machine with 128Go RAM running Linux OS. The resources of this machine are strictly partitioned using the Slurm Workload Manager[1] to run several tests in parallel. The resources available for each run (algorithm-instance) are set to 4 threads and a 20 Go RAM limit (we remark that our branch-and-price algorithms does not benefit from parallel processing since we choose to solve the pricing problems for each graph copy sequentially). This virtually creates six independent machines, each running one single instance at a time. All linear and mixed integer linear programs, are solved using Gurobi 9.1.2 with default parameters and 4 threads.

5.1 Description of the benchmark

Our benchmark includes randomly generated instances based on three different generators described in the literature, *i.e.*, the Saidman generator presented in Saidman et al. (2006), the heterogeneous generator presented in Ashlagi et al. (2013), and the sparse generator presented in John P Dickerson et al. (2012).Our implementation of these random generators is mostly a translation of the

[1] `https://slurm.schedmd.com/` (accessed July 2022)

corresponding Java functions in the code publicly shared by John Dickerson[2]. Saidman instances were often used in the literature in order to test branch-and-price implementations. For instance Abraham et al. (2007), Mak-Hau (2017) and Riascos-Álvarez et al. (2020) have used Saidman instances that are publicly available in the PrefLib dataset. This set, contains 310 instances, and consists of 10 randomly generated instances of kidney exchanges with 16, 32, 64, 128, 256, 512, 1024, 2048 pairs and, as a percentage of the number of pairs, altruists at 0%, 5%, 10%, and 15%.

The Saidman generator involves generating patients with random characteristics such as blood type, sex, and probability of being tissue-type incompatible with a randomly chosen donor based on low, medium or high percentage reactive antibodies (PRA) levels for the patient. These probabilities follow real-world population data. Each patient is then assigned a randomly generated potential donor with random blood type and relation to the patient. If the patient and potential donor are incompatible, the two are entered into the kidney exchange program as a pair. Blood and tissue type information is then used to determine compatibility information within the exchange program. The sparse generator is based on the Saidman generator, it adjusts the probabilities used by the generator in order to mimic the state of the UNOS pool as of April, 2013. The resulting instances are sparser since both the probability of a patient having high PRA and the probability of incompatibility in this case (as well as for low and medium PRA cases) are increased. The heterogeneous generator, on the other hand, is based on the existence of two types of patients: low PRA and high PRA. While the probability that a low PRA-patient being compatible with the donor of another pair is quite high, this probability is very low for high PRA-patients (also known as highly sensitized patients).

5.2 A Julia package for the kidney exchange problem

The solution methods described in Sections 3 and 4 and the random generation of instances have been implemented using the Julia language (Bezanson et al. 2017). Every interaction with mathematical programming models and solvers is managed using the functions of the mathematical optimization package JuMP (Dunning et al. 2017). Our implementation can be used with different mathematical programming solvers. Among these, Gurobi and CPLEX are two commercial solvers that may currently be used with a free academic license, and the other three are free and open software. The solver may be chosen among GLPK[3], Cbc (Forrest et al. 2022), Gurobi [4], CPLEX [5] or a combination of GLPK as the linear programming solver and Cbc as the integer programming solver. For reproducibility of our results and in order to facilitate future research on the topic, our code is publicly available on Github [6]. It is distributed under an MIT license. As a consequence, if one of the three open solvers is selected, the code can be used by anyone under a permissive open license. It is currently the first publicly available implementation that allows to solve the kidney exchange problem with an efficient branch-and-price algorithm.

We have chosen Julia for its great prototyping capabilities, its high performance and the existence of a rich and mature ecosystem of optimization packages[7]. In our view, just as Python or other non-compiled languages, Julia yields easy to read code, which should help understand our technical choices of implementation for those interested in these details. Our code is distributed as a Julia package KidneyExchange.jl. As such, it can be readily installed from any Julia prompt with a single command. Documentation of available functions is available on the Github repository of the package or directly from a Julia prompt. Given that no external library is required, there is no real knowledge required in order to reproduce our results. We also sincerely hope that these

[2]Github repository of the code: `https://github.com/JohnDickerson/KidneyExchange`
[3]GLPK website: `https://www.gnu.org/software/glpk/`
[4]CPLEX website: `https://www.gurobi.com`
[5]Gurobi website: `https://www.ibm.com/products/ilog-cplex-optimization-studio`
[6]Github repository: `https://github.com/jeremyomer/KidneyExchange.jl`
[7]See the JuMP-dev organization

choices will encourage the dynamic research community working on the subject to contribute some of their future developments to this package.

5.3 Numerical Results

In this section, we numerically test and compare our branch-and-price implementations as well as the compact formulations proposed in Section 3 on a number of benchmark instances. For all results presented in this section, the parameter κ that governs the maximum number of columns involving a given vertex that can be generated in an iteration of the column generation algorithm is set to 6.

5.3.1 Comparison of compact formulations on smaller instances

In this first set of numerical tests, we compare the numerical performance of the two compact MIP formulations proposed in Section 3 ((17)-(19) and (20)-(22)). We test these formulations on the three types of instances of our benchmark, *i.e.*, Saidman, heterogeneous and sparse, with instance sizes $|P| \in \{128, 256, 512, 1024\}$. For each instance type and size, we fix the number of altruistic donors at %10 of the number of pairs and generate 10 instances. We choose parameters $K = 3$ and $L \in \{4, 6\}$ as representative of the current kidney exchange practice. We remark that the Saidman instances we use are those provided in the PrefLib database.

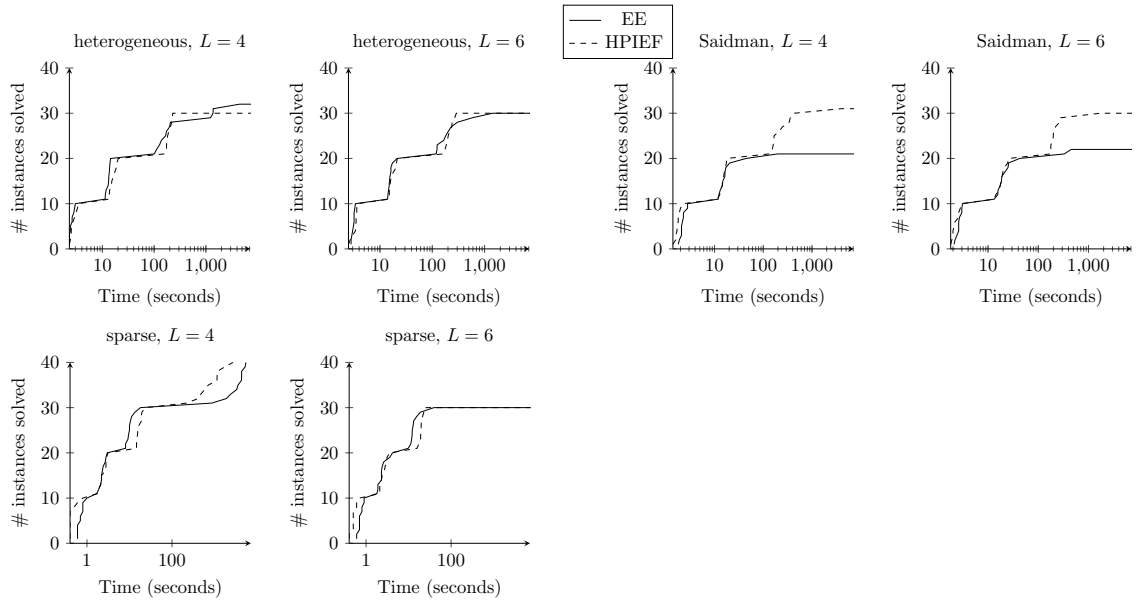


Figure 2: Performance profiles comparing the two compact formulations.

In Figure 2, we present the number of instances solved (out of 40 instances) using each compact formulation as a function of time (with a time limit of 2 hours). While both formulations managed to solve the smaller instances with 128 and 256 pairs within seconds, the extended-edge based formulation (17)-(19) started to struggle, especially with Saidman instances, starting from 512 pairs. Neither formulation could solve instances with 1024 pairs successfully (with the exception of the sparse instances with $K = 3$ and $L = 4$) even with all the model reductions and the symmetry breaking objective function. In Table 4 in the Appendix, we provide further insight into these results by providing the average solution time for those instances that were solved to optimality. The reported times do not include the time required to build the models using JuMP which may

be as large as 35 minutes for the larger Saidman instances. We remark that when instances are not solved to optimality both models terminate with extremely large optimality gaps.

Based on these results, the position-indexed model performs better than the extended-edge-based model although for smaller instances the two models are comparable. There does not seem to be a gain from the smaller number of variables afforded by the extended-edge-based model. The superior linear relaxation of the position-indexed HPIEF model justifies the increased number of variables. Finally, not surprisingly, sparse instances are those better solved by the compact formulations. Indeed, these instances require a smaller number of variables in the first place, and profit more from further model reduction based on pairwise distances.

5.3.2 Comparison of branch-and-price algorithms on PrefLib instances

In this section, we compare the two branch-and-price algorithms we have implemented, one based on the disaggregated cycle and chain formulation DCF (23) and the other based on the position-indexed chain edge formulation PICEF proposed by John P. Dickerson, Manlove, et al. (2016). The difference between these two algorithms comes from the treatment of chains in each formulation. While DCF uses chain selection variables and generates chains through the solution of subproblems, PICEF generates chains through the use of position-indexed chain edge variables. Both formulations use cycle selection variables and generate cycles through the solution of subproblems. In the algorithm based on PICEF, we initially generate up to 20 cycles of size 2 per vertex and add them to the restrictions $\hat{\mathcal{C}}_K^s, s \in S$ to warm up dual values.

As is typically done in the literature, we test both algorithms on PrefLib instances with $|P| \in \{16, 32, 64, 128, 256, 512, 1024, 2048\}$ and the number of altruists expressed as a percentage of the number of pairs (%5,%10 and %15). There are a total of 230 instances. We choose $K \in \{3, 4\}$ and $L \in \{3, 4, 5, 6\}$. We remark that DCF and PICEF lead to the same algorithm when there are no altruists in the graph or when $L = 0$. In Figure 3, we present the number of instances solved to optimality as a function of time by both algorithms. The DCF-based algorithm was able to solve all instances under 20 seconds whereas the PICEF-based algorithm was not able to solve some of the largest ($|P| = 2048$) instances when $L \in \{5, 6\}$. The PICEF-based algorithm was faster for the smaller instances ($|P| \leq 128$), whereas the DCF-based algorithm outperformed it for larger instances. This was especially the case for larger values of L where the DCF-based algorithm was one-to-two orders of magnitude faster. Additionally, the DCF-based algorithm solved all instances at the root node except for one whereas the PICEF-based algorithm treated up to 5 nodes for the largest instances.

5.3.3 Branch-and-price performance on larger instances

In this section, we test the DCF-based branch-and-price algorithm on larger kidney exchange instances. We test on the three types of instances, Saidman, heterogeneous and sparse, with $|P| \in \{2048, 6000, 10000\}$ and the number of altruists fixed at %10. We choose $K \in \{2, 3, 4\}$ and $L \in \{0, 4, 6\}$. We use the instances provided in the PrefLib database for Saidman instances when $|P| = 2048$. Otherwise, all instances are randomly generated and there are 10 instances in each category.

The number of instances solved to optimality is given in Table 1. We first remark that the branch-and-price algorithm was able to solve almost all of the instances considered for $K \in \{3, 4\}$ and $L \in \{0, 4, 6\}$ with the exception of one sparse instance for $K = 4$ and $L = 0$ for which an optimality gap of %0.015 was obtained. We further report the average solution times for these instances in Table 2. We remark that the largest Saidman instances (2048 pairs with 205 altruists) from the PrefLib library were solved within seconds whereas Riascos-Álvarez et al. (2020) have reported solution times of up to 20 minutes for these instances. Similarly, heterogeneous and sparse instances were solved to optimality successfully, although the solution time was significantly

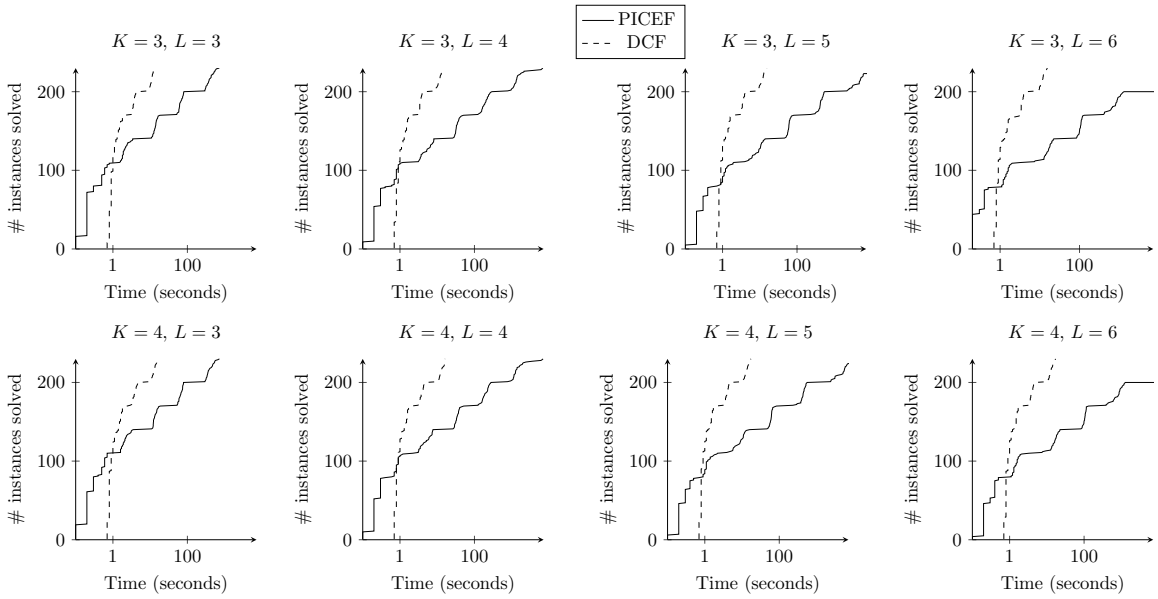


Figure 3: Performance profiles comparing the two branch-and-price algorithms on the PrefLib instances.

higher. The difficulty of solving such instances was noted by Riascos-Álvarez et al. (2020) who constructed sparse instances starting from PrefLib instances by removing arcs according to a given highly sensitized patient percentage and compatibility failure probability for these patients. In their numerical results, most of these instances with 256 and 512 patients were not solved to optimality within 30 minutes. These results indicate that our branch-and-price algorithm is able to solve instances that could not be solved with existing methods, and can be up to two orders of magnitude faster. Additionally, this is, to the best of our knowledge, the first time such large instances ($|P| \in \{6000, 10000\}$) are solved to optimality for the cycles-and-chains variant in the literature.

On the other hand, the algorithm was not able to solve some Saidman instances for $K = 2$ and $L = 0$ and some sparse instances for $K = 2$ and $L \in \{4, 6\}$ although the maximum optimality gap among all the instances that were not solved to optimality was %0.38. Indeed, these instances include many alternative optimal solutions. Branching, therefore, seems ineffective in this case, especially for the denser Saidman instances. While the maximum number of nodes explored with $K \in \{3, 4\}$ and $L \in \{0, 4, 6\}$ was 29 with %82 of instances solved at the root node, when $K = 2$ and $L = 0$ the branch-and-price algorithm has explored as much as 18,000 nodes. When $K = 2$ and $L = 4, 6$, the difficulty of solving the master problem was also an important factor. In this case, almost %95 of the solution time for the instances that were not solved to optimality was spent trying to solve RMP^{IP}. The results provided here are, to the best of our knowledge, the first numerical tests for $K = 2$ and various values of L. Indeed, the problem can be solved through a polynomial-time matching algorithm when $K = 2$ and $L = 0$. For this reason, branch-and-price algorithms have historically not been tested for $K = 2$. Our findings highlight that this case would merit further investigation when $L \neq 0$.

We finally comment on the value of long chains for these instances. To this end, we compare the optimal value for $K = 3$ when increasing the value of L. Increasing the value of L from 0 to 4 increases the objective value by %9.5, %13, and %5.4 for heterogeneous, Saidman and sparse instances, respectively. Increasing the value further from 4 to 6 only has an impact on

		2048			6000			10000		
K	L	het	Said	spa	het	Said	spa	het	Said	spa
2	0	10	7	10	10	3	10	10	6	10
	4	10	10	7	10	10	6	10	10	6
	6	10	10	7	10	10	3	10	10	6
3	0	10	10	10	10	10	10	10	10	10
	4	10	10	10	10	10	10	10	10	10
	6	10	10	10	10	10	10	10	10	10
4	0	10	10	10	10	10	10	10	10	9
	4	10	10	10	10	10	10	10	10	10
	6	10	10	10	10	10	10	10	10	10

Table 1: Number of instances solved to optimality using the DCF-based branch-and-price algorithm on larger instances.

		2048			6000			10000		
K	L	het	Said	spa	het	Said	spa	het	Said	spa
2	0	6.3	8.7	4.1	37.6	103.5	22.1	99.1	120.8	65.2
	4	30.6	15.8	1610.6	398.0	76.1	5418.2	1799.4	266.2	5949.5
	6	29.8	14.5	1915.8	995.7	75.4	6361.4	1390.4	265.1	5966.5
3	0	6.8	7.7	112.9	54.4	72.5	78.1	177.8	249.8	205.6
	4	9.9	12.0	7.6	127.6	98.4	60.4	520.9	352.3	399.0
	6	10.7	11.9	7.6	138.7	99.1	64.1	546.4	355.4	409.7
4	0	9.0	11.3	139.4	95.3	96.3	670.2	375.3	418.4	5386.3
	4	11.2	13.1	12.9	145.1	123.7	459.6	562.1	463.2	4160.5
	6	12.0	13.4	12.9	156.5	121.7	459.0	589.0	463.5	3713.0

Table 2: Average solution time (for instances solved to optimality) using the DCF-based branch-and-price algorithm on larger instances.

the heterogeneous instances with an average increase of %0.4 (similar values are obtained when $K = 4$). These findings support those of Riascos-Álvarez et al. (2020) with a larger benchmark. Indeed, chains do not need to be very long in order to have a significant impact and the added value of longer and longer chains seems to be marginal.

5.4 Impact of algorithmic choices

In this section, we evaluate the impact of certain algorithmic choices on the DCF-based branch-and-price algorithm. To do so, we choose the creation of graph copies based on an FVS, S versus the complete set of pairs, P as proposed by Riascos-Álvarez et al. (ibid.) (fvs), the generation of vertex-disjoint columns as proposed in Section 4.3.1 (vdjcol), the use of a tabu list for subproblems as proposed in Section 4.3.2 (tabu), and the use of the restart strategy for generating integer optimal solutions at the root note as proposed in Section 4.2 (restart).

For reasonable use of numerical resources, we use the three types of instances from our benchmark, Saidman, heterogeneous and sparse, with $|P| \in \{2048, 6000\}$ and altruists at %10. We fix $K = 3$, $L = 6$. The results are presented in Table 3 as a percentage difference from the default version of the DCF-based branch-and-price algorithm where we apply all of the algorithmic choices listed above. We present three metrics, the total solution time (Solution time), the number of columns generated at the root node (#Columns), and the number of calls made to the pricing algorithms, Algorithm 1 and 2, (#Call SP). In this table, we present 5 different versions of the

DCF-based branch-and-price algorithm: a version where the FVS-based graph copies are turned off, a version where the vertex-disjoint column generation is turned off, a version where the tabu list for treating the subproblems is turned off, a version where the vertex-disjoint column generation and the tabu list are turned off, and finally a version where all of these are turned off. The default version of the algorithm was able to solve all of the instances considered here at the root node without recourse to the restart strategy for finding integer solutions. As such, in the results presented in Table 3 we do not consider this algorithmic choice.

Based on the results presented in Table 3, turning off each algorithmic choice leads to a degradation in the solution time, except for the FVS-based graph copies, where the solution time may decrease for the smaller ($|P| = 2048$) instances. The decrease in the solution time can be explained by the preprocessing time required to calculate the FVS. Indeed, it is possible that for smaller instances this preprocessing time is more important than any numerical gains obtained for the branch-and-price algorithm. We also remark that the difference in this case is in the order of milliseconds. On the other hand, the increase in the solution time by turning certain options off is more significant. For instance, turning off the vertex-disjoint column generation causes the solution time to worsen up to an order of magnitude for heterogeneous and Saidman instances while significantly increasing the number of columns added to RMP. When coupled with the tabu list being turned off this effect is further amplified with the effect of the tabu list being turned off on its own being much less significant. Interestingly, turning the FVS-based graph copies off on top off vertex-disjoint column generation and the tabu list often improves the solution time while decreasing the number of columns at the root node although the number of calls made to pricing algorithms increases substantially.

As can be expected, turning off the FVS-based graph copies causes an increase in the number of calls made to the pricing algorithms, as does turning off the vertex-disjoint column generation and the tabu list. When compared to the last two, the effect of FVS-based graph copies alone seems marginal especially for heterogeneous and Saidman instances. Indeed, using FVS-based graph copies seems to be computationally advantageous only for the sparse instances based on our results.

				Solution time		#Columns		#Call SP	
	fvs	vdjcol	tabu	2048	6000	2048	6000	2048	6000
heterogeneous		✓	✓	-0.7	8.9	-14.6	-16.4	16.3	27.2
	✓		✓	260.6	977.0	296.5	611.7	201.2	398.4
	✓	✓		13.8	49.3	0.0	-0.2	60.4	81.1
	✓			300.6	1060.8	305.3	637.9	459.2	977.1
				158.1	506.2	106.3	233.9	620.5	1254.0
Saidman		✓	✓	-5.1	7.4	-6.2	-12.9	24.7	20.0
	✓		✓	581.8	1114.4	474.3	721.9	364.4	540.2
	✓	✓		3.9	-2.5	3.7	1.3	34.6	10.6
	✓			728.9	1429.9	515.9	771.9	925.9	976.2
				535.9	1073.1	362.9	430.2	2126.3	2266.9
sparse		✓	✓	19.0	20.8	2.4	-3.3	28.9	27.4
	✓		✓	22.1	73.3	36.1	51.5	20.1	31.5
	✓	✓		6.6	7.5	4.2	0.1	64.6	26.8
	✓			19.7	82.3	38.2	56.1	111.5	119.3
				50.9	98.9	64.2	93.8	197.3	261.7

Table 3: Indicators represented as percentage difference from the default version of the DCF-based branch-and-price algorithm for different versions based on listed algorithmic choices.

We finally evaluate the effect of the restart strategy used at the root node. To this end, we

choose instances solved at the root node by the default version of the DCF-based branch-and-price algorithm for which RMP^{IP} was called twice. These are instances for which an integer optimal solution at the root node was not readily available after the first execution of the column generation algorithm but was found after deleting half the columns and restarting the algorithm. There were only 3 such instances in the sparse category with parameters $K = 3, L = 4$ and $K = 3, L = 6$. Turning the restart off for these instances causes the solution time to increase on average by a factor of 3.5 for $L = 4$ and 4.1 for $L = 6$. Across the three instances the maximum number of nodes explored increases from 1 to 21 for $L = 4$ and to 25 for $L = 6$.

6 Conclusions and future research directions

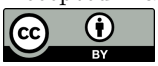
In this article, we present a branch-and-price algorithm for the exact solution of the kidney exchange problem in the presence of altruistic donors. This algorithm is based on the disaggregated cycle and chains formulation where the graph copies are created using a feedback vertex set (FVS) as proposed by Riascos-Álvarez et al. (2020). We additionally, present a branch-and-price algorithm based on the position-indexed chain-edge formulation (PICEF) proposed by John P. Dickerson, Manlove, et al. (2016) as well as two compact formulations. We formalize and analyze the complexity of the resulting pricing problems and identify the conditions under which they can be solved using polynomial-time algorithms. We propose several algorithmic improvements for the branch-and-price algorithms as well as for the pricing problems. We extensively test all of our implementations using a benchmark made up of different types of instances with the largest instances considered having 10000 pairs and 1000 altruists. Our numerical results show that the proposed algorithm can be up to two orders of magnitude faster compared to the state-of-the-art, and can solve instances that could not previously be solved in the literature such as sparse instances and very large instances. They also highlight the significant gains that can be obtained based on our algorithmic improvements. This efficient branch-and-price algorithm additionally permits us to test the importance of the chains on very large instances. Our findings indicate that chains do not need to be very long in order to obtain significant objective improvements.

An important aspect that we did not consider in this article is the presence of uncertainty. Indeed, the compatibility information considered for solving the deterministic problem is susceptible to change once additional tests are performed. In this case, the planned cycles cannot be performed whereas the chains can only be partially performed. In order to overcome this difficulty it is then appropriate to consider a version of the problem under uncertainty either through stochastic or robust optimization paradigms. The existence of an efficient deterministic solver can then significantly improve the solution of these versions under uncertainty as the deterministic problem can show up as a subproblem. This is especially the case since the linear relaxation of the disaggregated cycles and chains formulation is often very strong. Future work may consider this aspect and evaluate the value of considering uncertainty in the problem description with different optimization under uncertainty paradigms.

Acknowledgement

The authors gratefully thank Pierre Navarro, CNRS research engineer at IRMAR, for his skillful support with Julia Language.

Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see `https://www.plafrim.fr/`).



References

Abraham, David J., Blum, Avrim, and Sandholm, Tuomas (2007). "Clearing Algorithms for Barter Exchange Markets: Enabling Nationwide Kidney Exchanges". In: *Proceedings of the 8th ACM Conference on Electronic Commerce* (San Diego, California, USA). EC '07. New York, NY, USA: ACM, pp. 295–304.

Ahuja, Ravindra K., Magnanti, Thomas L., and Orlin, James B. (1988). "Network flows". In:

Anderson, Ross, Ashlagi, Itai, Gamarnik, David, and Roth, Alvin E. (Jan. 20, 2015). "Finding Long Chains in Kidney Exchange Using the Traveling Salesman Problem". In: *Proceedings of the National Academy of Sciences* 112.3, pp. 663–668.

Ashlagi, Itai, Jaillet, Patrick, and Manshadi, Vahideh H. (Apr. 2, 2013). *Kidney Exchange in Dynamic Sparse Heterogenous Pools*. URL: `http://arxiv.org/abs/1301.3509` (visited on 01/27/2022).

Bezanson, Jeff, Edelman, Alan, Karpinski, Stefan, and Shah, Viral B. (Jan. 2017). "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1, pp. 65–98.

Constantino, Miguel, Klimentova, Xenia, Viana, Ana, and Rais, Abdur (Nov. 2013). "New Insights on Integer-Programming Models for the Kidney Exchange Problem". In: *European Journal of Operational Research* 231.1, pp. 57–68.

Desaulniers, Guy, Desrosiers, Jacques, and Solomon, Marius M. (2002). "Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems". In: *Essays and Surveys in Metaheuristics*. Vol. 15. Operations Research/Computer Science Interfaces Series. https://doi.org/10.1007/978-1-4615-1507-4_14. Springer, pp. 309–324.

Dickerson, John P, Procaccia, Ariel D, and Sandholm, Tuomas (2012). "Optimizing Kidney Exchange with Transplant Chains: Theory and Reality". In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, p. 8.

Dickerson, John P., Manlove, David F., Plaut, Benjamin, Sandholm, Tuomas, and Trimble, James (2016). "Position-Indexed Formulations for Kidney Exchange". In: *Proceedings of the 2016 ACM Conference on Economics and Computation - EC '16*. The 2016 ACM Conference. Maastricht, The Netherlands: ACM Press, pp. 25–42.

Dickerson, John P., Procaccia, Ariel D., and Sandholm, Tuomas (Apr. 2019). "Failure-Aware Kidney Exchange". In: *Management Science* 65.4, pp. 1768–1791.

Dunning, Iain, Huchette, Joey, and Lubin, Miles (Jan. 2017). "JuMP: A Modeling Language for Mathematical Optimization". In: *SIAM Review* 59.2, pp. 295–320.

Forrest, John, Ralphs, Ted, Santos, Haroldo Gambini, Vigerske, Stefan, Hafer, Lou, Forrest, John, Kristjansson, Bjarni, Jpfasano, EdwinStraver, Lubin, Miles, Rlougee, Jpgoncal1, Jan-Willem, H-I-Gassmann, Brito, Samuel, Cristina, Saltzman, Matthew, Tosttost, MATSUSHIMA, Fumiaki, and To-St (Jan. 26, 2022). *COIN-OR Branch-and-Cut Solver*. Version releases/2.10.7.

Glorie, Kristiaan M., van de Klundert, J. Joris, and Wagelmans, Albert P. M. (Oct. 2014). "Kidney Exchange with Long Chains: An Efficient Pricing Algorithm for Clearing Barter Exchanges with Branch-and-Price". In: *Manufacturing & Service Operations Management* 16.4, pp. 498–512.

Klimentova, Xenia, Alvelos, Filipe, and Viana, Ana (2014). "A New Branch-and-Price Approach for the Kidney Exchange Problem". In: *Computational Science and Its Applications – ICCSA 2014*, pp. 237–252.

Lam, Edward and Mak-Hau, Vicky (Mar. 2020). "Branch-and-Cut-and-Price for the Cardinality-Constrained Multi-Cycle Problem in Kidney Exchange". In: *Computers & Operations Research* 115, p. 104852.

Mak-Hau, Vicky (Jan. 2017). "On the Kidney Exchange Problem: Cardinality Constrained Cycle and Chain Problems on Directed Graphs: A Survey of Integer Programming Approaches". In: *Journal of Combinatorial Optimization* 33.1, pp. 35–59.

Plaut, Benjamin, Dickerson, John P., and Sandholm, Tuomas (2016a). "Fast Optimal Clearing of Capped-Chain Barter Exchanges". In: Thirtieth AAAI Conference on Artificial Intelligence, p. 7.

— (June 1, 2016b). *Hardness of the Pricing Problem for Chains in Barter Exchanges*. URL: `http://arxiv.org/abs/1606.00117` (visited on 02/28/2021).

Riascos-Álvarez, Lizeth Carolina, Bodur, Merve, and Aleman, Dionne M. (Oct. 4, 2020). *A Branch-and-Price Algorithm Enhanced by Decision Diagrams for the Kidney Exchange Problem*. URL: `http://arxiv.org/abs/2009.13715` (visited on 02/22/2021).

Saidman, Susan L., Roth, Alvin E., Sönmez, Tayfun, Ünver, M Utku, and Delmonico, Francis L. (Mar. 15, 2006). "Increasing the Opportunity of Live Kidney Donation by Matching for Two- and Three-Way Exchanges". In: *Transplantation* 81.5, pp. 773–782.

Taccari, Leonardo (July 2016). "Integer Programming Formulations for the Elementary Shortest Path Problem". In: *European Journal of Operational Research* 252.1, pp. 122–130.

Tarjan, Robert Endre (1976). "Edge-Disjoint Spanning Trees and Depth-First Search". In: *Acta Informatica* 6.2, pp. 171–185.

Appendix

6.1 Position-indexed chain-edge formulation (PICEF)

$$\max \sum_{s\in S}\sum_{c\in\mathcal{C}_K^s} w_c z_c + \sum_{(u,v)\in A}\sum_{l=1}^{L} w_{u,v} y_{u,v,l} \tag{32}$$

$$\text{s.t.} \sum_{s\in S}\sum_{c\in\mathcal{C}_K^s(v)} z_c + \sum_{u|(u,v)\in A}\sum_{l=1}^{L} y_{u,v,l} \leq 1 \qquad \forall v \in V \tag{33}$$

$$\sum_{v|(v,u)\in A} y_{v,u,l} \geq \sum_{v|(u,v)\in A} y_{u,v,(l+1)} \qquad \forall u \in P, l = 1,\ldots,L-1 \tag{34}$$

$$\sum_{v|(u,v)\in A} y_{u,v,1} = 0 \qquad \forall u \in P \tag{35}$$

$$\sum_{v|(u,v)\in A} y_{u,v,l} = 0 \qquad \forall u \in D, l = 2,\ldots,L \tag{36}$$

$$z \in \{0,1\} \qquad \forall s \in S, c \in \mathcal{C}_K^s \tag{37}$$

$$y_{u,v,l} \in \{0,1\} \qquad \forall (u,v) \in A, l = 1,\ldots,L. \tag{38}$$

6.2 Detailed numerical results

		128		256		512		1024	
	L	EE	HPIEF	EE	HPIEF	EE	HPIEF	EE	HPIEF
het	4	3	3	13	16	633	193	1330	-
	6	3	3	16	17	226	217	1426	-
Said	4	2	2	18	15	197	237	-	3925
	6	3	2	21	19	391	363	-	-
spa	4	1	0	2	2	11	17	3584	1033
	6	1	1	3	3	16	20	-	-

Table 4: Average solution time (for instances solved to optimality) using each compact MIP formulation.