A Branch-Price-and-Cut algorithm for the Kidney Exchange Problem

Matteo Petris, Claudia Archetti, Diego Cattaruzza, Maxime Ogier, Frédéric Semet

A Branch-Price-and-Cut algorithm for the Kidney Exchange Problem

Matteo Petris[a,*], Claudia Archetti[b], Diego Cattaruzza[c], Maxime Ogier[c], Frédéric Semet[c]

[a] *Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France*
[b] *Department of Information Systems, Decision Sciences and Statistics, ESSEC Business School, Cergy-Pontoise, France*
[c] *Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France*

Abstract

In this paper, we study a Kidney Exchange Problem (KEP) with altruistic donors and incompatible patient-donor pairs. Kidney exchanges can be modelled in a directed graph as circuits starting and ending in an incompatible pair or as paths starting at an altruistic donor. For medical reasons, circuits and paths are of limited length and are associated with a medical benefit to perform the transplants. The aim of the KEP is to determine a set of disjoint kidney exchanges of maximal medical benefit or maximal cardinality.

We consider a set packing formulation for the KEP with exponentially-many variables associated with circuits and paths, and develop a Branch-Price-and-Cut algorithm (BPC) to solve it. We decompose the pricing problem into a subproblem to price out the path variables and several subproblems to price out the circuit variables. The former can be formulated as an Elementary Longest Path Problem with Length Constraints (ELPPLC), that is NP-hard in the strong sense, and solved by a label correcting dynamic programming algorithm. Conversely, the latter can be formulated as a Longest Path Problem with Length Constraints (LPPLC) and solved in polynomial time. We strengthen the linear relaxation via the inclusion of a family of non-robust inequalities, namely, the subset-row inequalities. We separate them heuristically by looking for violated clique and odd-hole inequalities which are then transformed into subset-row inequalities.

We perform extensive computational experiments to assess the performances of the BPC algorithm on three sets of instances from the literature. On the easiest instances, the BPC algorithm yields comparable results with the literature; while on the other two sets it clearly outperforms the previous results. Hence, contrary to the existing literature, the BPC algorithm is the only exact approach able to effectively solve various and difficult instances with both objective functions and long chains and cycles.

Keywords: `Kidney exchange, altruistic donors, elementary paths, elementary circuits, Branch-Price-and-Cut.`

*Corresponding author
Email addresses: `petris@essec.edu` (Matteo Petris), `archetti@essec.edu` (Claudia Archetti), `diego.cattaruzza@centralelille.fr` (Diego Cattaruzza), `maxime.ogier@centralelille.fr` (Maxime Ogier), `frederic.semet@centralelille.fr` (Frédéric Semet)

1. Introduction

Kidneys are essential organs for the survival of the human body: (i) they produce the urine by filtering the blood to expel wastes and toxins; (ii) they play a role in maintaining the homeostasis of the body by regulating the acid-base balance, and the concentrations of electrolyte; (iii) they secrete several hormones responsible, for example, of the maturation of the red blood cells or of regulating the blood pressure. Unfortunately, kidneys may suffer from chronic diseases or failures which prevent them to perform their usual tasks. According to Kovesdy (2022), more than 10% of the worldwide population was affected by such diseases in 2022.

Chronic kidney diseases or failures are often treated by dialysis or transplantation (Levey and Coresh, 2012). Transplantation is the most preferable treatment: it affects less the quality of life of the patients, it offers a longer expectancy of life and it is more cost efficient (see, e.g., Yoo et al., 2016; Axelrod et al., 2018). However, such operations have to be performed with extreme care in order to minimise the risk of rejection. For this reason, several medical requirements must be met for a patient and a donor to be considered eligible for a transplant. Precisely, patient and donor must be *compatible* according to several indicators such as blood type and presence of specific antibodies (Kälble et al., 2005). In addition, when different patients and donors are compatible between each other, each possible transplant is assigned with a *medical benefit* which quantifies the medical interest of performing the transplant.

Kidneys are harvested from either deceased or living donors. Usually, the former case constitutes the majority of the transplants. In 2021, more than the 60% of the 92 532 worldwide transplants involved a deceased donor[1]. However, patients must register to a waiting list and might wait several years before (possibly) receiving a kidney from a compatible deceased donor (Lentine et al., 2023). Transplants from living donors may help patients in receiving a kidney more promptly (Nemati et al., 2014). Such transplants are possible because one functioning kidney is enough for a human being to conduct a healthy life. The organisation of transplants from living donors is more complex than the one from deceased donors. Indeed, patients need to find a willing and compatible donor in their circle of acquaintances, that is often not easy. Most of the times, a patient is capable of finding a willing donor with whom it is unfortunately not compatible. However, together, they can form an *incompatible patient-donor pair* and join a *kidney exchange programme* (see Rapaport (1986) and Roth et al. (2004)). Such programmes aim to determine the best (in terms of medical benefit) set of kidney exchanges in a pool of incompatible patient-donor pairs. An *exchange* takes place when a donor of a pair gives a kidney to a patient of another pair with whom he/she is compatible. For obvious reasons, in this context, a donor is willing to be in an exchange only if his/her associated patient is guaranteed to receive a kidney as well. Hence, the set of exchanges to be determined must give rise to *cycles of donations*. The transplants associated with a cycle of donation are usually performed simultaneously to avoid the risk of donor withdrawals. Hence, for practical reasons (e.g., limited facilities or medical staff),

[1] `https://www.transplant-observatory.org/`

regulations impose a maximal number of pairs in a cycle, usually set up to four (Biró et al., 2019). However, we report the breakthrough cases of cycles involving five, six or seven pairs in the Czech-Austrian kidney exchange programme (Viklicky et al., 2020). In recent years, *altruistic donors*, i.e., donors who are not paired with any patient, may join the pool of a kidney exchange programme (Roodnat et al., 2010). Thanks to their presence, the set of exchanges may also include *chains of donations* (domino donations) where the altruistic donors are at the start of the chains. In a chain of donation, surgeries are not required to be performed simultaneously, hence, the number of incompatible pairs involved in them, although usually limited by regulations, is larger than the one of the cycles[2].

Typically, kidney exchange programmes are run regularly (e.g., each one, three, or four months depending on the country regulation (Biró et al., 2021)) within one or more hospitals belonging to a single or different countries. At each round, an optimisation algorithm is called to solve the *Kidney Exchange Problem* (KEP) whose aim is to determine a set of cycles and chains of donations in the current pool of incompatible patient-donor pairs and altruistic donors such that the overall medical benefit is maximised.

Recently, Riascos-Álvarez et al. (2023) and Arslan et al. (2024) proposed two *Branch-and-Price* (BP) algorithms to solve the same problem. Both works employ an extended formulation based on the *Extended Edge Formulation* introduced in Constantino et al. (2013). The exponentially-many variables correspond to cycles and chains. Hence, in the column generation procedure, the pricing problem is decomposed in several subproblems to price out variables associated with cycles or chains. All subproblems can be formulated as an *Elementary Longest Path Problem with Length Constraints* (ELPPLC) which is strongly NP-hard. However, the pricing subproblems associated with cycles can be solved in polynomial time. Indeed, the elementarity constraints can be dropped as they can be formulated as a *Longest Path Problem with Length Constraints* (LPPLC).

In our work, we propose an exact approach to efficiently solve the KEP with both long cycles and chains and where the objective function is either the maximization of the medical benefit or the maximization of the number of transplants (which corresponds to the case where each transplant is associated with a value of the medical benefit equal to one). Precisely, we propose a *Branch-Price-and-Cut* (BPC) algorithm based on the *cycle formulation* for the KEP introduced by Roth et al. (2007) where the exponentially-many variables correspond to cycles and chains. We exploit the theoretical results of Arslan et al. (2024) to solve as few as possible ELPPLCs. However, instead of calling an integer program to solve the ELPPLC in the case of the chains, as done by Arslan et al. (2024), we make use of a labelling algorithm. Labelling algorithms are known to be more efficient than integer programs to solve ELPPLCs, and the set of resources to consider can be extended to manage the dual variables of non-robust inequalities. In this respect, we strengthen the linear relaxation of the formulation by including non-robust inequalities, namely the *subset-row* inequalities. The separation of these inequalities is an NP-complete problem Jepsen et al. (2008). Usually, in the literature,

[2] `https://www.uwhealth.org/news/uw-health-led-nation-in-paired-kidney-exchanges-in-2020`

only small cardinality subset-row inequalities are separated heuristically by enumeration. However, given the size of the KEP instances, adopting such procedure becomes intractable in terms of computational time. Hence, we propose two fast heuristics to separate them. These heuristics are based on the separation of the *clique* and *odd-hole* inequalities, respectively. We also propose two original procedures to transform those inequalities into subset-row inequalities. In addition, we prove that each violated odd-hole inequality leads to a subset-row inequality with the same violation.

We identified three different sets of instances for the KEP in the literature, each of them with different characteristics: (i) the PrefLib dataset with small cycles and chains and maximisation of the number of transplants (medical benefit set to value one) where Arslan et al. (2024) report the state-of-art results, (ii) the set of instances introduced by Pansart et al. (2022) with long chains, and (iii) the set of instances introduced by Delorme et al. (2023) with long cycles only. The more challenging instances are the ones considering either long chains or the maximisation of the medical benefit (i.e. not all values set to one). Contrary to the solution approaches in the literature, our approach has the advantage of being able to efficiently solve all these sets of instances, and in particular the most difficult ones. Indeed, the solution approach of Arslan et al. (2024) struggles on instances with long chains and maximisation of the medical benefit as it will be discussed in Section 5.3.2. The approach of Pansart et al. (2022) is a restricted master heuristic that provides lower and upper bounds by solving the root node of the branch-and-bound tree, but it does not perform branching. As a consequence, their approach is not able to prove optimality when branching is required, that is especially on the largest instances. Finally, the results reported by Delorme et al. (2023) show that their approach already struggles on instances with maximal length of the cycles equal to six and the maximisation of the number of transplants, thus preventing it to efficiently scale on instances with long cycles/chains or the maximisation of the medical benefit.

The reminder of the paper is organised as follows. In Section 2, we review the existing literature regarding the KEP. In Section 3, we formally introduce the KEP and we present a set packing formulation for the problem. The main components of our Branch-Price-and-Cut algorithm are described in Section 4. Finally, in Section 5, we report and analyse the results obtained by the BPC algorithm on three sets of benchmark instances.

2. Literature review

In this section, we survey the existing literature on solution approaches for the Kidney Exchange Problem (KEP).

Roth et al. (2005) study the KEP with cycles of length two only. Such variant of the KEP can be reduced to a *Maximum Weighted Matching Problem* and thus solved in polynomial time by, e.g., the Edmond's algorithm (Edmonds, 1965). However, the KEP with cycles of length larger than two (and chains of arbitrary length) is proven to be NP-hard (Roth et al., 2007; Abraham et al., 2007). For this reason, several formulations and solution methods have been proposed through the years.

In the following we review the main formulations proposed for the KEP with cycles only. Roth et al. (2007) introduced the *edge formulation* for the KEP, a first formulation with an exponential number of constraints and a polynomial number of variables w.r.t. the size of the instance. Whereas the first extended formulation, the *cycle formulation*, was introduced by Roth et al. (2007) and Abraham et al. (2007). The authors proved that the cycle formulation dominates the edge formulation in terms of linear relaxation. As a solution method, they considered a column generation approach where the pricing problem is solved by enumeration. Constantino et al. (2013) proposed the first compact formulation for the KEP with cycles only, the *extended edge formulation*. Klimentova et al. (2014) presented the *disaggregated cycle formulation*, a variant of the cycle formulation where the set of cycles is decomposed in subsets, one per each patient-donor pair, containing the cycles starting at that pair. Both Constantino et al. (2013) and Klimentova et al. (2014) adopt an idea to break symmetries: a cycle starting at pair i does not contain pairs with an index lower than i.

In addition, Dickerson et al. (2016) introduced the *position-indexed edge formulation*, a compact formulation which achieves linear relaxation bounds equivalent to the ones of the cycle formulation. More recently, Delorme et al. (2023) exploited the idea of representing a cycle by two compatible halves and introduced the *half-cycle formulation* to solve the KEP with cycles only and where the objective is to maximise the number of transplants. Such formulation has less variables than the cycle formulation while keeping the same quality of the linear relaxation bound. The authors proposed to solve the formulation by enumerating all the possible half-cycles and by applying a destructive bound procedure coupled with a variable-fixing strategy. Precisely, first, an upper bound on the maximum number of transplants is obtained by solving the linear relaxation of the formulation. Then, an iterative procedure starts: the half-cycle formulation is solved by a commercial solver by imposing that the solution value is equal to the upper bound; if an integer solution is found the procedure stops, otherwise the upper bound is decreased by one and the procedure repeats. They tested instances with up to 1000 incompatible pairs and cycles of length up to eight. All these formulations can be easily adapted to solve the KEP with altruistic donors and thus with chains. For example, Pansart et al. (2022) extended the cycle formulation to include the chains, as well. The interested reader may refer to Mak-Hau (2015) for a survey on formulations for the KEP up to 2015.

For scalability reasons, several works focus on developing column generation approaches for the KEP. As already mentioned, Abraham et al. (2007) presented the first Branch-and-Price (BP) algorithm to solve the KEP with cycles only. The works of Glorie et al. (2012), Glorie et al. (2014), Klimentova et al. (2014) and Plaut et al. (2016a) developed BP algorithms to solve the KEP with both cycles and chains. However, Plaut et al. (2016b) proved that the algorithms of Glorie et al. (2012), Glorie et al. (2014) and Plaut et al. (2016a) were not correct. In addition, Klimentova et al. (2014) did not provide any computational experiments with KEP instances with chains. Based on the work of Klimentova et al. (2014), Lam and Mak-Hau (2020) built the first BPC algorithm for the KEP with cycles only. They tested the BPC algorithm on the 80 instances of the PrefLib dataset with 16 to 2048 pairs and without altruistic donors. The length

of the cycles was limited to three or four. The BPC algorithm was able to solve all the instances when the length is limited to three and the majority of them when the length is limited to four. However, the considered valid inequalities are rather ineffective in strengthening the linear relaxation. Pansart et al. (2022) is the first column generation approach capable of dealing with both cycles and chains. The objective is to maximise the total medical benefit. The authors proposed a restricted master heuristic: first, the linear relaxation of the extended formulation is solved by column generation, then the formulation is restricted to the subset of variables found so far and solved with a commercial solver. In their approach the cycles are enumerated and the chains are detected via the solution of the pricing problem, namely the ELPPLC. Precisely, they proposed new heuristic algorithms to solve the pricing problem. The correctness of their algorithm is based on the solution of the ELPPLC via an exact dynamic programming algorithm. Pansart et al. (2022) presented results on realistic instances with up to 1000 incompatible pairs, 111 altruistic donors, cycles of length up to three and chains with length up to 12. Precisely, their approach solved to optimality the majority of the instances with up to 250 pairs and shows its limits on the larger and most challenging instances which are left with an optimality gap.

The first work proposing a column generation approach which prices both variables associated with cycles and chains is Riascos-Álvarez et al. (2023). The authors developed a BP algorithm based on the disaggregated cycle formulation of Klimentova et al. (2014) where the element of novelty is solving the pricing problem with a three-steps approach. Precisely, the pricing subproblems associated with the cycles and chains are formulated as dynamic programs and Mixed Integer Programs (MIPs). The first step of the pricing problem solves restricted versions of the dynamic programs with the aim of quickly finding positive reduced cost cycles or chains. If the first step fails, the second step solves the ELPPLC to price out the chains via the solution of a MIP model with dynamic inclusion of cuts for subtours elimination constraints. If the second step fails, the third step solves the pricing for cycles via enumeration until a certain time limit, and then via the solution of a MIP formulation for the LPPLC. In addition, the authors exploited the idea of Constantino et al. (2013) to represent each cycle by its vertex with the lowest index, that permits to apply a preprocessing procedure which reduces the subset of cycles to consider in the formulation of Klimentova et al. (2014). Consequently, the number of the pricing subproblems associated with the cycles is heavily reduced. Their approach is tested on the instances of the PrefLib dataset with cycles and chains of maximum length equal to three or four and chains of maximum length from three to six. The objective is the maximisation of the number of transplants. The large majority of the instances are solved to optimality at the root node. However, their results are outperformed by the BP algorithm of Arslan et al. (2024). Precisely, the BP algorithm of Arslan et al. (2024) builds on the one of Riascos-Álvarez et al. (2023) by identifying theoretical conditions where also the pricing subproblems associated with the chain variables can be solved in polynomial time. Precisely, if the maximum length of the chains is less than or equal to the maximum length of the cycles plus one, proving that no more positive reduced cost chains exist can be done by relaxing the elementarity constraints. The authors model the pricing of both cycles and non-elementary chains as a LPPLC, that is solvable in polynomial

time by a variant of the Bellman-Ford algorithm. However, to ensure the correctness of the algorithm, it might be necessary to solve an ELPPLC to prove that no more elementary positive reduced cost chain exist. Arslan et al. (2024) do so by modelling the ELPPLC as an integer program with an exponential number of constraints and propose to solve it by means of a branch-and-cut algorithm. As Riascos-Álvarez et al. (2023), they tested their approach on the instances of the PrefLib dataset. All the instances are solved to optimality in less than 20 seconds and the integer program to solve the ELPPLC is almost never required.

The BPC algorithm we propose in this work exploits the preprocessing of Riascos-Álvarez et al. (2023) to reduce the number of the pricing subproblems associated with the cycles and the interesting properties of Arslan et al. (2024) to solve the pricing subproblems associated with the chains in polynomial time under some conditions. The contribution of our method are the inclusion of valid inequalities to strengthen the formulation and the use of a labelling algorithm to efficiently solve the ELPPLC in opposition to the integer program used by Arslan et al. (2024).

Our approach is then able to solve more difficult instances with long chains and where the objective is the maximisation of the total medial benefit. Thus, it can be considered as a generic approach capable of handling all instances and problem settings presented in the former literature.

3. Problem formulation

Let $\mathcal{I}$ be the set of incompatible patient-donor pairs and let $\mathcal{D}$ be the set of altruistic donors. The KEP can be defined on a directed weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ referred to as *compatibility graph*. Vertex set $\mathcal{V} = \mathcal{I} \cup \mathcal{D}$ contains a vertex for each incompatible patient-donor pair of $\mathcal{I}$ and each altruistic donor of $\mathcal{D}$. The arcs in $\mathcal{A}$ model all the possible transplants between donors and patients. Hence, arc set $\mathcal{A}$ contains an arc (i, j) from vertex $i \in \mathcal{V}$ to patient-donor pair $j \in \mathcal{I}$, if the kidney of the donor associated with i is compatible with the patient of pair j. We assign a weight W_{ij} to each arc $(i, j) \in \mathcal{A}$ representing the utility (medical benefit) of the associated transplant. Cycles and chains of kidney exchanges between donors and patients are modelled in graph $\mathcal{G}$ by two types of subgraphs, namely circuits and paths. The length of a cycle or a chain is equal to the number of arcs in the circuit or path. In this respect, we call an *exchange circuit* an elementary circuit in graph $\mathcal{G}$ of length at most $L^C > 1$. Similarly, we call an *exchange path* an elementary path in graph $\mathcal{G}$ of length at most $L^P > 1$, starting with a vertex in $\mathcal{D}$. Given that vertices associated with altruistic donors do not have in-going arcs, exchange circuits are composed only of vertices associated with patient-donor pairs. Given an exchange circuit or path e, the weight W_e of e is defined as the sum of the weights of the arcs traversed by e, i.e., $W_e \coloneqq \sum_{(i,j)\in\mathcal{A}(e)} W_{ij}$, where $\mathcal{A}(e)$ is the set of arcs traversed by e. Finally, we define an *exchange scheme* as a union of pairwise vertex-disjoint exchanges circuits and paths. The KEP aims to determine an exchange scheme of maximum weight, where the weight of an exchange scheme is the sum of the weights of the exchange circuits and paths composing it. An exchange scheme may not contain all the vertices of $\mathcal{V}$. Remark that if all weights W_{ij}, $(i, j) \in \mathcal{A}$, are set to one, the objective of the KEP is to maximise the number of transplants. In what follows, with abuse of language, we refer to an

exchange circuit or path with the term exchange.

Example 1. *We consider the following toy instance of the KEP. The compatibility graph* $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ *is shown in Figure 1. Vertex set* $\mathcal{V} = \mathcal{I} \cup \mathcal{D}$ *is composed of two altruistic donors* $\mathcal{D} = \{1, 2\}$ *and seven patient-donor pairs* $\mathcal{I} = \{3, 4, 5, 6, 7, 8, 9\}$*. The weights on the arcs are reported in the figure. The maximal length of the circuits and paths are set to* $L^C = 3$ *and* $L^P = 4$*, respectively. The optimal value of the KEP on this toy instance is* 335 *and is attained with an optimal solution composed of two paths,* $(1, 3, 5)$ *and* $(2, 8, 4)$*, and one cycle* $(6, 7, 6)$*. Note that the maximisation of the medical benefit favours quality over quantity, indeed, patient-donor pair* 9 *is not served even if it exists a feasible solution that would serve all patient-donor pairs.*

Conversely, if the weights on the arcs are all set to one, the optimal value is seven and an optimal solution to attain it is: $(1, 3)$*,* $(2, 8, 9, 6)$ *and* $(4, 5, 7, 4)$*. In this case, all the patient-donor pairs are served.*

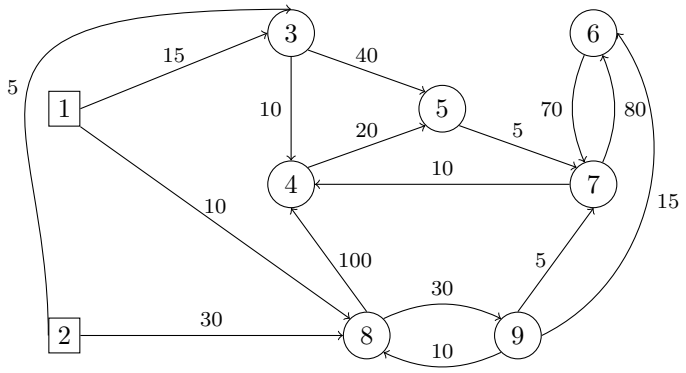

Figure 1: Compatibility graph of the toy instance of the KEP defined in Example 1.

We now report the set packing formulation proposed in Pansart et al. (2022) to model the KEP.

We denote by $\mathcal{E} = \mathcal{E}^C \cup \mathcal{E}^P$ the set of the exchanges in graph $\mathcal{G}$, where $\mathcal{E}^C$ ($\mathcal{E}^P$) denotes the set of the exchange circuits (paths) in graph $\mathcal{G}$. Let a_i^e be a binary parameter equal to one if vertex $i \in \mathcal{V}$ is involved in exchange $e \in \mathcal{E}$ and zero otherwise. For each exchange $e \in \mathcal{E}$, we define a binary variable λ_e taking value one if exchange e is part of the exchange scheme and zero otherwise.

The Set Packing formulation [SP] for the KEP reads as follows:

$$\text{[SP]} \quad \max \sum_{e \in \mathcal{E}} W_e \lambda_e \tag{1}$$

$$\text{s.t.} \sum_{e \in \mathcal{E}} a_i^e \lambda_e \leq 1 \qquad \forall i \in \mathcal{V} \tag{2}$$

$$\sum_{e \in \mathcal{E}} \lambda_e \leq \frac{|\mathcal{V}|}{2} \tag{3}$$

$$\lambda_e \in \{0, 1\} \qquad \forall e \in \mathcal{E}. \tag{4}$$

Objective function (1) maximises the weight of the exchange scheme. Constraints (2) referred to as *packing constraints* ensure that each incompatible pair and altruistic donor is involved in at most one

exchange circuit or path. Constraint (3) imposes an upper bound on the number of the exchanges in an exchange scheme. Remark that such constraint is redundant, indeed, each exchange circuit or path cannot be composed of less than two vertices. We include it in the formulation to compute a valid upper bound in the column generation procedure, namely the Lagrangian bound. Such bound may be used as a termination condition in the column generation procedure (see Desrosiers and Lübbecke, 2005, for more details). Finally, Constraints (4) define variables λ_e as binary.

Valid inequalities to strengthen formulation [SP] are presented in Section 4.4.

4. A Branch-Price-and-Cut algorithm

Formulation [SP] is an integer model defined over exponentially-many variables λ_e, $e \in \mathcal{E}$. We solve [SP] by means of an exact algorithm based on the Branch-Price-and-Cut (BPC) paradigm (Barnhart et al., 1998). BPC algorithms are variants of the branch-and-bound algorithm where, at each node of the branch-and-bound tree, the linear relaxation of [SP] is solved via a column generation procedure (Desrosiers and Lübbecke, 2005). The linear relaxation of [SP] is commonly referred to as the Master Problem (MP). If the solution of the MP is fractional, a separation procedure may be called to identify violated valid inequalities (see Section 4.4). If such inequalities are found, they are included in the MP and the column generation procedure is repeated. Finally, when no valid inequalities are detected, branching rules are applied to ensure the correctness of the BPC algorithm.

The outline of this section is as follows. Section 4.1 is devoted to the column generation procedure we employ in our BPC algorithm. In Sections 4.2 and 4.3, we formulate the pricing problem and we present an exact procedure to solve it. In Section 4.4, we present the family of valid inequalities and the cut generation strategy. Finally, the branching strategies are discussed in Section 4.5.

4.1. Column generation

The MP is solved by means of a column generation procedure at each node of the branch-and-bound tree. The Restricted Master Problem (RMP) denotes the MP restricted to a subset of the λ variables. Then, each iteration of the procedure comprises two consecutive steps: (i) solve the RMP to get the dual values associated with the constraints, and (ii) solve the so-called *pricing problem* to detect the most positive reduced cost variables (columns) that are added to the RMP. Note that the pricing problem searches for positive reduced cost variables because the RMP is a maximisation problem. When the pricing problem proves that no positive reduced cost variables exist, the current solution of the RMP is then proven to be optimal for the MP, as well. Note that if the pricing problem reports some positive reduced cost variables without proving they are the most positive, the column generation procedure is still valid.

Let $\mathcal{E}' \subseteq \mathcal{E}$, the RMP associated with $\mathcal{E}'$ reads as:

$$[\text{RMP}(\mathcal{E}')] \ \max \sum_{e \in \mathcal{E}'} W_e \lambda_e \tag{5}$$

$$\text{s.t.} \ \sum_{e \in \mathcal{E}'} a_i^e \lambda_e \leq 1 \qquad \forall i \in \mathcal{V} \tag{6}$$

$$\sum_{e \in \mathcal{E}'} \lambda_e \leq \frac{|\mathcal{V}|}{2} \tag{7}$$

$$\lambda_e \geq 0 \qquad \forall e \in \mathcal{E}'. \tag{8}$$

4.2. Pricing problem formulation

Let $\pi_i \geq 0$, $i \in \mathcal{V}$ and $\beta \geq 0$ be the dual prices associated with Constraints (6) and (7), respectively. The reduced cost of a λ_e variable is:

$$\bar{W}_e = W_e - \sum_{i \in \mathcal{V}} a_i^e \pi_i - \beta.$$

The pricing problem reads as follows:

$$[\text{PP}] \ \max\{\bar{W}_e : e \in \mathcal{E}\}.$$

We observe that exchange set $\mathcal{E}$ can be partitioned as:

$$\mathcal{E} = \mathcal{E}^C \cup \mathcal{E}^P = \bigcup_{i \in \mathcal{I}} \mathcal{E}_i^C \cup \mathcal{E}^P,$$

where $\mathcal{E}_i^C$ is the set of the exchange circuits starting and ending at incompatible pair $i \in \mathcal{I}$. Hence, pricing problem [PP] can be decomposed in $|\mathcal{I}| + 1$ independent subproblems:

$$[\text{PP-C}](i) \ \max\{\bar{W}_e : e \in \mathcal{E}_i^C\}, \qquad i \in \mathcal{I}$$

$$[\text{PP-P}] \ \max\{\bar{W}_e : e \in \mathcal{E}^P\}.$$

The aim of problem [PP-C](i) is to determine the most positive reduced cost variable λ_e, $e \in \mathcal{E}_i^C$, associated with exchange circuits starting and ending in $i \in \mathcal{I}$ or state that no positive reduced cost variable exists. Similarly, the aim of problem [PP-P] is to determine the most positive reduced cost exchange path variable λ_e, $e \in \mathcal{E}^P$ or state that no positive reduced cost variable exists.

In the following, we show that problems [PP-C](i), $i \in \mathcal{I}$, and [PP-P] share the same structure: they can be formulated as an *Elementary Longest Path Problem with Length Constraint* (ELPPLC), i.e., a variant of the *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC).

First, we formally define the ELPPLC having as a reference Pansart et al. (2022). Let $\mathcal{H} = (\mathcal{N}, \mathcal{F})$ be a directed weighted graph, where the weights associated with the arcs $(i, j) \in \mathcal{F}$ are denoted by $\bar{W}_{ij}$. The objective of the ELPPLC is to find an elementary path e^* in graph $\mathcal{H}$ starting at a given vertex $s \in \mathcal{N}$, ending at a given vertex $t \in \mathcal{N}$ and such that its weight $\bar{W}_{e^*} = \sum_{(i,j) \in \mathcal{A}(e^*)} \bar{W}_{ij}$ is maximum and its length is less than or equal to $\bar{L} > 0$. The ELPPLC is NP-hard in the strong sense (see Dror (1994)).

Now, we describe the topology of the graph $\mathcal{H}$ arising when pricing circuits and paths.

Let $i \in \mathcal{I}$ be an incompatible patient-donor pair. To model [PP-C](i) as an ELPPLC, we define a directed weighted graph $\mathcal{H}_i^C = (\mathcal{N}_i^C, \mathcal{F}_i^C)$ as follows. The vertices of $\mathcal{H}_i^C$ are the incompatible pairs of $\mathcal{I}$ plus a copy i' of pair i, i.e., $\mathcal{N}_i^C \coloneqq \mathcal{I} \cup \{i'\}$. The arcs set of $\mathcal{H}_i^C$ is $\mathcal{F}_i^C = \{(h,j) \in \mathcal{A} : h, j \in \mathcal{I}, j \neq i\} \cup \{(j,i') : (j,i) \in \mathcal{A}\}$ that is $\mathcal{F}_i^C$ contains the arcs of compatibility graph $\mathcal{G}$ between the incompatible pairs, except those entering in pair i and for all arcs $(j,i) \in \mathcal{A}$ entering in i, we add an arc (j,i') entering in the copy i' of i.

Remark that circuits in graph $\mathcal{G}$ starting and ending in i correspond to paths in graph $\mathcal{H}_i^C$ starting in i and ending in i'. Weights $\bar{W}_{hj}$ assigned to arcs $(h,j) \in \mathcal{F}$ are defined as follows

$$\bar{W}_{hj} = \begin{cases} W_{ij} - \pi_j - \beta, & \text{if } h = i; \\ W_{hi} - \pi_i, & \text{if } j = i'; \\ W_{hj} - \pi_j, & \text{otherwise.} \end{cases} \tag{9}$$

To consider the path length, we define a resource consumption $L_{hj} = 1$ for each arc $(h,j) \in \mathcal{F}$. Solving problem [PP-C](i) consists in finding the most positive reduced cost elementary path starting at i and ending at i' in graph $\mathcal{H}_i^C$ such that the length of the path is less than or equal to L^C.

We model [PP-P] as an ELPPLC on graph $\mathcal{H}^P = (\mathcal{N}^P, \mathcal{F}^P)$ defined as compatibility graph $\mathcal{G}$ augmented with source vertex s and an arc from s to each altruistic donor of $\mathcal{D}$. Precisely, we set $\mathcal{N}^P \coloneqq \{s\} \cup \mathcal{V}$ and $\mathcal{F}^P = \mathcal{A} \cup \{(s,i) : i \in \mathcal{D}\}$. Given an arc $(i,j) \in \mathcal{F}^P$, its weight $\bar{W}_{ij}$ is defined as follows

$$\bar{W}_{ij} = \begin{cases} W_{ij} - \pi_j - \beta, & \text{if } i = s; \\ W_{ij} - \pi_j, & \text{otherwise.} \end{cases} \tag{10}$$

Resource consumption L_{ij} is set to be one on all arcs $(i,j) \in \mathcal{F}^P$. Problem [PP-P] consists in finding the most positive reduced cost elementary path starting at $s \in \mathcal{N}$ and ending in any vertex $j \in \mathcal{N}^P \setminus \{s\}$ in graph $\mathcal{H}^P$ such that the length of the path is less than or equal to $L^P + 1$. Remark that to be consistent with the definition of exchange paths given in Section 3, here, [PP-P] looks for paths of length at most $L^P + 1$. Indeed, in the underlying graph $\mathcal{H}^P$ used to solve [PP-P], we add a source vertex s which is not included in the compatibility graph $\mathcal{G}$.

In the following, we report two theoretical results which identify conditions under which subproblems [PP-C](i), $i \in \mathcal{I}$, and [PP-P] can be formulated as *Longest Path Problems with Length Constraint* (LPPLC), i.e., the elementarity constraint can be removed (see, e.g., Arslan et al., 2024). The interest of these two results stems from the following remark. The LPPLC is a variant of the *Shortest Path Problem with Resource Constraints* (SPPRC) where the unique resource is the length. The SPPRC is NP-hard (Di Puglia Pugliese and Guerriero, 2013) for a generic resource constraint and it can be solved on a graph $\mathcal{H} = (\mathcal{N}, \mathcal{F})$ by pseudo-polynomial algorithms with a temporal complexity of $O(\bar{R}|\mathcal{F}|)$ (Desrochers, 1988) where $\bar{R}$ is the maximal amount of resource. Conversely, the LPPLC can be solved in polynomial time with a temporal complexity of $O(|\mathcal{N}||\mathcal{F}|)$. Indeed, in the LPPLC, the maximal amount of resource $\bar{R}$ is the maximum length of the path $\bar{L}$, which is bounded from above by the number of vertices in the graph $|\mathcal{N}|$. In addition, being the LPPLC a relaxation of the ELPPLC, the correctness of the column generation procedure is ensured.

The first result states that the subproblems to price the circuits reduce to an LPPLC. Remark that a non-elementary circuit is composed of at least two elementary circuits. Hence, if the solution of the LPPLC returns a positive reduced cost non-elementary circuit when solving problem [PP-C](i), $i \in \mathcal{I}$, then at least one of the elementary circuits composing it must have a positive reduced cost. Remark that such circuit might not be a solution of [PP-C](i), but of a different subproblem to price circuits starting with another incompatible pair. Hence, formulating the pricing problems for the circuits as a LPPLC is enough to detect positive reduced cost elementary circuits, even if it may not be the most positive one.

The second result is an original result of Arslan et al. (2024) and it identifies a subclass of the problem instances where it is possible to solve problem [PP-P] by dropping the elementarity requirement of the paths, i.e., [PP-P] reduces to a LPPLC, as well. Precisely, if $L^P \leq L^C + 1$ and problems [PP-C](i), $i \in \mathcal{I}$, do not identify any circuit with a reduced cost larger than $-\beta$, then problem [PP-P] reduces to a LPPLC. Indeed, a non-elementary path of length l contains a sub-tour of length at most $l - 1$ which is composed of pairs only. Hence, if problems [PP-C](i), $i \in \mathcal{I}$, do not identify any circuit (of length at most L^C) with a reduced cost larger than $-\beta$, then [PP-P] will not provide any positive reduced cost path which is non-elementary. Note that the comparison is done with $-\beta$ and not zero since for the chains the β term is subtracted from the weight of the outgoing arcs from altruistic donors.

Remark that if $L^P > L^C + 1$, solving the LPPLC on graph $\mathcal{H}^P$ provides an upper bound on the value of [PP-P] since the LPPLC is a relaxation of the ELPPLC.

4.3. Pricing problem solution

In this section, we present the strategy we use to solve the pricing problem in the column generation procedure. The procedure is based on the one of Arslan et al. (2024) that makes use of the theoretical results presented in Section 4.2.

The pseudocode of such a procedure is presented in Algorithm 1. It can be noticed that an ELPPLC needs to be solved for [PP-P] only if: (i) no positive reduced cost circuits is detected, i.e, [PP-C](i) $W_i^* = \max\{\bar{W}_e : e \in \mathcal{E}_i^C\} \leq 0$, for all $i \in \mathcal{I}$; (ii) the LPPLC version of the [PP-P] provides no positive reduced cost elementary path, but provides a positive reduced cost non-elementary path; (iii) either the maximum length of the chains is strictly larger than the maximum length of the cycles plus one, i.e., $L^P > L^C + 1$, or in the other case no reduced cost circuit has a value greater than $-\beta$. Indeed, if the LPPLC do not find any positive reduced cost non-elementary path, then [PP-P] does not admit any positive reduced cost elementary path, as well. In addition, if some positive reduced cost elementary paths are found when solving the LPPLC, these can be included in the RMP without the need to solve [PP-P] with an ELPPLC. Finally, in the case where $L^P \leq L^C + 1$, as mentioned in Section 4.2, if the solution of the LPPLC did not report any circuit with a reduced cost larger than $-\beta$, then [PP-P] will not provide any positive reduced cost path that is non-elementary.

The procedure of Algorithm 1 solves the MP to optimality at each node of the branch-and-bound tree.

Algorithm 1: Column generation procedure.

Initialisation: Initialise the set of columns to the RMP.
do
  solve the RMP;
  get the optimal solution of the dual of the RMP;
  build pricing subproblems [PP-C](i), $i \in \mathcal{I}$ and [PP-P];
  solve [PP-C](i) and let $\bar{W}_i^*$ be its optimal value, for all $i \in \mathcal{I}$;
  let $\bar{W}^{*C} = \max\{\bar{W}_i^* : i \in \mathcal{I}\}$;
  if $\bar{W}^{*C} \leq 0$ **then**
    Solve the LPPLC on graph $\mathcal{H}^P$ and let $\bar{W}^{*P}$ be its optimal value;
    if $\bar{W}^{*P} > 0$ *and all the positive reduced cost paths returned by solving the LPPLC are non-elementary* **then**
      if *(*$L^P \leq L^C + 1$ *and* $\bar{W}^{*C} > -\beta$*) or (*$L^P > L^C + 1$*)* **then**
        solve the ELPPLC on graph $\mathcal{H}^P$ and let $\bar{W}^{*P}$ be its optimal value;
      end
    end
  end
  add positive reduced cost elementary circuits and paths to the RMP, if any is detected;
while $\max\{\bar{W}^{*C}, \bar{W}^{*P}\} > 0$;

In the following, we describe the algorithms to solve: (i) the LPPLC to provide solutions for subproblems [PP-C](i) and [PP-P], and (ii) the ELPPLC to provide optimal solutions for [PP-P].

We report a polynomial time procedure based on the Bellman-Ford algorithm to solve the LPPLC. Precisely, such algorithm corresponds to the first $\bar{L}$ iterations of the Bellman-Ford algorithm, and is detailed in Algorithm 2, where we make use of the generic notation introduced in Section 4.2. In addition, we denote by W_i^l the weight of a path of length $l = 0, \ldots, \bar{L}$ starting in s and ending in vertex $i \in \mathcal{N}$. Matrix $(p_i^l)_{l=0,\ldots,\bar{L}, i \in \mathcal{N}}$ stores the predecessor p_i^l of vertex $i \in \mathcal{N}$ in a path of length l starting in s and ending in vertex i. We write $\mathcal{P}$ for the set of the positive reduced cost paths found by the algorithm.

To complete this section, we describe the label correcting dynamic programming algorithm (Feillet et al., 2004) we propose to solve the ELPPLC. In the work of Arslan et al. (2024), the ELPPLC is tackled by a mixed integer programming formulation with an exponential number of constraints. Here, we chose a label correcting dynamic programming algorithm because of its efficiency and its flexibility when it comes to incorporate non-robust valid inequalities to strengthen the formulation (see Section 4.4).

In a label correcting algorithm, vertices $i \in \mathcal{N}$ are repeatedly processed and their associated labels (set $\mathcal{L}_i$), which identify paths in the graph, are extended. Precisely, a label $l = (L, \bar{W}, \mathcal{U}, i) \in \mathcal{L}_i$ represents a path in graph $\mathcal{H}$ starting at vertex s and ending at vertex $i \in \mathcal{N}$ characterised by its accumulated weight $\bar{W}$, its length L and the subset of visited vertices $\mathcal{U} \subseteq \mathcal{N}$. A label $l = (L, \bar{W}, \mathcal{U}, i)$ associated with vertex i can be extended along arc $(i, j) \in \mathcal{F}$ if the length constraint ($L < \bar{L}$) and the elementarity constraint ($j \notin \mathcal{U}$) are respected. If this is the case, a new label $l' = (L + 1, \bar{W} + W_{ij}, \mathcal{U} \cup \{j\}, j)$ associated with vertex $j \in \mathcal{V}$ is obtained. Dominance rules are commonly applied in labelling algorithms to prune labels which would not lead to any optimal solution. Given two labels $l = (L, \bar{W}, \mathcal{U}, i)$ and $l' = (L', \bar{W}', \mathcal{U}', i)$ associated with the

Algorithm 2: A Bellman-Ford algorithm for the LPPLC

Input: Graph $\mathcal{H} = (\mathcal{N}, \mathcal{F})$.

Initialisation: $W_s^0 \coloneqq 0$ and $p_s^0 \coloneqq s$, $W_i^l \coloneqq -\infty$ and $p_i^l = null$ for all $i \in \mathcal{N} \setminus \{s\}$ and $l = 0, \ldots, \bar{L}$ and $\mathcal{P} = \emptyset$.

forall $l = 1, \ldots, \bar{L}$ **do**
  forall $i \in \mathcal{N}$ **do**
    forall $(i, j) \in \delta^+(i)$ **do**
      if $W_i^{l-1} + \bar{W}_{ij} > W_j^l$ **then**
        set $W_j^l \coloneqq W_i^{l-1} + \bar{W}_{ij}$;
        set $p_j^l = i$;
      end
    end
  end
end

// retrieve positive reduced cost paths

forall $l = 1, \ldots, \bar{L}$ **do**
  forall $i \in \mathcal{N}$ **do**
    if $W_i^l > 0$ **then**
      apply backtracking to p_i^l to build the path;
      add the path to $\mathcal{P}$;
    end
  end
end

return $\mathcal{P}$;

same vertex i, we say that l dominates l' if:

$$\begin{cases} L \leq L' \\ \bar{W} \geq \bar{W}' \\ \mathcal{U} \subseteq \mathcal{U}' \end{cases} \tag{11}$$

and one of the inequalities (inclusion) is strict. The labelling algorithm is presented in Algorithm 3.

The elementarity constraint makes the ELPPLC strongly NP-hard and slows down the solution algorithm. Adopting the *ng-path* relaxation (Baldacci et al., 2011), which partially relaxes the elementarity constraint of the paths, helps in accelerating the solution of the ELPPLC and, in practice, the paths found when solving this relaxation are usually elementary. Each vertex $i \in \mathcal{N}$ is assigned with a neighbourhood $\mathcal{U}_i$ of a given size. Circuits along a path are then allowed only if each vertex visited more than once does not belong to the neighbourhoods of its predecessors in the path. The label definition and extension rule are modified accordingly. Precisely, the set of visited customers $\mathcal{U}$ becomes the *memory* of the label. The extension of $l = (L, \bar{W}, \mathcal{U}, i)$ along arc (i, j) is $l' = (L + 1, \bar{W} + W_{ij}, (\mathcal{U} \cap \mathcal{U}_j) \cup \{j\}, j)$.

4.4. Cut generation

In order to improve the linear relaxation bound at each node of the branch-and-bound tree, we strengthen the formulation of the MP via the inclusion of subset-row (SR) inequalities (Jepsen et al., 2008). A common practice from the literature is to separate such inequalities by enumeration (see, e.g., Pecin et al., 2017).

Algorithm 3: A Labelling algorithm for the ELPPLC.

Input: Graph $\mathcal{H} = (\mathcal{N}, \mathcal{F})$.

Initialisation: $\mathcal{L}_j \coloneqq \emptyset$, for all $j \in \mathcal{N}$; $\mathcal{L}_s \coloneqq \{(0, 0, \emptyset, s)\}$; $termination \coloneqq false$.

```
while termination ≠ true do
    set termination := true;
    forall i ∈ N do
        forall l = (L, W̄, U, i) ∈ L_i s.t. L < L̄ do
            if l has not been extended yet then
                forall (i, j) ∈ F s.t. j ∉ U do
                    let l' = (L + 1, W̄ + W_ij, U ∪ {j}, j) be the extension of l to j;
                    if l' is not dominated by any label in L_j then
                        remove from L_j labels dominated by l';
                        add l' to L_j;
                        set termination := false;
                    end
                end
                mark l as extended;
            end
        end
    end
end
```

However, the characteristics of the KEP instances make this procedure prohibitive in terms of computational time. Indeed, on instances with thousands of incompatible pairs, one may end up with millions of subset-row inequalities even associated with only subsets of cardinality equal to three. Note that although these subsets can be determined a priori, they need to be inspected at each separation round to identify violated inequalities. To overcome this computational burden, instead of separating the SR inequalities directly, we propose two fast heuristics to separate them. Each heuristic is based on the separation of a different family of inequalities, namely the *clique* (Cl) and *odd-hole* (OH) inequalities (see Padberg, 1973), respectively. Fast procedures from the literature permit the detection of violated Cl and OH inequalities, which are then translated into SR inequalities. We show that each violated OH inequality leads to a SR inequality with the same violation. Finally, the obtained SR inequalities are inserted in the RMP. However, these inequalities are *non-robust*, that is, considering their dual variables in the pricing problem solution modifies the structure of the pricing problem itself. This generally increases the difficulty of solving the pricing problem. On the other side, such non-robust inequalities are worth to be used because of their great potential to reduce the integrality gaps.

Cl and OH inequalities are identified thanks to the so-called conflict graph. In the following, we introduce the notion of a conflict graph in the context of our problem. Given an exchange $e \in \mathcal{E}$, we denote by $\mathcal{V}(e)$ the set of vertices in exchange e. We say that two exchanges $e, e' \in \mathcal{E}$ are *in conflict* if they share common vertices, i.e., if $\mathcal{V}(e) \cap \mathcal{V}(e') \neq \emptyset$. Variables associated with exchanges in conflict cannot both take value one in a feasible solution of formulation [SP], because of Constraints (2). Given a fractional solution $\tilde{\lambda}$ of the MP, we define the so-called *conflict graph* $\mathcal{C} = (\mathcal{Q}, \mathcal{T})$ as follows. Node set $\mathcal{Q}$ contains nodes associated with

the exchanges $e \in \mathcal{E}$ whose corresponding variable has a fractional value, i.e., $\tilde{\lambda}_e \in]0,1[$. Edge set $\mathcal{T}$ contains an edge (e, e') if exchanges $e \in \mathcal{Q}$ and $e' \in \mathcal{Q}$ are in conflict. In the following, we say that an edge $(e, e') \in \mathcal{T}$ is *covered* by a vertex $v \in \mathcal{V}$ if $v \in \mathcal{V}(e) \cap \mathcal{V}(e')$.

In Section 4.4.1, we first introduce the subset-row inequalities and how they are managed in the pricing algorithm. Then, in Sections 4.4.2 and 4.4.3, we present clique and odd-hole inequalities, respectively. For each family of inequalities, we present the separation algorithm considered and the procedure to transform them into subset-row inequalities. Finally, the strategy used to separate the inequalities and add them to the MP is described in Section 4.4.4.

4.4.1. Subset-row inequalities

Given a subset $\mathcal{S} \subseteq \mathcal{V}$ and a multiplier p_i for each $i \in \mathcal{S}$, subset-row (SR) inequalities are obtained as a Chvátal-Gomory rounding of the Constraints (2) associated with elements in $\mathcal{S}$:

$$\sum_{e \in \mathcal{E}} \left\lfloor \sum_{i \in \mathcal{S}} p_i a_i^e \right\rfloor \lambda_e \leq \left\lfloor \sum_{i \in \mathcal{S}} p_i \right\rfloor, \qquad \mathcal{S} \subseteq \mathcal{V}. \tag{12}$$

We recall that we do not directly separate SR inequalities and in the following, we detail how to manage these non-robust inequalities in the pricing algorithm. As mentioned above, considering non-robust inequalities modifies the structure of the pricing problem. For the KEP, this translates into formulating all pricing problems [PP-C](i), $i \in \mathcal{I}$, and [PP-P] as an Elementary Longest Path Problem with Resource Constraints (ELPPRC). Indeed, each subset-row inequality enforces a resource. It is important to notice this implies that when considering such inequalities, it is no longer possible to solve any pricing problem in polynomial time. Hence, all pricing subproblems are solved via the labelling algorithm presented in Section 4.3 where the following modifications have to be applied. Let $\sigma_{\mathcal{S}} \geq 0$ be the dual variable associated with SR inequality (12) defined for subset $\mathcal{S} \subseteq \mathcal{V}$. To consider such dual variable in the ELPPRC, we follow the procedure proposed by Pecin et al. (2017). We choose to do so because Pecin et al. (2017) provide a general management rule of the SR inequalities which is valid also in the case where the elementarity of the paths is relaxed. In the label definition, we include an attribute for each SR inequality in the RMP whose associated dual variable is different from zero. The attribute of a SR inequality identified by $\mathcal{S} \subseteq \mathcal{V}$ is a parameter $M(\mathcal{S})$ which records the visits of a path to vertices in $\mathcal{S}$ and establishes when dual variable $\sigma_{\mathcal{S}}$ has to be discounted from the reduced cost of the path. Precisely, parameter $M(\mathcal{S})$ is initialised to 0 and every time a path visits a vertex $i \in \mathcal{S}$, $M(\mathcal{S})$ is incremented by $p_i = 1/2$. If $M(\mathcal{S}) \geq 1$, dual variable $\sigma_{\mathcal{S}}$ is discounted from the reduced cost and $M(\mathcal{S})$ is decremented by one unit.

Moreover, the new label attributes $M(\mathcal{S})$ have to be included also in the dominance rule (11). To do so, the second condition in dominance rule has to be replaced by:

$$\bar{W} \geq \bar{W}' + \sum_{\substack{\mathcal{S} \in \mathcal{M}: \\ M(\mathcal{S}) > M'(\mathcal{S})}} \sigma_{\mathcal{S}},$$

where $\mathcal{M}$ is the set of subsets $\mathcal{S} \subseteq \mathcal{V}$ representing SR inequalities in the RMP whose dual variable $\sigma_{\mathcal{S}}$ is different from zero.

4.4.2. Clique inequalities

Clique inequalities (Cl) correspond to cliques in conflict graph $\mathcal{C}$. A *clique* is a set of nodes $\mathcal{W} \subseteq \mathcal{Q}$ that induces a complete subgraph, i.e., such that $(e, e') \in \mathcal{T}$, for each $e, e' \in \mathcal{W}$, $e \neq e'$. The clique inequality associated with $\mathcal{W}$ is:

$$\sum_{e \in \mathcal{W}} \lambda_e \leq 1. \tag{13}$$

The separation of these inequalities is performed via the iterative heuristic algorithm proposed by Marzi et al. (2019). First, each node $e \in \mathcal{Q}$ is assigned with a weight equal to the number of its incident edges times $\tilde{\lambda}_e$. At each iteration, the algorithm selects a node $e \in \mathcal{Q}$ with the highest weight and builds a maximal weighed clique containing that node according to a greedy criterion. When the clique is maximal, its nodes are removed from the conflict graph and a new iteration starts. Finally, if such a clique has at least cardinality three it may give rise to a clique inequality.

Cl inequalities are defined over a subset of variables $\mathcal{W}$, hence, the management of the associated dual variable $\gamma_{\mathcal{W}} \geq 0$ when solving the pricing problem is not tractable. Indeed, when solving the pricing problem, the dual variable $\gamma_{\mathcal{W}}$ has to be discounted if the associated exchange is the same as one of the exchanges in set $\mathcal{W}$. When solving the ELPPRC with a labelling algorithm, it would require to compare, for each label, the associated partial path with the elements in set $\mathcal{W}$, that would be computationally very costly.

Now, we propose a heuristic procedure to retrieve a violated SR inequality (12) given a violated Cl inequality defined on a clique $\mathcal{W}$. We denote by $\mathcal{T}(\mathcal{W}) = \{(e, e') \in \mathcal{T} : e, e' \in \mathcal{W}, e \neq e'\}$ the subset of the edges whose incident nodes are in clique $\mathcal{W}$. The procedure first determines a subset of vertices $\mathcal{S} \subseteq \mathcal{V}$ such all the edges $(e, e') \in \mathcal{T}(\mathcal{W})$ are covered with at least one vertex $v \in \mathcal{S}$. Then, it defines multipliers p_i, $i \in \mathcal{S}$ to obtain a violated SR inequality.

Subset $\mathcal{S}$ is defined as follows. First, we determine the subset of candidate vertices $\mathcal{V}(\mathcal{W}) \subseteq \mathcal{V}$ that can cover the edges in $\mathcal{T}(\mathcal{W})$, i.e., $\mathcal{V}(\mathcal{W}) = \bigcup_{e,e' \in \mathcal{W}, e \neq e'} \mathcal{V}(e) \cap \mathcal{V}(e')$. The vertices of $v \in \mathcal{V}(\mathcal{W})$ are ordered by decreasing number of nodes $e \in \mathcal{W}$ such that $v \in \mathcal{V}(e)$, i.e., of exchanges in the clique which visit vertices v. Then, each vertex $v \in \mathcal{V}(\mathcal{W})$ is processed in such order and inserted in $\mathcal{S}$ if there exists an edge $(e, e') \in \mathcal{T}(\mathcal{W})$ that can be covered with v and has not been covered with any of the vertices already in $\mathcal{S}$. Such edges are then covered with vertex v.

Finally, the procedure determines multipliers p_i, $i \in \mathcal{S}$. First, if the cardinality of $\mathcal{S}$ is less or equal to five, we assign the multipliers proposed in Pecin et al. (2017): (i) if $|\mathcal{S}| = 3$, then $p_i = 1/2$, for all $i \in \mathcal{S}$; (ii) if $|\mathcal{S}| = 4$, then $p_i = 2/3$, for all $i \in \mathcal{S}$; (iii) if $|\mathcal{S}| = 5$, then $p_i = 1/2$, for all $i \in \mathcal{S}$. If with this choice of multipliers the subset-row is violated we stop. Otherwise, if it is not violated or the cardinality of $\mathcal{S}$ is strictly larger than five we determine the multipliers by solving the following mixed-integer program.

For each $e \in \mathcal{W}$, we introduce an integer non-negative variable y_e^{int} and a continuous variable $y_e^{frac} \in$

$[0, 0.99]$ to model the integer and fractional parts of coefficients of the λ variables in SR inequality (12). Similarly, we introduce an integer non-negative variable r^{int} and a continuous variable $r^{frac} \in [0, 0.99]$ to model the integer and fractional parts of the right hand-side of SR inequality (12). With abuse of notation, we denote the continuous non-negative variables modelling the multipliers by p_i, $i \in \mathcal{S}$. In addition, we denote by $\underline{m}$ and $\bar{m}$ two continuous non-negative variables to model the minimal and maximal values of the multipliers. Finally, we denote by $\alpha_1, \alpha_2 \in \mathbb{R}_{>0}$ two weights.

Given a fractional solution $\tilde{\lambda}$ and subset $\mathcal{S}$, we propose the following mixed-integer program to determine the multipliers:

$$[\mathrm{M}(\tilde{\lambda}, \mathcal{S})] \ \min \ \alpha_1 \left(r^{int} - \sum_{e \in \mathcal{W}} \tilde{\lambda}_e y_e^{int} \right) + \alpha_2 (\bar{m} - \underline{m}) \tag{14}$$

$$\text{s.t. } y_e^{int} = \sum_{i \in \mathcal{S}} a_i^e p_i - y_e^{frac} \qquad \forall e \in \mathcal{W} \tag{15}$$

$$r^{int} = \sum_{i \in \mathcal{S}} p_i - r^{frac} \tag{16}$$

$$\underline{m} \leq p_i \qquad \forall i \in \mathcal{S} \tag{17}$$

$$\bar{m} \geq p_i \qquad \forall i \in \mathcal{S} \tag{18}$$

$$r^{int} \in \mathbb{Z}_{\geq 0}, \qquad 0 \leq r^{frac} \leq 0.99 \tag{19}$$

$$y_e^{int} \in \mathbb{Z}_{\geq 0}, \qquad 0 \leq y_e^{frac} \leq 0.99 \qquad \forall e \in \mathcal{W} \tag{20}$$

$$0 \leq p_i \leq 1 \qquad \forall i \in \mathcal{S} \tag{21}$$

$$\underline{m}, \bar{m} \in \mathbb{R}_{\geq 0}. \tag{22}$$

Objective (14) minimises two terms: the first one is the negative of the violation of the SR inequality w.r.t. solution $\tilde{\lambda}$ and the second one is the distance between the minimal and the maximal value of the multipliers to avoid them receiving extreme values. We set weights α_1 and α_2 so that the two objectives are managed hierarchically: among the solutions which maximise the violation, we look for the one that minimises the distance between the minimal and maximal multiplier. Constraints (15) model the value of variable y_e^{int} for each $e \in \mathcal{W}$. Similarly, Constraint (16) models the value of variable r^{int}. Constraints (17) and (18) compute the minimal and maximal values of the multipliers, respectively. Finally, Constraints (19), (20), (21) and (22) define the domains of the variables of the model.

Such model is solved by a commercial solver, its solution time is negligible in practice. If the value of the first objective is positive, a violated SR inequality is found for set $\mathcal{S}$ with the values of multipliers variables p_i, $i \in \mathcal{S}$.

Example 2. *We make use of the toy instance of the KEP presented in Example 1. We consider fractional solution $\tilde{\lambda}$ with five exchanges e_1, e_2, e_3, e_4 and e_5 defined as in Table 1. The associated conflict graph $\mathcal{C} = (\mathcal{Q}, \mathcal{T})$ is shown in Figure 2 where on each edge we report the set of vertices of the compatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ that can cover that edge. It is easy to see that set of nodes $\mathcal{W} = \{e_1, e_2, e_3, e_4\}$ forms a clique*

in the conflict graph which gives rise to a violated clique inequality. Indeed, by looking at Table 1, we have $\tilde{\lambda}_{e_1} + \tilde{\lambda}_{e_2} + \tilde{\lambda}_{e_3} + \tilde{\lambda}_{e_4} = 1.33 > 1$*, thus the violation of the Cl inequality is equal to* 0.33*.*

Now, we retrieve the associated SR inequality, i.e., we need to define set $\mathcal{S}$ *and multipliers* p_i*,* $i \in \mathcal{S}$*. We first determine* $\mathcal{V}(\mathcal{W}) = \{1, 6, 7, 8, 9\}$ *such that the edges of* $\mathcal{T}(\mathcal{W}) = \{(e_1, e_2), (e_1, e_3), (e_1, e_4), (e_2, e_3), (e_2, e_4), (e_3, e_4)\}$ *can be covered.*

In Table 2, we propose an ordering of the vertices of $\mathcal{V}(\mathcal{W})$ *by decreasing number of exchanges of the clique that visit them. In order to define set* $\mathcal{S}$*, the first vertex to be processed is* 7 *and the edges of* $\mathcal{T}(\mathcal{W})$ *covered by it are* (e_2, e_3)*,* (e_2, e_4) *and* (e_3, e_4)*. We insert* 7 *in* $\mathcal{S}$*. The next vertex is* 8 *and the edges of* $\mathcal{T}(\mathcal{W})$ *covered by it are* (e_1, e_2)*,* (e_1, e_4) *and* (e_2, e_3)*. The first two were not covered previously, hence, we insert* 8 *in* $\mathcal{S}$*. The next vertex is* 1*, it covers only edge* (e_1, e_4) *which is already covered by* $8 \in \mathcal{S}$*, hence, it is not inserted in* $\mathcal{S}$*. The next vertex is* 6 *and the edge of* $\mathcal{T}(\mathcal{W})$ *covered by it is* (e_1, e_3) *which was not covered previously, hence, we insert* 6 *in* $\mathcal{S}$*. All the edges of* $\mathcal{T}(\mathcal{W})$ *are now covered, thus, there is no need to process other vertices of* $\mathcal{V}(\mathcal{W})$*. Then, we obtain set* $\mathcal{S} = \{7, 8, 6\}$*.*

It is easy to see that by assigning multipliers $p_1 = 1/2$*, for each* $i \in \mathcal{S}$*, we obtain a violated SR inequality with the same violation of the Cl inequality.*

Table 1: Fractional solution $\tilde{\lambda}$.

exchange	value in $\tilde{\lambda}$
$e_1 = (1, 8, 9, 6)$	0.33
$e_2 = (2, 8, 9, 7)$	0.33
$e_3 = (6, 7, 6)$	0.33
$e_4 = (1, 8, 4, 5, 7)$	0.33
$e_5 = (2, 3, 4, 5)$	0.66

Table 2: Ordering of the elements of $\mathcal{V}(\mathcal{W})$.

$\mathcal{V}(\mathcal{W})$	exchanges of $\mathcal{W}$
7	e_2, e_3, e_4
8	e_1, e_2, e_4
1	e_1, e_4
6	e_1, e_3
9	e_1, e_2

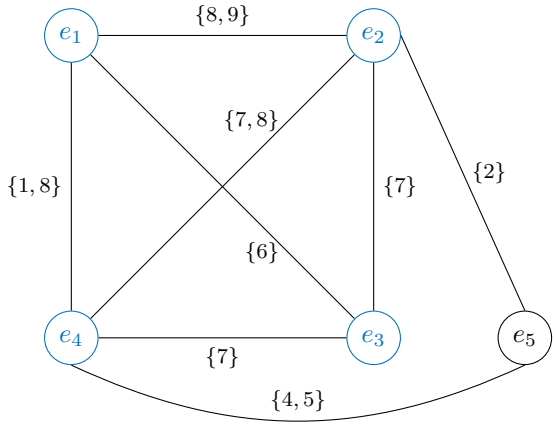

Figure 2: Conflict graph arising from the fractional solution of Table 1, where clique $\mathcal{W} = \{e_1, e_2, e_3, e_4\}$ is marked in blue.

4.4.3. Odd-hole inequalities

Odd-hole inequalities (OH) correspond to particular subgraphs in conflict graph $\mathcal{C}$, namely to odd cycles without chords. An *odd cycle without chords* is a set of nodes $\mathcal{P} = \{e_1, e_2, \ldots, e_{2k+1}\}$, $k \geq 2$, such that edge set $\mathcal{T}$ contains edges $t_i = (e_i, e_{i+1})$, $i = 1, \ldots, 2k$, and $t_{2k+1} = (e_{2k+1}, e_1)$ and no other edge incident on two nodes of $\mathcal{P}$. The odd-hole inequality associated with $\mathcal{P}$ is:

$$\sum_{e \in \mathcal{P}} \lambda_e \leq \frac{|\mathcal{P}| - 1}{2}. \tag{23}$$

The separation of these inequalities is done based on the exact procedure of Hoffman and Padberg (1993). Such procedure considers each vertex of conflict graph $\mathcal{C}$ and determines all the odd-hole inequalities associated with odd cycles without chords starting at the considered vertex. This procedure is rather time consuming. Therefore, as suggested by Hoffman and Padberg (1993), when determining the inequalities, we randomly consider only the 30% of the vertices of the conflict graph as starting vertex.

OH inequalities are defined over a subset of variables $\mathcal{P}$, hence, the management of the associated dual variable $\theta_{\mathcal{P}} \geq 0$ when solving the pricing problem is not tractable. The explanation is the same as the one for the Cl inequalities (see Section 4.4.2).

For this reason, we propose a procedure to retrieve a SR inequality from an OH inequality defined on cycle $\mathcal{P}$. Moreover such SR inequality is a lifted version of the associated OH inequality. For each edge t of the odd cycle $\mathcal{P}$, we select in the compatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ a vertex $v \in \mathcal{V}$ such that t is covered by v. Precisely, for each edge $t_i = (e_i, e_{i+1}) \in \mathcal{T}$, $i = 1, \ldots, 2k$, we select a vertex v_i in the set $\mathcal{V}(e_i) \cap \mathcal{V}(e_{i+1})$; and for edge $t_{2k+1} = (e_{2k+1}, e_1) \in \mathcal{T}$, we select a vertex v_{2k+1} in the set $\mathcal{V}(e_{2k+1}) \cap \mathcal{V}(e_1)$.

Remark that in the odd cycle there are no chords, hence, all the v_i are different from each other. The associated SR inequality is defined over $\mathcal{S} = \{v_1, \ldots, v_{2k+1}\}$ and multipliers $p_i = 1/2$ for each $i \in \mathcal{S}$:

$$\sum_{e \in \mathcal{E}} \left\lfloor \frac{1}{2} \sum_{i \in \mathcal{S}} a_i^e \right\rfloor \lambda_e \leq \left\lfloor \frac{|\mathcal{S}|}{2} \right\rfloor. \tag{24}$$

Lemma 1. *If a fractional solution $\tilde{\lambda}$ violates OH inequality* (23) *then it violates also SR inequality* (24) *associated with inequality* (23).

Proof. First, we observe that the right hand-side of inequality (24) coincides with the one of inequality (23), indeed, it holds $|\mathcal{S}| = |\mathcal{P}|$ by construction and $|\mathcal{P}|$ is odd. Then, in the left hand-side of inequality (24), at least the variables associated with exchanges in $\mathcal{P}$ appear with coefficient one. Indeed, such exchanges visit exactly two vertices in $\mathcal{S}$. Finally, it holds

$$\sum_{e \in \mathcal{E}} \left\lfloor \frac{1}{2} \sum_{i \in \mathcal{S}} a_i^e \right\rfloor \tilde{\lambda}_e \geq \sum_{e \in \mathcal{P}} \left\lfloor \frac{1}{2} \sum_{i \in \mathcal{S}} a_i^e \right\rfloor \tilde{\lambda}_e = \sum_{e \in \mathcal{P}} \tilde{\lambda}_e > \frac{|\mathcal{P}| - 1}{2} = \left\lfloor \frac{|\mathcal{S}|}{2} \right\rfloor \tag{25}$$

and $\tilde{\lambda}$ violates inequality (24). □

Note that Inequality (24) is a lifted version of Inequality (23). Indeed, the left hand-side of (24) contains additional variables in $\mathcal{E} \setminus \mathcal{P}$ with positive coefficients.

The impact on the pricing problem is the same as the one of the SR inequalities.

Example 3. *We make use of the toy instance of the KEP presented in Example 1. We consider fractional solution $\tilde{\lambda}$ defined as in Table 3. The associated conflict graph $\mathcal{C} = (\mathcal{Q}, \mathcal{T})$ is shown in Figure 3 where on each edge we report the set of vertices of the compatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ that can cover that edge. It is easy to see that the set of nodes $\mathcal{P} = \{e_1, e_2, e_3, e_4, e_5\}$ forms an odd cycle without chords in the conflict graph which give rise to a violated OH inequality. Indeed, by looking at Table 3, we have $\tilde{\lambda}_{e_1} + \tilde{\lambda}_{e_2} + \tilde{\lambda}_{e_3} + \tilde{\lambda}_{e_4} + \tilde{\lambda}_{e_5} = 2.5 > (|\mathcal{P}| - 1)/2 = 2$, thus the violation of the OH inequality is equal to 0.5.*

Now, we retrieve the associated SR inequality. In this case, we just need to define set $\mathcal{S}$. For each edge in the odd cycle, we select one of the vertices reported in Figure 3 on that edge. By defining $\mathcal{S} = \{1, 4, 6, 7, 8\}$, and multipliers $p_i = 1/2$, $i \in \mathcal{S}$, we obtain a violated SR inequality with the same violation of the OH inequality.

Table 3: Fractional solution $\tilde{\lambda}$.

exchange	value in $\tilde{\lambda}$
$e_1 = (1, 3, 4)$	0.5
$e_2 = (4, 5, 7, 4)$	0.5
$e_3 = (6, 7, 6)$	0.5
$e_4 = (2, 8, 9, 6)$	0.5
$e_5 = (1, 8, 9)$	0.5
$e_6 = (2, 3, 5)$	0.5

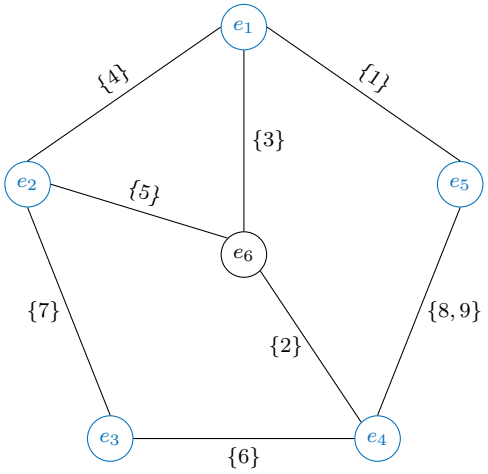

Figure 3: Conflict graph arising from the fractional solution of Table 3, where odd cycle without cords $\mathcal{P} = \{e_1, e_2, e_3, e_4, e_5\}$ is marked in blue.

4.4.4. Cut generation strategy

We use the following cut generation strategy. Preliminary results reveal that the best strategy is to add at most 100 inequalities to the MP in the branch-and-bound tree; after reaching this threshold, their separation is stopped. We allow ten minutes (out of the one-hour global time limit of the BPC algorithm) for the total time spent in the separation procedures; beyond this threshold, we do not separate any inequality. We insert a violated inequality in the RMP if its violation is greater than or equal to 0.1. At each separation round, we separate Cl and OH inequalities in this order. When 20 violated inequalities have been detected, we stop looking for other inequalities in that round. Separating the OH inequalities is rather time consuming and preliminary tests showed that the best strategy is to separate them only at the root node of the branch-and-bound tree. Hence, only Cl inequalities are separated at the other nodes of the tree. A time limit of one

minute is imposed for the separation of each of the two families of inequalities at each round. In addition, if during one round no Cl inequalities are detected, we keep separating OH inequalities only at that node.

Note that if non-robust valid inequalities are active in the RMP, the circuits are priced by a labelling algorithm instead of the variant of the Bellman-Ford algorithm presented in Section 4.3. Indeed, dealing with non-robust inequalities makes the pricing problem more complicated as it requires more resources, one per each non-robust inequality. In that context, labelling algorithms are well-suited for the management of these additional resources.

Finally, we recall that if non-robust valid inequalities are active in the RMP, the pricing of circuits and paths can no longer be modelled as a LPPLC, but must be modelled as an ELPPRC. In this case, all pricing problems are NP-hard.

4.5. Branching scheme

To ensure the integrality of the solution returned by the BPC algorithm, we branch on the use of the arcs in $\mathcal{A}$. Let $\tilde{\lambda}$ be an optimal fractional solution of the MP at a node of the branch-and-bound tree. We branch on values $f_{ij} = \sum_{e \in \mathcal{E}} b_{ij}^e \tilde{\lambda}_e$, $(i,j) \in \mathcal{A}$, where parameter b_{ij}^e takes value one if arc $(i,j) \in \mathcal{A}$ belongs to exchange $e \in \mathcal{E}$ and zero otherwise. Precisely, we select the branching candidate whose value is the closest to 0.5. Remark that value $f_{ij} \in]0,1[$, hence, in one branch we force arc (i,j) to be used ($\sum_{e \in \mathcal{E}} b_{ij}^e \lambda_e \geq 1$) while in the other branch, we forbid its use ($\sum_{e \in \mathcal{E}} b_{ij}^e \lambda_e = 0$). In the former case, dual variable $\phi_{ij} \leq 0$ associated with the branching constraint is considered when solving the pricing problem. Precisely, it is discounted from arc weight $\bar{W}_{ij}$ in Equation (9) or and (10). In the latter case, arc (i,j) is removed from graph $\mathcal{H}$ when we solve the pricing problem. Last, the branch-and-bound tree is explored according to the best-first strategy.

4.6. Accelerating techniques

We make use of the following accelerating techniques to speed up our BPC algorithm.

Preprocessing of Pansart et al. (2022). Imposing length constraints on cycles and chains allows us to perform a preprocessing procedure to reduce the size of compatibility graph $\mathcal{G}$. Such procedure was introduced by Pansart et al. (2022) and is based on the Floyd-Warshall algorithm to compute the shortest path between each pair of vertices in a graph. Precisely, we consider a graph $\tilde{\mathcal{G}} = (\{s\} \cup \mathcal{V}, \tilde{A})$, where $\tilde{\mathcal{A}} = \mathcal{A} \cup \{(s,i) : i \in \mathcal{D}\}$. Graph $\tilde{\mathcal{G}}$ is defined as graph $\mathcal{H}^P$ used to price out variables associated with exchange paths. The weights on the arcs are set to one. Then, the Floyd-Warshall algorithm is applied to $\tilde{\mathcal{G}}$ to detect the minimum length $l^*(i,j)$ paths between each pair of vertices $i,j \in \{s\} \cup \mathcal{V}$. We remove all arcs $(i,j) \in \mathcal{A}$ that satisfy the following two conditions:

1. the length of the shortest path from source s to vertex i is larger than or equal to the maximum path length, i.e., $l^*(s,i) \geq L^P + 1$;

2. the length of the shortest path from vertex j to vertex i is larger than or equal to the maximum cycle length, i.e., $l^*(j,i) \geq L^C$.

Finally, isolated vertices are removed from $\mathcal{V}$ as well.

Preprocessing of Riascos-Álvarez et al. (2023). Riascos-Álvarez et al. (2023) proposed a procedure to reduce the number of pricing subproblems [PP-C](i), $i \in \mathcal{I}$, to be processed in the pricing problem solution. In addition, the same procedure allows to reduce the size of pricing graphs $\mathcal{H}_i^C$. As Constantino et al. (2013) and Klimentova et al. (2014), Riascos-Álvarez et al. (2023) exploit the symmetry of the circuits subproblems: a circuit starting at vertex i considered in subproblem [PP-C](i) and containing vertex j will also appear as a circuit starting at vertex j when solving [PP-C](j). The procedure reads as follows. In the following, we work on a copy of compatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where we progressively remove vertices and arcs. For the ease of readability, we keep notation $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ to refer to this copy. We write $l^*(i,j)$ for the value of the shortest path, in terms of length, from i to j in compatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. First, the incompatible pairs with no in-going or no out-going arcs are removed from $\mathcal{I}$. Indeed, these pairs cannot be part of any circuit. The remaining ones are sorted by decreasing number of incident arcs. Then, the first pair $i \in \mathcal{I}$ is selected and graph $\mathcal{H}_i^C = (\mathcal{N}_i^C, \mathcal{F}_i^C)$ is built as follows. We include in $\mathcal{N}_i^C$ all pairs $j \in \mathcal{I}$ whose distance from i is less than or equal to the cycle length, i.e., $l^*(i,j) + l^*(j,i) \leq L^C$. Once the vertex set is built, arc set $\mathcal{F}_i^C$ contains all the arcs $(i',j') \in \mathcal{A}$ whose extremes are in $\mathcal{N}_i^C$ and such that $l^*(i,i') + l^*(j',i) + 1 \leq L^C$. Finally, vertex i is removed from graph $\mathcal{G}$ along with its incident arcs. Set $\mathcal{I}$ is re-ordered according to the reduced graph $\mathcal{G}$ and a new iteration starts. The procedure terminates when either all the pairs in $\mathcal{I}$ are processed or there is no pair with both in-going or out-going arcs. Those reduced graphs $\mathcal{H}_i^C$ are then used to price the variables associated with the circuits.

Initialisation of [RMP($\mathcal{E}'$) .] It is well known that column generation suffers from degeneracy which may slow down its convergence (Desrosiers and Lübbecke, 2005). A good initialisation of the set of columns helps in reducing this inconvenience. Hence, we enumerate at most 30000 exchange circuits and paths of length up to three and insert them in set $\mathcal{E}'$. Precisely, to ensure diversity, we generate at most $30000/|\mathcal{V}|$ exchange circuits/paths starting at each incompatible pair/altruistic donor in the compatibility graph.

Primal heuristic. Embedding primal heuristics in a BPC algorithm helps in improving the lower bound and, consequently, reducing the integrality gap (Archetti et al., 2013). We implement the so-called *restricted master heuristic*, i.e., we solve formulation [SP] restricted to the subset of variables generated so far by means of a commercial solver. We call such a heuristic after the root node is solved and at the end of the execution of the BPC algorithm, only if the time limit is reached. We impose a time limit of 60 seconds for the first call and 1800 seconds for the second one. Indeed, preliminary tests revealed that the restricted master heuristic struggles to find good lower bounds on the most complex

instances within reasonable computational times. The results also showed that it was useless to call such a heuristic during the execution of the BPC algorithm with smaller time limits: the lower bound did not improve within the time limit, and the number of nodes of the branch-and-bound tree pruned due to the lower bound was negligible.

Tabu list. As in Arslan et al. (2024), when a pricing subproblem [PP-C](i), $i \in \mathcal{I}$, does not provide any positive reduced cost circuit, we insert such subproblem in a tabu list. All subproblems in the tabu list are not solved in the following column generation iterations. They are solved only when subproblem [PP-P] does not provide any positive reduced cost path to ensure the correctness of the column generation procedure.

5. Computational experiments

Our BPC algorithm is implemented in C++ and compiled in release mode under a 64-bit version of MS Visual Studio 2019. The linear programming models in the column generation procedure, the mixed-integer programming models in the separation heuristic based on the Cl inequalities and the integer programming models in the restricted master heuristic are solved by GUROBI 9.5.2 (64-bit version). All the experiments are run on a 64-bit Windows machine equipped with a Intel(R) Xeon(R) Silver 4214 processor with 24 cores hyper-threaded to 48 virtual cores, with a base clock frequency of 2.2 GHz, and 96 GB of RAM. For each run of the algorithm, we impose one hour time limit and allow a single thread. If such time limit is reached, the primal heuristic is called to seek for lower bound improvements.

In this section, first, we present the characteristics of the benchmark instances we consider. We assess the impact of the valid inequalities to select the best configuration for the BPC algorithm (Section 5.2). Then, we compare the results obtained with the BPC algorithm on the benchmark instances with those reported in Pansart et al. (2022), Arslan et al. (2024) and Delorme et al. (2023). Note that these authors reported computational results on different datasets. Finally, in Section 5.4, we assess the impact of the length constraints on the objective function value.

The instances and detailed results can be found at Petris et al. (2024).

5.1. Benchmark instances

We assess the efficiency of the BPC algorithm on three different sets of instances considered in the literature: the "Kidney Data (00036)" set from the *PrefLib* dataset (Mattei and Walsh, 2013), the set proposed by Pansart et al. (2022) and the one considered by Delorme et al. (2023). The instances belonging to the first two sets are produced by the so-called *Saidman generator* (Saidman et al., 2006), while the ones belonging to the third set are produced by the generator introduced by Delorme et al. (2022). Both generators take as input the number of incompatible patient-donor pairs ($|\mathcal{I}|$), the number of altruistic donors ($|\mathcal{D}|$) and several medical related parameters to build a realistic compatibility graph $\mathcal{G}$. The medical related

parameters influence the density of the graph and the medical benefit associated with the arcs of the graph. The interested reader may refer to Saidman et al. (2006) and Delorme et al. (2022) for more information.

Table 4 summarises the characteristics of the instances in the three sets. The rows of the tables are associated with the sets of instances. The column headings are as follows: *set*: name of the set; #: total number of instances in the set; $|\mathcal{I}|$: number of incompatible patient-donor pairs in the set; $|\mathcal{D}|[\%:]$ percentage of altruistic donors w.r.t. the number of pairs $|\mathcal{I}|$; *avg. density of* $\mathcal{G}$: average density of compatibility graph $\mathcal{G}$ expressed as a percentage of arcs w.r.t. the number of arcs in a complete graph with the same number of vertices as $\mathcal{G}$; L^C: maximum length of the cycles; L^P: maximum length of the paths; *obj.*: type of objective function, either maximisation of the number of transplants ($\#TR$) or of the medical benefit (MB). Note that the total number of vertices in the compatibility graph is $(1 + |\mathcal{D}|[\%])|\mathcal{I}|$.

Table 4: Characteristics of the sets of instances.

set	#	Characteristics					
		$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert[\%]$	avg. density of $\mathcal{G}$	L^C	L^P	obj.
PrefLib	2000	16, 32, 64, 128, 256, 512, 1024, 2048	0%, 5%, 10%, 15%	25%	3, 4	0, 3, 4, 5, 6	#TR
Pansart et al. (2022)	270	50, 100, 250, 500, 750, 1000	5%, 10%, 15%	5%	3	3, 6, 12	MB
Delorme et al. (2023)	840	50, 100, 200, 400, 600, 800, 1000	0%	10%	3, 4, 5, 6, 7, 8	0	#TR, MB

5.2. Impact of the valid inequalities

In this section, we evaluate the impact of the valid inequalities presented in Section 4.4, namely the SR inequalities heuristically separated from either the Cl inequalities (13) or the OH inequalities (23).

Precisely, we first assess their impact at the root node of the branch-and-bound tree, then on the entire tree. We conduct this analysis on the most challenging instances, namely the 135 instances of Pansart et al. (2022) with a number of pairs greater than or equal to 500. The objective is the maximisation of the medical benefit. Although the instances of Delorme et al. (2023) with a number of pairs greater than 200 are also challenging when the medical benefit is maximised, we choose to report only the results for the 135 challenging instances of Pansart et al. (2022) as they are similar to the ones obtained on the set of Delorme et al. (2023).

In order to perform this analysis, we define the following four configurations of the BPC algorithm: `BPC`: no valid inequalities is separated in the course of the algorithm; `BPC+Cl`: only clique inequalities (13) are separated; `BPC+OH`: only odd-hole inequalities (23) are separated; `BPC+Cl+OH`: all valid inequalities are separated.

First, in Table 5, we compare the results obtained by the four configurations at the end of the solution of the root node of the branch-and-bound tree. The rows of the table correspond to the four configurations. The first column reports the name of the configuration (*configuration*). The following three columns report statistics regarding the impact of the valid inequalities: the number of instances solved to optimality at the root node (*#opt.*); the average gap at the root node (*avg.gap[%no.cuts]*), expressed as a percentage of the gap at the root node when no cuts are added, that is computed as $100(UB^{VI} - LB^*)/(UB^{root} - LB^*)100$, where UB^{root} and UB^{VI} are the upper bounds obtained, respectively, before and after the valid inequalities and LB^* is the best lower bound among those returned by the four configurations; the average time in seconds to solve the root node (*avg.t[s]*). We chose to assess the impact of the valid inequalities by measuring how much they help to close the optimality gap rather than the average upper bound improvement because the upper bounds at the root node are already of very good quality. Hence, when adding the valid inequalities, the upper bounds can decrease only by a small amount that does not reflect the real impact of the inequalities. With this measure, the smaller the value, the better the performance of the inequalities. The last three columns report statistics on the cut generation strategy: the average number of SR inequalities arising from the Cl inequalities (*avg.#Cl*), the average number of SR inequalities arising from the OH inequalities (*avg.#OH*) and the average separation time (*avg.t[s] sep.*). We use '-' when the information is not available.

Table 5: Comparison of four configurations of the BPC algorithm at the root node of the branch-and-bound tree.

configuration	#opt.	avg.impr. cl.gap[%]	avg.t[s]	avg.#Cl	avg.#OH	avg.t[s] sep.
`BPC`	2	100.00	84.02	-	-	-
`BPC+Cl`	5	87.31	89.08	7.97	-	0.28
`BPC+OH`	14	75.36	187.64	-	19.57	101.59
`BPC+Cl+OH`	14	74.34	169.60	9.52	13.60	78.54

From the root node analysis, it is clear that separating the Cl and/or the OH inequalities is beneficial w.r.t. configuration `BPC`. Indeed, when the inequalities are considered, the number of solution proven to be optimal increases: it grows to five with `BPC+Cl` and to 14 with the configurations involving the OH inequalities. In addition, adding the valid inequalities help in reducing the gap at the root node w.r.t. `BPC`. The best configuration is when separating all inequalities (`BPC+Cl+OH`) that leads to a gap of 74.34% w.r.t. the one obtained by `BPC`. Note that only separating OH inequalities also leads to a very good root node gap (75.36% w.r.t. the one at the root node), while only separating Cl inequalities has worse performance (87.31% w.r.t. the one at the root node). The price to pay with the inequalities is an increase of the computational time, which is mostly due to the time required by the separation procedures. Precisely, separating the Cl inequalities is much faster than separating the OH inequalities: the separation time in `BPC+Cl` is on average 0.28 seconds whereas the one of `BPC+OH` is 101.59 seconds. This difference in the time taken by the algorithm can be explained as follows: the procedure used to separate Cl inequalities (see Section 4.4.2) is designed as

a very fast heuristic, whereas the procedure to separate OH inequalities (see Section 4.4.3) is designed as an exact algorithm which is turned into a good quality heuristic by considering 30% of starting vertices, that is still time consuming.

In Table 6, we compare the results obtained by the four configurations when the instances are solved to optimality. The rows and the first column of the table are the same as those of Table 5. The second column reports the number of additional instances solved to optimality by each configuration in addition to the 50 instances solved to optimality by all configurations (*#add.opt.*). The following five columns report statistics regarding the 50 instances solved to optimality by all configurations: the average number of nodes of the branch-and-bound tree (*avg.#nodes*), the average computational time in seconds (*avg.t[s]*), the average number of SR inequalities arising from the Cl inequalities (*avg.#Cl*), the average number of SR inequalities arising from the OH inequalities (*avg.#OH*) and the average separation time in seconds (*avg.t[s] sep.)*. We use '-' when the information is not available.

Table 6: Comparison of four configurations of the BPC algorithm on the instances solved to optimality.

configuration	#add.opt.	avg.#nodes	avg.t[s]	avg.#Cl	avg.#OH	avg.t[s] sep.
`BPC`	3	682.70	332.43	-	-	-
`BPC+Cl`	4	269.46	257.60	23.70	-	1.26
`BPC+OH`	6	253.38	282.88	-	24.82	14.51
`BPC+Cl+OH`	6	174.42	254.8	20.76	18.56	14.18

The results of the root node analysis are confirmed: including the valid inequalities is beneficial. First, the inequalities allow to solve more instances to optimality: 54 with `BPC+Cl` and 56 `BPC+OH` and `BPC+Cl+OH`. In addition, on the 50 instances solved to optimality by all four configurations, adding the inequalities permit to reduce: (i) the number of the nodes of the branch-and-bound tree by a factor of at least 2.5; (ii) the computational time by a factor of at least 1.2. The best configuration is `BPC+Cl+OH`: these two reduction factors become 4.0 and 1.3, respectively. The number of separated valid inequalities is on the order of tens for all the configurations where they are considered. The separation time follows the same trend as in Table 5.

Finally, Table 7 compares the results obtained by the four configurations when the instances are not solved to optimality. The rows and the first column of the table are the same as those of Table 5. The second column reports the number of additional instances not solved to optimality by each configuration in addition to the 77 instances not solved to optimality by all configurations (*#add.noOpt.*). The following seven columns report statistics regarding the 77 instances not solved to optimality by all configurations: the average number of nodes of the branch-and-bound tree (*avg.#nodes*), the average lower bound when the time limit is reached (*avg.LB*), the average upper bound when the time limit is reached (*avg.UB*), the average optimality gap expressed as a percentage, i.e., $100(UB-LB)/LB$ (*avg.gap[%]*), the average number of SR inequalities arising from the Cl inequalities (*avg.#Cl*), the average number of SR inequalities arising

from the OH inequalities (*avg.#OH*) and the average separation time in seconds (*avg.t[s] sep.)*. We use '-' when the information is not available.

Table 7: Comparison of four configurations of the BPC algorithm on the instances not solved to optimality.

configuration	#add.noOpt.	avg.#nodes	avg.LB	avg.UB	avg.gap[%]	avg.#Cl	avg.#OH	avg.t[s] sep.
BPC	5	1908.64	39571.57	39747.08	0.41	-	-	-
BPC+Cl	4	1197.48	39567.40	39746.16	0.43	91.95	-	11.82
BPC+OH	2	1271.79	39577.23	39746.04	0.40	-	15.69	165.98
BPC+Cl+OH	2	1085.87	39560.12	39744.84	0.43	85.25	10.14	136.69

The benefit of the valid inequalities on the instances not solved to optimality is less evident. Overall the configurations involving the inequalities explore on average less nodes of the branch-and-bound tree w.r.t. BPC. The explanation is that when valid inequalities are active in the RMP, the pricing of the circuits is modelled as an ELPPRC (i.e. a NP-hard problem) and can no longer be modelled as an LPPLC (i.e. a polynomial problem). Therefore, each call to pricing is more time consuming when valid inequalities are active.

In addition, it can be observed that the SR inequalities arising from the OH inequalities are few, and even in the case where only OH inequalities are separated, this number (15.69 on average) is far from the maximum of 100 valid inequalities to be added in the model. On the contrary, there are many SR inequalities arising from the Cl inequalities. This can be explained by the fact that only Cl inequalities are separated after solving the root node of the branch-and-bound tree. The time spent in the separation of the inequalities is small w.r.t. the total time (less than 3 minutes on average out of one hour). As observed at the root node, when separating only Cl inequalities the separation time is negligible, only a few seconds.

Finally, in the four configurations the average lower bound, upper bound and optimality gap are rather similar. For the lower bounds, they are slightly worse for the configurations involving the separation of the Cl inequalities, and slightly better for the configuration BPC+OH. The quality of the lower bound comes from the final call to the primal heuristic where a commercial solver solves formulation [SP] restricted to the columns generated so far. As already mentioned, such formulation is very difficult to solve, and even with a 30 minutes time limit, the reported lower bound is usually not proven optimal. Hence, there is no direct correlation between the use of valid inequalities and the quality of the lower bounds. When looking at the average upper bounds, although less nodes are explored, the average upper bound of the configurations with the inequalities is slightly better than the one of BPC. In this respect, the best configuration is BPC+Cl+OH. This balance between worsening the lower bounds and improving the upper bounds yields the optimality gap to be comparable among all the configurations.

We choose configuration BPC+Cl+OH for the BPC algorithm because: (i) it provides the best results on the instances solved to optimality; (ii) it provides the best upper bound improvements on the instances not

solved to optimality. Hereafter, we will simply refer to this configuration as the BPC algorithm.

5.3. Results on the whole testbed

In this section, we present the aggregated results obtained by the BPC algorithm on the three sets of instances. We compare our results with those reported in the literature.

Throughout the section the percentage optimality gap is computed as $100(UB - LB)/LB$, where UB and LB are the upper and lower bounds returned by the BPC algorithm.

5.3.1. Results on the PrefLib dataset

In this section, we compare the results obtained by the BPC algorithm on the instances of the PrefLib dataset with those obtained by the BP algorithm of Arslan et al. (2024).

Table 8 reports the results on the PrefLib dataset. The rows of the table group instances with given maximum path lengths and given numbers of incompatible pairs. Note that the first row groups all small size instances with a maximum of 512 pairs, while the results for large instances with 1024 or 2048 pairs are reported in the other rows. The first three columns report some information about the instance subset. The next three columns summarise the results obtained by the BPC algorithm: number of instances solved to optimality (*#opt.*), average number of nodes of the branch-and-bound tree (*avg.#nodes*) and average computational time in seconds (*avg.t[s]*). The last column reports the computational time in seconds of the BP algorithm of Arslan et al. (2024) (*avg.t[s]*). The authors implemented their BP algorithm using Julia language and ran the experiments on a 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 2.5 GHz machine with 128Go RAM running Linux OS. In this column, we make use of '-' when no result is available: Arslan et al. (2024) do not test the BP algorithm on instances where only circuits are to be priced, i.e., those with $L^P = 0$. No further information is required about their algorithm, since, it manages to solve to optimality all the instances at the root node.

Table 8: Results on the PrefLib dataset.

Instances			BPC			Arslan et al. (2024)
L^P	$\|\mathcal{I}\|$	#	#opt.	avg.#nodes	avg.t[s]	avg.t[s]
0, 3, 4, 5, 6	16, 32, 64, 128, 256, 512	1480	1480	1.00	1.34	0.99
0	1024	20	20	1.00	18.43	-
	2048	20	20	1.00	167.59	-
3	1024	60	60	1.00	25.01	3.75
	2048	60	60	18.17	353.08	11.93
4	1024	60	60	1.00	25.30	3.70
	2048	60	60	1.00	294.06	12.46
5	1024	60	60	1.00	25.62	3.72
	2048	60	60	2.98	353.75	12.56
6	1024	60	60	1.00	25.74	3.72
	2048	60	60	1.00	295.66	12.46

From the results of Table 8, it can be observed that both approaches yield comparable results on instances with up to 512 pairs, regardless of the maximum length of the paths. Indeed, both algorithms solve those instances to optimality at the root node (before the first separation round), except in one case where the BPC algorithm required the insertion of valid inequalities and the exploration of two nodes. The computational times are of the same order of magnitude. Even though all the instances characterised by a number of pairs equal to 1024 or 2048 are solved to optimality by both approaches, the BP algorithm of Arslan et al. (2024) is faster than the BPC algorithm by an order of magnitude. The majority of the computational time of the BPC algorithm is spent in the column generation procedure. With our BPC algorithm, valid inequalities are inserted into the RMP for two instances only, and in such a cases the pricing problem is then solved with a labelling algorithm for the ELPPRC. For all other instances of the PrefLib dataset, solving the LPPLC is sufficient to solve the pricing problem, the conditions to necessarily solve an ELPPLC are never reached. Hence, the BP and BPC algorithms make use of the same pricing strategy and the difference in computation times are probably due to a more efficient implementation of the algorithms by Arslan et al. (2024).

It can be noticed that no instances of this set is difficult to solve. With the BPC algorithm all instances are solved in on average 44.80 seconds.

5.3.2. Results on the set of instances of Pansart et al. (2022)

In this section, we discuss the results obtained by the BPC algorithm on the instances introduced by Pansart et al. (2022). We compare the BPC algorithm performances with the restricted master heuristic of Pansart et al. (2022) on the subset of small instances characterised by a number of pairs up to 250. We recall that the approach of Pansart et al. (2022) is a heuristic with a performance guarantee, that is, it provides valid lower and upper bounds on the optimal value of formulation [SP]. The restricted master

heuristic is run on a machine equipped with an Intel Xeon E5-2440 v2 @ 1.9 GHz processor and 32 GB of RAM. Note that Pansart et al. (2022) do not test their procedure on all these instances with more than 500 pairs.

Table 9 summarises the results of the comparison between the BPC algorithm and the restricted master heuristic of Pansart et al. (2022). The rows of the table group instances with the same maximum path length and the same number of incompatible pairs. The first three columns report some information about the instance subset: maximum path lengths (L^P), number of incompatible patient-donor pairs ($|\mathcal{I}|$) and number of instances (#). The next three columns show the results obtained by the BPC algorithm: number of instances solved to optimality (*#opt.*), average number of nodes of the branch-and-bound tree (*avg.#nodes*) and average computational time in seconds (*avg.t[s]*). The last three columns report some statistic related to the algorithm of Pansart et al. (2022): number of instances solved to optimality (*#opt.*), average computational time in seconds (*avg.t[s]*) and optimality gap expressed as a percentage for the instances not solved to optimality (*avg.gap[%] noOpt.*). If all the instances in a row are solved to optimality by the algorithm of Pansart et al. (2022), we make use of '-' in column *avg.gap[%] noOpt.*.

Table 9: Results on the set of instances of Pansart et al. (2022) with up to 250 pairs.

Instances			BPC			Pansart et al. (2022)		
L^P	$\|\mathcal{I}\|$	#	#opt.	avg.#nodes	avg.t[s]	#opt.	avg.t[s]	avg.gap[%] noOpt.
	50	15	15	1.00	0.04	15	1.07	-
3	100	15	15	1.00	0.15	9	1.21	0.33
	250	15	15	1.00	1.06	7	3.45	0.07
	50	15	15	1.40	0.04	14	1.20	0.05
6	100	15	15	2.07	0.36	8	2.17	0.64
	250	15	15	45.27	16.55	2	27.74	0.32
	50	15	15	1.00	0.04	15	2.52	-
12	100	15	15	1.13	0.29	14	26.59	0.19
	250	15	15	40.47	142.81	8	1420.20	1.26

From the results in Table 9, the BPC algorithm solves all the 135 instances to optimality. The optimality is proven at the root node for 104 instances in on average 0.48 seconds. For the remaining ones, the BPC algorithm explores a few tens of nodes on 29 instances and a few hundreds on the remaining two. For these instances, the computational time increases. On average, it is equal to 29.35 seconds, however it is always less than 300 seconds. Conversely, the heuristic with a performance guarantee of Pansart et al. (2022) proves the optimality of 92 instances, leaving the remaining ones with a positive optimality gap equal to 0.47%, on average. In terms of computational time, the BPC algorithm is faster than the algorithm of Pansart et al. (2022) on 133 instances. The two instances where the BPC algorithm is slower correspond to those where the optimality is not proved at the root node. Hence the BPC algorithm branches, while the heuristic of Pansart et al. (2022) stops at the root node. Moreover, it is interesting to note that on the most difficult

instances, that is with $L^P = 12$ and 250 pairs, the BPC algorithm is faster by an order of magnitude w.r.t. the algorithm of Pansart et al. (2022).

In Table 10, we report the results obtained by the BPC algorithm on the instances of Pansart et al. (2022) with $|\mathcal{I}| = 500, 750, 1000$. The rows of the table correspond to instances with the same maximum path length and the same number of incompatible pairs. The first three columns have the same meaning as those in Table 9. The next three columns summarise the results obtained by the BPC algorithm on the instances solved to optimality: number of instances solved to optimality (*#*), average number of nodes of the branch-and-bound tree (*avg.#nodes*) and average computational time in seconds (*avg.t[s]*). The next three columns summarise the results obtained by the BPC algorithm on the instances not solved to optimality: number of instances not solved to optimality (*#*), average number of nodes of the branch-and-bound tree (*avg.#nodes*) and optimality gap expressed as a percentage (*avg.gap[%]*). The last three columns report the same information about the results of Pansart et al. (2022) as in Table 9. We write '-' in columns *avg.#nodes* and *avg.t[s]* if there are no instances solved/unsolved in a row by the BPC algorithm and in the columns concerning the results of Pansart et al. (2022) when they are not available. Indeed, we recall that Pansart et al. (2022) only provide results for three subset of instances with at least 500 pairs.

Table 10: Results on the set of instances of Pansart et al. (2022) with at least 500 pairs.

Instances			BPC results						Pansart et al. (2022)		
			solved instances			unsolved instances					
L^P	$\|\mathcal{I}\|$	#	#	avg.#nodes	avg.t[s]	#	avg.#nodes	avg.gap[%]	#opt.	avg.t[s]	avg.gap[%] noOpt.
3	500	15	15	158.93	73.98	0	-	-	1	14.23	0.13
	750	15	13	646.85	418.59	2	3361.00	0.04	0	85.51	0.15
	1000	15	5	177.00	129.37	10	2711.40	0.05	-	-	-
6	500	15	6	89.67	90.32	9	2233.44	0.10	0	1011.70	0.44
	750	15	5	286.60	370.89	10	918.70	0.25	-	-	-
	1000	15	1	1353	2733.54	14	942.43	0.24	-	-	-
12	500	15	7	75.43	599.08	8	523.50	0.40	-	-	-
	750	15	4	74.75	1206.27	11	227.91	0.65	-	-	-
	1000	15	0	-	-	15	118.47	1.04	-	-	-

Table 10 shows that the performances of the BPC algorithm decrease as the size of the instances increases, both in terms of maximum path length and number of pairs. The BPC algorithm identifies 56 optima out of the 135 instances: their distribution among the instances with $L^P = 3$, $L^P = 6$ and $L^P = 12$ is 33, 12 and 11, respectively. Detailed results show that the time required to prove the optimality varies heavily even within instances characterised by the same values of L^P and $|\mathcal{I}|$. Precisely, it increases with the number of nodes of the branch-and-bound tree explored by the BPC algorithm. As an example, in the 13 instances solved to optimality with $L^P = 3$ and $|\mathcal{I}| = 750$, the computational time is less than four seconds when the instances are solved to optimality at the root node while it is larger than 200 seconds when more than 100

nodes are explored.

The BPC algorithm is not able to prove the optimality of 79 instances. However, they are left with a small optimality gap, on average, equal to 0.05%, 0.21% and 0.76% for the instances with $L^P = 3$, $L^P = 6$ and $L^P = 12$, respectively. We observe that exploring the branch-and-bound tree is rather inefficient in improving the upper bound. Indeed, the improvement of the upper bound w.r.t. the one obtained at the root node is on average equal to 0.08%.

The comparison with the results of Pansart et al. (2022) considers the subset of 45 instances with at most 750 incompatible pairs and path length up to six where their restricted master heuristic is tested. The results provided on those instances are worse than ours. Only one instance is solved to optimality, the remaining ones are left with an average optimality gap of 0.24% (see last three columns of the table).

Finally, we tested the code of the BP algorithm of Arslan et al. (2024)[3] on the instances of Pansart et al. (2022) characterised by values of the maximum length of the paths larger or equal to six. Our tests show that the algorithm do not scale on such large instances and we have not been able to get results within a time limit of one hour. We suppose this is due to the necessity of solving the ELPPLC to price out paths on such instances, while it was never the case when solving the PrefLib instances. In the BP algorithm of Arslan et al. (2024), the ELPPLC is solved with a mixed integer program with exponentially-many constraints. It turns to be very time consuming to employ such an algorithm to solve the ELPPLC. Hence, we do not report any detailed results for the BP on these instances.

5.3.3. Results on the set of instances of Delorme et al. (2023)

Table 11 reports the results obtained on the instances of Delorme et al. (2023) when the objective function is the maximisation of the number of transplants. A comparison with the results of Delorme et al. (2023) is also conducted. The algorithm of Delorme et al. (2023) was implemented in C++ and ran on a machine equipped with an Intel(R) Core(TM) i5-1135G7, 2.40GHz and 32GB of memory.

Each row of the table corresponds to a subset of instances characterised by the same number of pairs and the same maximum length of the cycles. The first three columns report the characteristics of the subset of instances: the maximum length of the cycles (L^C), the number of pairs ($|\mathcal{I}|$) and the number of instances in the subset ($\#$). The next two columns report the results obtained by the BPC algorithm, namely, the number of instances solved to optimality ($\#opt.$) and the average time expressed in seconds to solve the instances to optimality ($avg.t[s]$). The last two columns report the same two statistics on the results obtained by Delorme et al. (2023). Columns are doubled to include all the instances in the table.

The results of Table 11 clearly highlight the superiority of the BPC algorithm w.r.t. the destructive bound procedure of Delorme et al. (2023). Indeed, the BPC algorithm solves to optimality each of the 840 instances in less than 350 seconds. More precisely, the computational time is less than 30 seconds for 837 instances. 812 instances are solved at the root node, the remaining ones by exploring 244 nodes of the

[3]the code is available online at `https://github.com/jeremyomer/KidneyExchange.jl`

Table 11: Results on the set of instances of Delorme et al. (2023) where the objective is the maximisation of the number of transplants.

Instances			BPC		Delorme et al. (2023)		Instances			BPC		Delorme et al. (2023)	
L^C	$\|\mathcal{I}\|$	#	#opt.	avg.t[s]	#opt.	avg.t[s]	L^C	$\|\mathcal{I}\|$	#	#opt.	avg.t[s]	#opt.	avg.t[s]
	50	20	20	0.02	20	0.00		50	20	20	0.02	20	0.00
	100	20	20	0.02	20	0.00		100	20	20	0.08	20	0.00
	200	20	20	0.09	20	0.00		200	20	20	7.11	20	15.00
3	400	20	20	0.53	20	0.00	6	400	20	20	0.94	20	680.00
	600	20	20	1.92	20	1.00		600	20	20	2.25	0	3600.00
	800	20	20	2.87	20	3.00		800	20	20	3.70	-	-
	1000	20	20	3.88	20	6.00		1000	20	20	6.00	-	-
	50	20	20	0.02	20	0.00		50	20	20	0.02	20	0.00
	100	20	20	0.04	20	0.00		100	20	20	0.25	20	0.00
	200	20	20	0.55	20	0.00		200	20	20	17.90	20	57.00
4	400	20	20	0.75	20	7.00	7	400	20	20	0.93	10	3138.00
	600	20	20	2.09	20	41.00		600	20	20	2.43	0	-
	800	20	20	3.14	20	107.00		800	20	20	4.38	-	-
	1000	20	20	4.30	20	294.00		1000	20	20	6.50	-	-
	50	20	20	0.02	20	0.00		50	20	20	0.02	20	0.00
	100	20	20	0.11	20	0.00		100	20	20	0.22	20	1.00
	200	20	20	1.78	20	2.00		200	20	20	2.36	20	250.00
5	400	20	20	0.84	20	76.00	8	400	20	20	1.04	0	-
	600	20	20	2.28	20	623.00		600	20	20	2.44	-	-
	800	20	20	3.55	13	2735.00		800	20	20	4.35	-	-
	1000	20	20	5.13	0	3600.00		1000	20	20	7.01	-	-

branch-and-bound tree, on average. Conversely, the procedure of Delorme et al. (2023) suffers from lack of scalability. Indeed, such procedure entails enumerating all the half-cycles of length up to $\lceil L^C \rceil + 1$. This operation becomes intractable when the size of the instance grows.

In Table 12, we present the results obtained by the BPC algorithm where the objective function is the maximisation of the medical benefit. Delorme et al. (2023) do not test their procedure in the case of maximisation of the medical benefit. The rows and the first three columns of the table have the same meaning as those in Table 11. The next three columns report results regarding the instances solved to optimality by the BPC algorithm: the number of such instances (#), the average number of nodes of the branch-and-bound tree to prove the optimality (*avg.#nodes*) and the average time expressed in seconds (*avg.t[s]*). The last three columns report the results regarding the instances not solved to optimality by the BPC algorithm: the number of such instances (#), the average number of nodes of the branch-and-bound tree explored by the BPC algorithm (*avg.#nodes*) and the average optimality gap expressed as a percentage (*avg.gap[%]*). We consider only instances characterised by a number of pairs up to 400.

From the results of Table 12, it can observed that the type of the objective function has a huge impact on the results. Maximising the medical benefit makes the instances much harder to be solved by the BPC algorithm. The BPC algorithm still provides 384 optima out of the 480 instances. However, contrary to the results where the objective is to maximise the number of transplants, those optima are obtained by exploring more nodes of the branch-and-bound tree (167, on average) and by spending more time (90 seconds, on average). The 96 instances not solved to optimality are characterised by either 200 or 400 pairs and a maximum length of the cycles at least equal to four. None of the instances with 400 pairs and maximum length greater or equal to five is solved to optimality. Nonetheless, the average optimality gap when the time limit is reached is small, 0.43%, on average. The same behaviour was observed when considering instances with more than 400 pairs.

5.4. Impact of the length constraints on the objective function

In this section, we assess the impact of the length constraints of the exchange paths on the objective function value. To conduct such analysis, we take into account both objective function types, i.e., either maximisation of the number of transplants (#TR) or maximisation of the medical benefit (MB). For the former case, we only consider the instances of the PrefLib dataset with maximum length of the circuits $L^C = 3$; for the latter case, we only consider the instances of Pansart et al. (2022) with 50, 100, or 250 incompatible pairs. In both sets, we group the instances characterised by the same compatibility graph and we observe how the objective function value behaves when the maximum length of the exchange paths increases. Note that all these instances are solved to optimality by the BPC algorithm.

In the case of the maximisation of the number of transplants, we do not report any result because the improvements are extremely sporadic. Indeed, on the 230 instances characterised by the same compatibility graph, the number of transplants improves only 19 times when we increase the length of the paths from 3 to 4, from 4 to 5 or from 5 to 6. Therefore, it seems that the influence of the path length on the maximum

Table 12: Results on the set of instances of Delorme et al. (2023) where the objective is the maximisation of the medical benefit.

Instances			BPC results					
			solved instances			unsolved instances		
L^C	$\|\mathcal{I}\|$	#	#	avg.#nodes	avg.t[s]	#	avg.#nodes	avg.gap[%]
3	50	20	20	1.00	0.05	0	-	-
	100	20	20	1.10	0.02	0	-	-
	200	20	20	1.40	0.09	0	-	-
	400	20	20	24.40	4.80	0	-	-
4	50	20	20	1.00	0.05	0	-	-
	100	20	20	2.50	0.07	0	-	-
	200	20	20	66.10	6.97	0	-	-
	400	20	9	1485.22	574.63	11	8452.18	0.07
5	50	20	20	1.00	0.05	0	-	-
	100	20	20	3.00	0.18	0	-	-
	200	20	20	154.10	36.59	0	-	-
	400	20	0	-	-	20	2295.15	0.32
6	50	20	20	1.00	0.05	0	-	-
	100	20	20	8.85	0.72	0	-	-
	200	20	19	652.79	294.29	1	6989.00	0.04
	400	20	0	-	-	20	834.00	0.56
7	50	20	20	1.00	0.02	0	-	-
	100	20	20	14.65	2.44	0	-	-
	200	20	19	1407.32	887.05	1	5965.00	0.01
	400	20	0	-	-	20	466.25	0.55
8	50	20	20	1.00	0.02	0	-	-
	100	20	20	8.65	2.45	0	-	-
	200	20	17	344.47	335.58	3	3115.67	0.08
	400	20	0	-	-	20	326.70	0.59

number of transplants is negligible. We try to explain this phenomenon with the results in Table 13. In this table, we group the instances by maximum length of the paths L^P. The first three columns report the value of L^P, the number of instances in each group (#) and the total number of altruistic donors (*tot.*$|\mathcal{D}|$), that is the sum of the altruistic donors over the instances. The following six columns report the total number of paths in the optimal solution (*tot.#paths*) and, for each length of the paths $l = 1, \ldots, L^P$, the percentage of the total number of paths with length l (*#paths per length[%]*).

Table 13: Statistics on the length of the paths in the optimal solutions of the set of instances of the PrefLib dataset with $L^C = 3$.

Instances			#paths per length[%]						
L^P	#	tot.$\|\mathcal{D}\|$	tot.#paths	1	2	3	4	5	6
3	230	12120	12120	77.02	17.24	5.74	-	-	-
4	230	12120	12120	77.06	16.96	5.07	0.92	-	-
5	230	12120	12120	77.09	16.85	5.15	0.65	0.26	-
6	230	12120	12108	77.27	16.73	5.10	0.64	0.26	0.13

First, almost all the altruistic donors available in the compatibility graphs start a path in the optimal solutions (see columns *tot.*$|\mathcal{D}|$ and *tot.#paths*). Precisely, all of them are used in the instances with $L^P = 3, 4, 5$ and only 12 out of 12120 of them remain unused in the instances with $L^P = 6$. However, the vast majority of the paths are of small length: in each group of instances, at least the 99% of the paths is of length less than or equal to three.

Finally, Table 14 reports the average improvements of the medical benefit as the maximum length of the paths increases on the instances of Pansart et al. (2022) with up to 250 incompatible pairs. The rows of the table group instances with the same number of incompatible pairs (same compatibility graph). The first two columns report the number of incompatible pairs ($|\mathcal{I}|$) and the number of instances (#) in each group. The remaining two columns report the average improvement of the medical benefit (objective function value) expressed as a percentage (*avg.impr[%]*) as the maximum path length increases from three to six ($L^P : 3 \to 6$) and from six to twelve ($L^P : 6 \to 12$), respectively.

Table 14: Impact of the path length increase on the set of instances of Pansart et al. (2022) when the objective is the maximisation of the medical benefit (MB).

Instances		avg.impr[%]	
$\|\mathcal{I}\|$	#	$L^P : 3 \to 6$	$L^P : 6 \to 12$
50	15	9.31	2.13
100	15	15.77	5.41
250	15	14.36	5.13

A different trend emerges from the results in Table 14 w.r.t. the case of the maximisation of the number

of transplants. Indeed, we observe that the maximum length constraints on the paths has a great impact on the value of the objective function. The largest improvements of the medical benefit are obtained when the path length is increased from three to six. These improvements are reduced when the maximum length is increased from six to twelve.

6. Conclusions

In this paper, we introduced an exact approach which is able to solve the Kidney Exchange Problem (KEP), especially for the most difficult instances where long chains of donations and maximisation of the medical benefit are considered. Our approach is based on a Branch-Price-and-Cut (BPC) algorithm where the pricing problem is decomposed in subproblems to price out variables associated with cycles and chains, respectively. We consider the subset-row inequalities to strengthen the relaxation. We propose two heuristics to separate them based on the separation of the clique and odd-hole inequalities, respectively.

We test the BPC algorithm against the three algorithms recently proposed in the literature, namely, the restricted master heuristic of Pansart et al. (2022), the destructive bound procedure of Delorme et al. (2023) and the Branch-and-Price algorithm of Arslan et al. (2024). The results reveal that the BPC algorithm positions itself in the literature as the only exact algorithm capable of solving efficiently instances with various characteristics and with different degrees of difficulty. Indeed, although the approaches of Pansart et al. (2022) and Arslan et al. (2024) are able to solve any kind of instance, the BPC algorithm clearly outperform the first and the second one does not scale on instances with long chains and maximisation of medical benefit as objective. The approach of Delorme et al. (2023) solves instances with cycles only and, in any case, its results are clearly outperformed by those of the BPC algorithm.

In addition, the impact of the valid inequalities is assessed: they permit to reduce both the time and the exploration of the branch-and-bound tree when the instances are solved to optimality and they slightly improve the upper bound on the instances not solved to optimality.

From our results, there are still some instances difficult to solve, the ones with a large number of incompatible pairs, and long chains of donations where the objective is the maximisation of the medical benefit, For such instances, the bottleneck seems to be the quality of the lower bound. Hence, an interesting future work would be to design fast and efficient heuristic for the KEP instances with such characteristics. Having a very good lower bound would help to cut unpromising nodes in the branch-and-bound tree. Another interesting research direction may consider the development of a BPC algorithm based on the formulation introduced by Delorme et al. (2023), where the exponentially-many variables are the *half-cycles*.

References

Abraham, D.J., Blum, A., Sandholm, T., 2007. Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges, in: Proceedings of the 8th ACM conference on Electronic commerce, pp. 295–304.

Archetti, C., Bianchessi, N., Speranza, M., 2013. Optimal solutions for routing problems with profits. Discrete Applied Mathematics 161, 547–557. doi:10.1016/j.dam.2011.12.021.

Arslan, A.N., Omer, J., Yan, F., 2024. Kidneyexchange. jl: A julia package for solving the kidney exchange problem with branch-and-price. Mathematical Programming Computation , 1–34.

Axelrod, D.A., Schnitzler, M.A., Xiao, H., Irish, W., Tuttle-Newhall, E., Chang, S.H., Kasiske, B.L., Alhamad, T., Lentine, K.L., 2018. An economic assessment of contemporary kidney transplant practice. American Journal of Transplantation 18, 1168–1176.

Baldacci, R., Mingozzi, A., Roberti, R., 2011. New route relaxation and pricing strategies for the vehicle routing problem. Operations Research 59, 1269–1283. doi:10.1287/opre.1110.0975.

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H., 1998. Branch-and-price: Column generation for solving huge integer programs. Operations Research 46, 316–329. doi:10.1287/opre.46.3.316.

Biró, P., Haase-Kromwijk, B., Andersson, T., Ásgeirsson, E.I., Baltesová, T., Boletis, I., Bolotinha, C., Bond, G., Böhmig, G., Burnapp, L., et al., 2019. Building kidney exchange programmes in europe—an overview of exchange practice and activities. Transplantation 103, 1514.

Biró, P., van de Klundert, J., Manlove, D., Pettersson, W., Andersson, T., Burnapp, L., Chromy, P., Delgado, P., Dworczak, P., Haase, B., Hemke, A., Johnson, R., Klimentova, X., Kuypers, D., Costa, A.N., Smeulders, B., Spieksma, F., Valentín, M.O., Viana, A., 2021. Modelling and optimisation in european kidney exchange programmes. European Journal of Operational Research 291, 447–456. doi:10.1016/j.ejor.2019.09.006.

Constantino, M., Klimentova, X., Viana, A., Rais, A., 2013. New insights on integer-programming models for the kidney exchange problem. European Journal of Operational Research 231, 57–68. doi:10.1016/j.ejor.2013.05.025.

Delorme, M., García, S., Gondzio, J., Kalcsics, J., Manlove, D., Pettersson, W., Trimble, J., 2022. Improved instance generation for kidney exchange programmes. Computers &amp Operations Research 141, 105707. doi:10.1016/j.cor.2022.105707.

Delorme, M., Manlove, D., Smeets, T., 2023. Half-cycle: A new formulation for modelling kidney exchange problems. Operations Research Letters 51, 234–241.

Desrochers, M., 1988. An algorithm for the shortest path problem with resource constraints. École des hautes études commerciales, Groupe d'études et de recherche en ....

Desrosiers, J., Lübbecke, M.E., 2005. A primer in column generation, in: Column Generation. Springer-Verlag, pp. 1–32. doi:10.1007/0-387-25486-2_1.

Di Puglia Pugliese, L., Guerriero, F., 2013. A survey of resource constrained shortest path problems: Exact solution approaches. Networks 62, 183–200. doi:10.1002/net.21511.

Dickerson, J.P., Manlove, D.F., Plaut, B., Sandholm, T., Trimble, J., 2016. Position-indexed formulations for kidney exchange, in: Proceedings of the 2016 ACM Conference on Economics and Computation, pp. 25–42.

Dror, M., 1994. Note on the complexity of the shortest path models for column generation in vrptw. Operations Research 42, 977–978.

Edmonds, J., 1965. Maximum matching and a polyhedron with 0, 1-vertices. Journal of research of the National Bureau of Standards B 69, 55–56.

Feillet, D., Dejax, P., Gendreau, M., Gueguen, C., 2004. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. Networks 44, 216–229. doi:10.1002/net.20033.

Glorie, K., Wagelmans, A., van de Klundert, J., et al., 2012. Iterative branch-and-price for large multi-criteria kidney exchange. Econometric institute report 11, 2012.

Glorie, K.M., van de Klundert, J.J., Wagelmans, A.P., 2014. Kidney exchange with long chains: An efficient pricing algorithm for clearing barter exchanges with branch-and-price. Manufacturing & Service Operations Management 16, 498–512.

Hoffman, K.L., Padberg, M., 1993. Solving airline crew scheduling problems by branch-and-cut. Management science 39, 657–682.

Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D., 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56, 497–511. doi:10.1287/opre.1070.0449.

Kälble, T., Lucan, M., Nicita, G., Sells, R., Revilla, F.B., Wiesel, M., 2005. Eau guidelines on renal transplantation. European urology 47, 156–166.

Klimentova, X., Alvelos, F., Viana, A., 2014. A new branch-and-price approach for the kidney exchange problem, in: Computational Science and Its Applications–ICCSA 2014: 14th International Conference, Guimarães, Portugal, June 30–July 3, 2014, Proceedings, Part II 14, Springer. pp. 237–252.

Kovesdy, C.P., 2022. Epidemiology of chronic kidney disease: an update 2022. Kidney International Supplements 12, 7–11.

Lam, E., Mak-Hau, V., 2020. Branch-and-cut-and-price for the cardinality-constrained multi-cycle problem in kidney exchange. Computers & Operations Research 115, 104852.

Lentine, K.L., Smith, J.M., Miller, J.M., Bradbrook, K., Larkin, L., Weiss, S., Handarova, D.K., Temple, K., Israni, A.K., Snyder, J.J., 2023. Optn/srtr 2021 annual data report: Kidney. American Journal of Transplantation 23, S21–S120.

Levey, A.S., Coresh, J., 2012. Chronic kidney disease. The lancet 379, 165–180.

Mak-Hau, V., 2015. On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. Journal of Combinatorial Optimization 33, 35–59. doi:10.1007/s10878-015-9932-4.

Marzi, F., Rossi, F., Smriglio, S., 2019. Computational study of separation algorithms for clique inequalities. Soft Computing 23, 3013–3027.

Mattei, N., Walsh, T., 2013. Preflib: A library for preferences http://www. preflib. org, in: Algorithmic Decision Theory: Third International Conference, ADT 2013, Bruxelles, Belgium, November 12-14, 2013, Proceedings 3, Springer. pp. 259–270.

Nemati, E., Einollahi, B., Pezeshki, M.L., Porfarziani, V., Fattahi, M.R., 2014. Does kidney transplantation with deceased or living donor affect graft survival? Nephro-urology monthly 6.

Padberg, M.W., 1973. On the facial structure of set packing polyhedra. Mathematical programming 5, 199–215.

Pansart, L., Cambazard, H., Catusse, N., 2022. Dealing with elementary paths in the kidney exchange problem. arXiv preprint arXiv:2201.08446 .

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., 2017. Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation 9, 61–100.

Petris, M., Archetti, C., Cattaruzza, D., Ogier, M., Semet, F., 2024. Instances and detailed results for the whole testbed of "A Branch-Price-and-Cut algorithm for the Kidney Exchange Problem". URL: https://doi.org/10.57745/IHQAPY, doi:10.57745/IHQAPY.

Plaut, B., Dickerson, J., Sandholm, T., 2016a. Fast optimal clearing of capped-chain barter exchanges, in: Proceedings of the AAAI Conference on Artificial Intelligence.

Plaut, B., Dickerson, J.P., Sandholm, T., 2016b. Hardness of the pricing problem for chains in barter exchanges. arXiv preprint arXiv:1606.00117 .

Rapaport, F., 1986. The case for a living emotionally related international kidney donor exchange registry. Transplantation proceedings 18, 5—9. URL: http://europepmc.org/abstract/MED/11649919.

Riascos-Álvarez, L.C., Bodur, M., Aleman, D.M., 2023. A branch-and-price algorithm enhanced by decision diagrams for the kidney exchange problem. Manufacturing & Service Operations Management .

Roodnat, J., Zuidema, W., van de Wetering, J., de Klerk, M., Erdman, R., Massey, E., Hilhorst, M., IJzermans, J., Weimar, W., 2010. Altruistic donor triggered domino-paired kidney donation for unsuccessful couples from the kidney-exchange program. American Journal of Transplantation 10, 821–827.

Roth, A.E., Sönmez, T., Ünver, M.U., 2004. Kidney exchange. The Quarterly journal of economics 119, 457–488.

Roth, A.E., Sönmez, T., Ünver, M.U., 2005. Pairwise kidney exchange. Journal of Economic theory 125, 151–188.

Roth, A.E., Sönmez, T., Ünver, M.U., 2007. Efficient kidney exchange: Coincidence of wants in markets with compatibility-based preferences. American Economic Review 97, 828–851.

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

Viklicky, O., Krivanec, S., Vavrinova, H., Berlakovich, G., Marada, T., Slatinska, J., Neradova, T., Zamecnikova, R., Salat, A., Hofmann, M., et al., 2020. Crossing borders to facilitate live donor kidney transplantation: the czech-austrian kidney paired donation program–a retrospective study. Transplant International 33, 1199–1210.

Yoo, K.D., Kim, C.T., Kim, M.H., Noh, J., Kim, G., Kim, H., An, J.N., Park, J.Y., Cho, H., Kim, K.H., et al., 2016. Superior outcomes of kidney transplantation compared with dialysis: an optimal matched analysis of a national population-based cohort study between 2005 and 2008 in korea. Medicine 95.