Branch-and-cut-and-price for the Cardinality-constrained Multi-cycle Problem in Kidney Exchange

Edward Lam*[1,2] and Vicky Mak-Hau†[3]

[1]Monash University, Australia
[2]CSIRO Data61, Australia
[3]Deakin University, Australia

September 1, 2019

Abstract

The establishment of kidney exchange programs has dramatically improved rates for kidney transplants by matching donors to compatible patients who would otherwise fail to receive a kidney for transplant. Rather than simply swapping kidneys between two patient-donor pairs, having multiple patient-donors pairs simultaneously donate kidneys in a cyclic manner enables all participants to receive a kidney for transplant. For practicality reasons, the cycles must be limited to short lengths. Finding these cycles can be accomplished by solving the Cardinality-constrained Multi-cycle Problem, which generalizes the Prize-collecting Assignment Problem with constraints that bound the length of the subtours. This paper presents a series of additions to existing works—new constraints, some polyhedral results, new separation algorithms and a new pricing algorithm—and integrates them in the first branch-and-cut-and-price model of the problem. The model is shown to empirically outperform the state-of-the-art by solving 149 of 160 standard benchmarks, compared to 115 by the position-indexed chain-edge formulation and 114 by the position-indexed edge formulation.

*edward.lam@monash.edu
†vicky.mak@deakin.edu.au

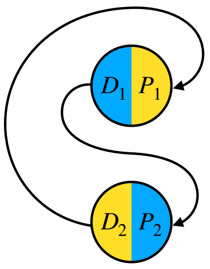

Figure 1: A two-way exchange.

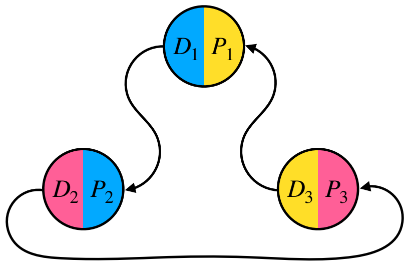

Figure 2: A three-way exchange.

1 Introduction

Chronic kidney disease is difficult to treat. One effective treatment option is to entirely remove and replace a patient's kidney via a transplant. Even though many patients have family and friends who are willing to donate a kidney, the kidney itself might not be compatible with the patient due to many factors such as blood type and tissue type.

Transplant rates can be improved by kidney exchange programs, which match donors to compatible recipients. Figure 1 depicts two patients P_1 and P_2 who respectively have a willing donor D_1 and D_2. D_1 is incompatible with P_1, and likewise, D_2 is incompatible with P_2. The four individuals can enroll in a kidney exchange program, whereby D_2 donates a kidney to P_1 and D_1 donates a kidney to P_2, provided the exchange is compatible. This event, called a two-way exchange is indicated by two patients and two donors. In general, three-way exchanges (Figure 2) and K-way exchanges, for any $K \geq 2$, are possible.

As the number of patients and donors increase and the factors leading to incompatibility grow, it becomes increasingly difficult to match patients and donors by hand. Instead, matchings can be found algorithmically using combinatorial optimization. The problem can be stated on a directed graph. Every incompatible patient-donor pair is given a vertex and every compatible donation is represented by an arc. Every arc is associated with a weight representing some measure of value such as compatibility, chance of success or urgency. A feasible solution to the problem is a set of disjoint cycles that cover the vertices such that each vertex appears in at most one cycle, representing the fact that each patient-donor pair can donate and receive at most one kidney. An optimal solution maximizes the total weight of the cycles. We note that different objective functions are considered in other studies on kidney exchange problems.

The length of the cycles is usually restricted to a small value K, typically $K = 2$ or $K = 3$ (Abraham et al., 2007). This limitation arises from practicality reasons; for example, it may be difficult to ensure a large number of operating theaters and sufficient medical staff are simultaneously available. When $K = \infty$, the problem reduces to the Prize-collecting Assignment Problem, and hence, is polynomially solvable due to its totally unimodular structure (Nemhauser and Wolsey, 1999). The problem is also polynomially solvable when $K = 2$ (Abraham et al., 2007; Biró et al., 2009). In contrast, the problem is NP-hard for any finite $K \geq 3$.

Despite its complexity and difficulty, the problem described above is reasonably solved using mathematical programming. Even though branch-and-cut and branch-and-price have some success, their integration in branch-and-cut-and-price has not yet been studied. This paper presents the first branch-and-cut-and-price formulation of the problem. The contributions are (1) two new constraints that enforce the length of the cycles, (2) a proof that a subset of an existing inequality is facet-defining in a Dantzig-Wolfe reformulation of the problem, (3) new separation algorithms for existing inequalities, (4) a new pricing algorithm, and (5) the integration of various components in the first branch-and-cut-and-price model of the problem.

The remainder of this article is structured as follows. Section 2 presents the mathematical definition of the problem. Section 3 reviews existing models and valid inequalities. Sections 4 and 5 present theoretical and implementation contributions respectively. Section 6 discusses details of the implementation. Section 7 analyzes experimental results. Section 8 proposes ideas for future work. Section 9 concludes this paper.

2 Problem Definition

Two types of kidney exchange are practiced. The problem described in the previous section is known as the Cardinality-constrained Multi-cycle Problem (CCMCP). Another problem seen in kidney exchange is the Cardinality-constrained Cycles and Chains Problem (CCCCP), which additionally allows paths as well as cycles. The paths represent a chain of kidney donations. A path starts with a healthy altruistic donor, who does not receive anything in return, donating a kidney to the patient of one incompatible pair, and the donor from this pair in turn donates their kidney to the patient of another incompatible pair, and so on. Finally, the path terminates at a patient without a donor, e.g., a patient with a recently deceased donor or whose donor becomes a bridging donor in a long chain. The focus of this paper is the CCMCP; we simply mention the CCCCP for completeness. However, most of our contributions are also applicable to the CCCCP but their exact form and/or implementation are a topic for future study. A comprehensive survey of the CCMCP and the CCCCP was conducted by Mak-Hau (2017).

The CCMCP is defined on a directed graph $G = (V, A)$ with vertices $V = \{1, \ldots, |V|\}$ and arcs $A \subseteq \{(i, j) : i, j \in V, i \neq j\}$. Every vertex $i \in V$ represents an incompatible patient-donor pair; that is, donor i has agreed to donate a kidney to recipient i but is physiologically incompatible. The arcs identify all compatible recipients of a donor: the arc $(i, j) \in A$ signifies that donor i can donate to patient j. Every arc (i, j) is associated with a weight $w_{i,j} \in \mathbb{R}_+$ that represents the value of the donation.

An (elementary) path $p = (i_1, i_2, \ldots, i_k)$ of length $k \in \{1, 2, \ldots\}$ is a sequence of k vertices such that every vertex and its next vertex form a valid arc (i.e., $(i_1, i_2), \ldots, (i_{k-1}, i_k) \in A$) and the vertices are not repeated (i.e., $i_s \neq i_t$ for all $s, t \in \{1, \ldots, k\}$ and $s \neq t$). An (elementary) cycle $c = (i_1, i_2, \ldots, i_k, i_{k+1})$ of length k is a sequence of vertices such that $(i_1, i_2, \ldots, i_k)$ is a path, $i_{k+1} = i_1$ and $(i_k, i_1) \in A$. The weight $w_c = \sum_{t=1}^{k} w_{i_t, i_{t+1}} \in \mathbb{R}_+$ of a cycle is the sum of its arc weights. The goal of the CCMCP is to find cycles that have length at most K, for a given $K \in \{2, 3, \ldots\}$, such that every vertex appears in at most one cycle and the total weight of all cycles is maximized. Under this specification, not every vertex needs to belong to a cycle.

3 Related Works

In this section, we review existing models of the CCMCP and various classes of constraints that bound the length of the cycles.

3.1 Arc-based Models

Arc-based models use binary decision variables to select arcs that form cycles on G. These models can be categorized as polynomial-size and exponential-size.

3.1.1 Polynomial-size Models

Polynomial-size models are usually implemented in a way that all problem variables and constraints are included in the beginning, i.e., they are solved without the need for column or row generation. These models have decision variables indexed by at least three dimensions: two indices to represent the arc and one to index the cycle in which the arc belongs. Since cycles are explicitly numbered, permutations of the cycle index leads to symmetry in the model unless deliberately removed.

One example of a polynomial-size model is the extended edge (EE) formulation by Constantino et al. (2013). Let $x_{i,j,l}$ be a binary decision variable indicating whether the arc $(i, j) \in A$ belongs in the lth cycle, where $l \in L = \{1, \ldots, |L|\}$

and $|L|$ is an upper bound on the number of cycles, e.g., $|L| = |V|$. The EE model is shown in Problem (1).

Problem (1)

$$\max \sum_{l \in L} \sum_{(i,j) \in A} w_{i,j} x_{i,j,l} \tag{1a}$$

subject to

$$\sum_{l \in L} \sum_{j:(i,j) \in A} x_{i,j,l} \leq 1 \qquad \forall i \in V, \tag{1b}$$

$$\sum_{h:(h,i) \in A} x_{h,i,l} = \sum_{j:(i,j) \in A} x_{i,j,l} \qquad \forall i \in V, l \in L, \tag{1c}$$

$$\sum_{(i,j) \in A} x_{i,j,l} \leq K \qquad \forall l \in L, \tag{1d}$$

$$x_{i,j,l} \in \mathbb{Z}_+ \qquad \forall (i,j) \in A, l \in L. \tag{1e}$$

The Objective Function (1a) maximizes the total weight of the selected arcs. Constraint (1b) clones the graph into $|L|$ copies, and requires that every vertex is selected in at most one copy of the graph. Constraint (1c) requires the in-degree of every vertex to be equal to its out-degree. Constraint (1d) allows at most K arcs to be selected in the lth copy of the graph. Notice that more than one cycle can be selected in each copy, but this constraint is valid under the problem definition. Constraint (1e) is the integrality constraint. For reasons described later, the variables are specified as integer variables, rather than binary variables. In any case, the variables can only take binary values due to Constraint (1b). The EE formulation solved blood-type test instances with up to 100 vertices and K ranging from 4 to 6.

This formulation is closely related to the three-index formulations of the Vehicle Routing Problems (VRPs) (Vigo and Toth, 2014). The CCMCP can be considered as a variation on the Prize-collecting Capacitated Vehicle Routing Problem in which every vehicle has capacity K, each customer has load 1 and vehicles can start and end at any vertex, not only at a central depot vertex.

The EE model is shown experimentally to outperform prior arc-based models (e.g., the two-index model in Section 3.1.2), despite being theoretically proven to not dominate the other models in its linear relaxation bound. Such results are unusual, and contradicts well-established results from VRPs showing that three-index models are inferior to two-index formulations (e.g., Letchford and Salazar-González, 2006).

Dickerson et al. (2016) recently improved the EE formulation in the position-indexed edge formulation (PIEF), which uses binary variables to determine whether an arc appears in a particular position of a cycle in a particular copy of the digraph—a concept that is an extension to the EE formulation. The model is also theoretically proven to have a stronger linear relaxation bound. A polynomial-time algorithm for variable elimination in preprocessing was discussed and implemented. An extension of PIEF to the CCCCP was extensively tested on large-scale problem instances with over 700 vertices and with $K = 3$ and chain lengths varying from 2 to 12. This model is shown to substantially outperform many other models.

3.1.2 Exponential-size Models

Exponential-size models use binary variables indexed by two dimensions to denote the use of an arc. These models contain exponentially many constraints to enforce the length of the cycles. Because of this, these models are typically solved using branch-and-cut.

The branch-and-cut method iterates between solving a master problem to obtain a candidate solution, and solving separation problems on the given candidate solution to identify violated constraints and add them to the master problem (e.g., Mitchell, 2010).

The two-index branch-and-cut model of the CCMCP has binary decision variables $x_{i,j}$ that indicate if $(i, j) \in A$ is selected. The initial master problem is given in Problem (2).

Problem (2)

$$\max \sum_{(i,j)\in A} w_{i,j} x_{i,j} \tag{2a}$$

subject to

$$\sum_{j:(i,j)\in A} x_{i,j} \leq 1 \qquad \forall i \in V, \tag{2b}$$

$$\sum_{h:(h,i)\in A} x_{h,i} = \sum_{j:(i,j)\in A} x_{i,j} \qquad \forall i \in V, \tag{2c}$$

$$x_{i,j} \in \mathbb{Z}_+ \qquad \forall (i, j) \in A. \tag{2d}$$

Objective Function (2a) maximizes the total weight of the selected arcs. Constraint (2b) allows at most one outgoing edge for every vertex. Constraint (2c) ensures that every vertex that has an incoming edge also has an outgoing edge. Constraint (2d) enforces integrality.

Since Problem (2) closely resembles the two-index formulation of the Prize-Collecting Capacitated Vehicle Routing Problem (Vigo and Toth, 2014), we call it the Prize-collecting Assignment Problem. Again, it differs in the lack of a central depot vertex. This small difference invalidates all of the cutting planes that generalize subtour elimination, e.g., rounded capacity cuts, 2-path cuts, etc.

Problem (2) itself does not bound the length of the cycles. Instead, constraints are added during the solution process to enforce the cycle length constraints whenever they are violated by the current feasible solution, i.e., to ensure solutions of the Prize-collecting Assignment Problem are also solutions of the CCMCP. Cycle lengths can be restricted using any one of the following families of constraints.

Combinatorial Benders A Combinatorial Benders cut disallows one variable from any given set of binary variables (Codato and Fischetti, 2004). Roth et al. (2007); Anderson et al. (2015) applied combinatorial Benders cuts to the CCMCP. Whenever a solution contains a cycle $c = (i_1, \ldots, i_k, i_1)$ of length $k > K$, Constraint (3) is added to Problem (2) in order to forbid c by requiring at least one of the k edges of c to be unused.

$$\sum_{t=1}^{k} x_{i_t, i_{t+1}} \leq k - 1 \tag{3}$$

TNT For instances with $K = 3$, let $p = (i_1, i_2, i_3, i_4)$ be a path of length 4. Then, the constraints

$$\sum_{t=1}^{3} x_{i_t, i_{t+1}} + 3x_{i_4, i_1} + 2x_{i_2, i_1} + 2x_{i_3, i_1} + 2x_{i_4, i_2} \leq 4, \tag{4a}$$

and

$$\sum_{t=1}^{3} x_{i_t, i_{t+1}} + 3x_{i_4, i_1} + 2x_{i_4, i_2} + 2x_{i_4, i_3} + 2x_{i_3, i_1} \leq 4, \tag{4b}$$

are proven valid by Mak-Hau (2018). Based on their coefficients, we call Constraints (4a) and (4b) the Three and Twos (TNT) cuts. These inequalities are generalized to larger K in Section 4.1.

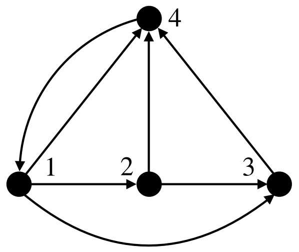

Figure 3: A sailboat cut for $K = 4$.

Tournament Given a cardinality-infeasible path $p = (i_1, \dots, i_K, i_{K+1})$ of length $K + 1$, the constraint

$$\sum_{s=1}^{K} \sum_{t=s+1}^{K+1} x_{i_s,i_t} \leq K - 1, \tag{5}$$

called a tournament inequality, forbids at least one of the arcs between any pair of vertices in p. Ascheuer et al. (2000) proposed the tournament inequalities for the Asymmetric Traveling Salesman Problem, which is later shown to be valid and facet-defining for the CCMCP by Mak-Hau (2018).

Sailboat Mak-Hau (2018) showed that for any path $p = (i_1, \dots, i_K)$ of length K, the constraint

$$\sum_{s=1}^{K-1} \sum_{t=s+1}^{K} x_{i_s,i_t} - x_{i_K,i_1} \leq K - 2 \tag{6}$$

is valid and facet-defining. We call Constraint (6) the sailboat inequalities, based on the illustration in Figure 3.

3.2 Cycle-based Models

Instead of selecting arcs to form cycles, cycle-based formulations select a subset of cycles from a large collection C of valid cycles. There are up to $\sum_{j=2}^{K} \binom{|V|}{j} (j-1)!$ elements in C. Hence, it is extremely difficult, and usually impossible, to generate C on large and/or dense instances in practice. Therefore, cycle-based approaches are solved using column generation and branch-and-price.

Column generation is a method for solving linear programs with a large number of variables. Instead of handing a large matrix to a linear programming solver, column generation progressively adds variables to a linear program until a point when all the variables taking non-zero value in some optimal solution appears in the linear program. Branch-and-price is a branch-and-bound algorithm that computes a dual bound in every node of the branch-and-bound tree using a linear relaxation solved using column generation.

In cycle-based models, column generation begins with a small (possibly empty) restricted subset $C' \subseteq C$ of cycles, and then iterates between a restricted master problem, which is a linear relaxation that selects a fractionally-optimal subset of cycles from C', and pricing problems, which fill C' with better cycles to bring C' closer towards a subset of C sufficient for proving optimality in the restricted master problem (e.g., Desaulniers et al., 2005; Lübbecke and Desrosiers, 2005).

Restricted Master Problem Problem (7) describes the integer master problem, a set packing problem. The restricted master problem discussed above is its linear relaxation. Every cycle $c \in C$ is associated with a weight $w_c \in \mathbb{R}_+$ defined in Section 2. A binary decision variable y_c is used to indicate whether $c \in C'$ is selected.

Problem (7)

$$\max \sum_{c \in C'} w_c y_c \tag{7a}$$

subject to

$$\sum_{c \in C'} a_c^i y_c \leq 1 \qquad \forall i \in V, \tag{7b}$$

$$y_c \in \mathbb{Z}_+ \qquad \forall c \in C'. \tag{7c}$$

Objective Function (7a) maximizes the total weight of the cycles selected from C'. Constraint (7b) requires every vertex to be visited at most once. The constant $a_c^i \in \{0, 1\}$ takes value 1 if cycle $c \in C'$ visits $i \in V$ and takes value 0 otherwise. Constraint (7c) is the integrality constraint.

Pricing Problem The set C' is filled by solving the pricing problem formalized in Problem (8).

Problem (8)

$$z := \max \sum_{(i,j) \in A} (w_{i,j} - \pi_i) x_{i,j} \tag{8a}$$

subject to

$$\sum_{j:(i,j)\in A} x_{i,j} \leq 1 \qquad \forall i \in V, \quad (8b)$$

$$\sum_{h:(h,i)\in A} x_{h,i} = \sum_{j:(i,j)\in A} x_{i,j} \qquad \forall i \in V, \quad (8c)$$

$$\sum_{(i,j)\in A} x_{i,j} \leq K, \qquad (8d)$$

$$x_{i,j} \in \mathbb{Z}_+ \qquad \forall (i,j) \in A. \quad (8e)$$

Objective Function (8a) finds cycles with maximum reduced cost $\bar{w} = w_{i,j} - \pi_i$, where π_i is the value of the dual variable of Constraint (7b). Notice that y_c in the integer master problem is specified as an integer variable, rather than a binary variable. This alleviates the need to consider an additional dual variable in Objective Function (8a) arising from the implicit upper bound constraint. In any case, y_c is naturally restricted to take binary values by Constraint (7b).

Observe that Constraints (8b) to (8e) are nearly identical to Constraints (1b) to (1e). This is not a coincidence; these constraints emerge from a Dantzig-Wolfe reformulation of Problem (1). A solution to Problem (8) is equivalent to the cycles chosen in any one of the l copies of the graph in Problem (1). Recall that any one copy of the graph in Problem (1) can contain multiple cycles. After solving Problem (8), each cycle must be isolated in a post-processing step.

Even though the pricing problem is stated as a mixed integer program, in practice, it is often solved using algorithms tailored to the problem. Roth et al. (2007) used mixed integer programming. Abraham et al. (2007) used heuristics and exhaustive search to solve instances with up to 10,000 vertices but arc density information is not reported. Glorie et al. (2014) solved a cycle-based model using U.S. population data with up to 1000 vertices. Since long chains are allowed, this is not a direct comparison to the CCMCP. Nonetheless, their Bellman-Ford dynamic programming pricer is later proved incorrect by Plaut et al. (2016). Corrections are implemented by Dickerson et al. (2016). Klimentova et al. (2014) used heuristics on a different master problem to solve instances with up to 2000 vertices with $K = 3$, 1000 vertices with $K = 4$, and 900 vertices with $K = 5$ and $K = 6$. Arc densities were again not specified. The position-indexed chain-edge formulation (PICEF) by Dickerson et al. (2016) models the CCCCP but reduces to the CCMCP in the absence of chains. In PICEF, all cycles are enumerated as variables in the master problem, which is given to a black-box mixed integer programming solver; thus avoiding the need for pricing.

Column Generation Column generation begins by solving the restricted master problem to produce a primal solution and values to π_i. It then proceeds to solve the pricing problem, resulting in a value for z and a cycle c given by the values to the $x_{i,j}$ variables. Whenever $z > 0$, c can potentially appear in a solution better than the current solution to the restricted master problem. Therefore, c is added to C' and the process returns to the restricted master problem, which may or may not select the new cycle c. Column generation terminates when the pricing problem reports that $z \leq 0$, indicating that all cycles that could appear in an optimal solution already exist in C', i.e., none are in $C \setminus C'$.

3.3 Branch-and-cut-and-price

Branch-and-cut-and-price provides two complementary views into a problem by integrating branch-and-cut and branch-and-price. For this reason, it powers state-of-the-art algorithms for many graph optimization problems, including the Traveling Salesman Problem (Applegate et al., 2006), Vehicle Routing Problem with Time Windows (Røpke, 2012), Capacitated Vehicle Routing Problem (Pecin et al., 2014) and Multi-agent Path Finding (Lam et al., 2019). Despite impressive successes, branch-and-cut-and-price has not yet been implemented for CCMCP; hence the motivation of this study.

In general, branch-and-cut-and-price takes the master problem of an existing branch-and-price model and progressively adds cutting planes to tighten its linear relaxation. Fukasawa et al. (2006) categorized cuts in branch-and-cut-and-price as either robust or non-robust.

3.3.1 Robust Cuts

Consider a general constraint in the two-index arc-based model Problem (2):

$$\sum_{(i,j)\in A} a_{i,j} x_{i,j} \leq b, \tag{9}$$

where $a_{i,j} \in \mathbb{R}$ and $b \in \mathbb{R}$ are constants. This cut can be convexified to

$$\sum_{c\in C'} \left(\sum_{(i,j)\in A} a_{i,j} a_c^{i,j} \right) y_c \leq b, \tag{10}$$

in the master problem (Problem (7)), where $a_c^{i,j} \in \{0, 1\}$ takes the value 1 if cycle c traverses the arc (i, j) and takes the value 0 otherwise. Via this translation, all cuts valid in the branch-and-cut model are automatically valid in the branch-and-price model.

Let ρ be the dual variable of Constraint (10). To account for one additional Constraint (10) in the master problem, Objective Function (8a) is modified to

$$\max \sum_{(i,j)\in A} (w_{i,j} - \pi_i - a_{i,j}\rho) x_{i,j}.$$

An inequality taking the form of Constraint (10) is called *robust* (Fukasawa et al., 2006) since it only modifies the objective function of the pricing problem by adding one additional term to the reduced cost. Now,

$$\bar{w}_{i,j} = w_{i,j} - \pi_i - a_{i,j}\rho.$$

Even though all robust cuts from branch-and-cut are valid in branch-and-price, some families are rendered useless since they are implicitly described within the definition of the variables in the Dantzig-Wolfe reformulation (e.g., Letchford and Salazar-González, 2006; Røpke and Cordeau, 2009). The following example shows that the sailboat inequalities are not implied by the reformulation and can be violated in a feasible solution to the linear relaxation of the master problem. Examples for the TNT and tournament cuts can be constructed in a similar fashion.

Proposition 1. The Dantzig-Wolfe reformulation does not imply the sailboat inequalities.

Proof. Consider the CCMCP with $K = 3$. Figure 4 shows an example of four cycles $c_1, \dots, c_4$ over six vertices labeled 1 to 6. The linear relaxation of the master problem containing these four cycles is given below:

$$\begin{array}{lccccccccc}
\text{max} & w_1y_1 & + & w_2y_2 & + & w_3y_3 & + & w_4y_4 & & \\
\text{subject to} & & & & & & & y_4 & \leq & 1 \\
& & & y_2 & + & y_3 & & & \leq & 1 \\
& y_1 & + & y_2 & & & & & \leq & 1 \\
& y_1 & & & + & y_3 & & & \leq & 1 \\
& y_1 & + & y_2 & & & & & \leq & 1 \\
& & & & & y_3 & + & y_4 & \leq & 1 \\
& y_1, & & y_2, & & y_3, & & y_4 & \in & \mathbb{R}_+.
\end{array}$$

Consider the path $p = (3, 4, 2)$. This path induces the sailboat cut

$$x_{3,4} + x_{3,2} + x_{4,2} - x_{2,3} \leq 1,$$

which is convexified to

$$y_1 + y_2 + y_3 - 0 \leq 1.$$

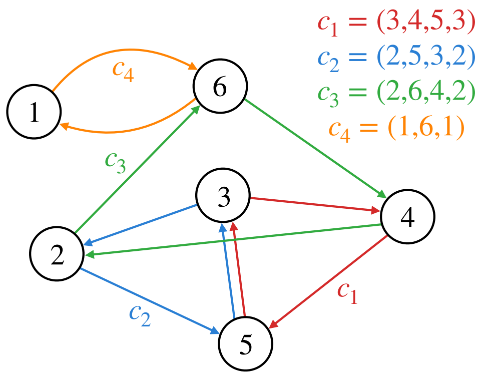

Figure 4: Example of four cycles.

Next, consider the feasible solution $\hat{\mathbf{y}} = (0.5, 0.5, 0.5, 0.5)$. Substituting this solution into the above cut yields

$$0.5 + 0.5 + 0.5 - 0 \leq 1.$$

This solution violates the cut, and hence, the family of sailboat cuts is not implied by the reformulation. □

3.3.2 Non-robust Cuts

Cuts are classified as *non-robust* if they are valid for the master problem but cannot be expressed as a linear combination of the arc variables, i.e., Constraint (10). In general, adding a non-robust cut to the master problem introduces a dual variable that is difficult to handle in the pricing problem and results in increased time complexity (Jepsen et al., 2008).

Below, we review two classes of non-robust cuts as prerequisites for later discussions. We do not use them in the branch-and-cut-and-price model because an efficient implementation and detailed analysis of their performance are beyond the scope of this initial study. In Section 8, we discuss their potential for future work.

Clique For a set $W \subseteq C$, clique inequalities specify that every cycle $c \in W$ conflicts with every other cycle $c' \in W$, where $c' \neq c$. For a clique W, the clique cut is

$$\sum_{c \in W} y_c \leq 1. \tag{11}$$

Clique constraints are facets of the convex hull of the set packing polyhedron (Nemhauser and Wolsey, 1999), i.e., they are facet-defining on the master problem (Constraints (7b) and (7c)).

For a VRP, Spoorendonk and Desaulniers (2010) showed empirically that clique inequalities reduce the number of branch-and-bound nodes but increase the total run-time due to the additional dual variables.

Subset-row Jepsen et al. (2008) developed another class of non-robust cuts, called the subset-row inequalities, for master problems with a set packing structure. Subset-row cuts are Chvatal-Gomory rank-1 cuts, defined as

$$\sum_{c\in C}\left\lfloor \frac{1}{k}\sum_{i\in S} a_c^i \right\rfloor y_c \leq \left\lfloor \frac{|S|}{k} \right\rfloor, \tag{12}$$

where $S \subseteq V$ and $k \in \{1, \ldots, |S|\}$. In the special case of $|S| = 3$ and $k = 2$ (as well as others), Constraint (12) reduces to Constraint (11) where

$$W = \left\{ c \in C : \left\lfloor \frac{1}{2}\sum_{i\in S} a_c^i \right\rfloor = 1 \right\},$$

and therefore, is facet-defining. Unlike the clique inequalities, experiments on a VRP showed that the subset-row cuts parameterized by $|S| = 3$ and $k = 2$ does improve computational run-time, despite using a separation algorithm that enumerates all such subsets S.

4 Theoretical Contributions

This section presents several theoretical contributions regarding constraints used in the branch-and-cut-and-price model of the CCMCP.

4.1 Generalizing the TNT Constraints

The TNT constraints were developed specifically for the case of $K = 3$. They are generalized to any $K \geq 3$ as follows. Let $p = (i_1, \ldots, i_K, i_{K+1})$ be a minimally cardinality-violated path of length $K + 1$, then Constraints (4a) and (4b) can be generalized to

$$\sum_{j=1}^{K} x_{i_j,i_{j+1}} + 3x_{i_{K+1},i_1} + 2\sum_{j=2}^{K} x_{i_j,i_1} + 2x_{i_{K+1},i_2} \leq K + 1 \tag{13a}$$

and

$$\sum_{j=1}^{K} x_{i_j,i_{j+1}} + 3x_{i_{K+1},i_1} + 2\sum_{j=2}^{K} x_{i_{K+1},i_j} + 2x_{i_K,i_1} \leq K+1. \tag{13b}$$

Proposition 2. Constraints (13a) and (13b) are valid for the CCMCP.

Proof. Notice that both constraints have $\sum_{j=1}^{K} x_{i_j,i_{j+1}} + 3x_{i_{K+1},i_1} + 2x_{i_K,i_1} + 2x_{i_{K+1},i_2}$ in common in the left hand side. Hence, we rewrite Constraint (13a) as below and prove its validity by sequential lifting. Consider

$$\sum_{j=1}^{K} x_{i_j,i_{j+1}} + \alpha x_{i_{K+1},i_1} + \beta x_{i_K,i_1} + \gamma x_{i_{K+1},i_2} + \delta \sum_{\ell=2}^{K-1} x_{i_\ell,i_1} \leq K+1$$

First, we consider lifting α. When $x_{i_{K+1},i_1} = 1$, $\sum_{j=1}^{K} x_{i_j,i_{j+1}} \leq K-2$, otherwise a cardinality violation is induced, hence α can be lifted to 3.

When $x_{i_K,i_1} = 1$, $x_{i_{K+1},i_1} = 0$, and $x_{i_K,i_{K+1}} = 0$, so $\sum_{j=1}^{K} x_{i_j,i_{j+1}} = \sum_{j=1}^{K-1} x_{i_j,i_{j+1}} \leq K-1$, and therefore $\beta = 2$.

When $x_{i_{K+1},i_2} = 1$, clearly $x_{i_1,i_2} = 0$ and $x_{i_{K+1},i_1} = 0$. When $x_{i_K,i_1} = 0$, then $\sum_{j=2}^{K} x_{i_j,i_1} \leq K-1$, and hence $\gamma = 2$. When $x_{i_K,i_1} = 1$, then $\sum_{j=2}^{K-1} x_{i_j,i_1} \leq K-3$, otherwise a cardinality violation is induced, so $\gamma = 2$ as well.

Finally, we lift δ, If $x_{i_\ell,i_1} = 1$ for any $\ell = 2, \ldots, K-1$, we have that $x_{i_{\ell,\ell+1}} = x_{i_K,i_1} = x_{i_{K+1},i_1} = 0$. When $x_{i_{K+1},i_2} = 0$, $\sum_{j=1}^{\ell-1} x_{i_j,i_{j+1}} + \sum_{j=\ell+1}^{K} x_{i_j,i_{j+1}} \leq K-1$, so $\delta = 2$. When $x_{i_{K+1},i_2} = 1$, $\sum_{j=2}^{\ell-1} x_{i_j,i_{j+1}} + \sum_{j=\ell+1}^{K} x_{i_j,i_{j+1}} \leq K-3$ otherwise a cardinality violation is induced, hence $\delta = 2$ as well. This gives us Constraint (13a). The proof for Constraint (13b) is similar.

We now consider lifting η in the inequality below.

$$\sum_{j=1}^{K} x_{i_j,i_{j+1}} + 3x_{i_{K+1},i_1} + 2x_{i_K,i_1} + 2x_{i_{K+1},i_2} + \eta \sum_{\ell=3}^{K} x_{i_{K+1},i_\ell} \leq K+1$$

If $x_{i_{K+1},\ell} = 1$ for any $\ell = 3, \ldots, K$, we have that $x_{i_{\ell-1,\ell}} = x_{i_{K+1,1}} = x_{i_{K+1,2}} = 0$. When $x_{i_K,i_1} = 0$, $\sum_{j=1}^{\ell-2} x_{i_j,i_{j+1}} + \sum_{j=\ell}^{K} x_{i_j,i_{j+1}} \leq K-1$, so $\eta = 2$. Otherwise, $\sum_{j=1}^{\ell-2} x_{i_j,i_{j+1}} + \sum_{j=\ell}^{K} x_{i_j,i_{j+1}} \leq K-3$ to avoid a cardinality violation, so $\eta = 2$ either way. □

4.2 Sailboat Facets

Recall that the sailboat inequalities are facets of the convex hull of the integer points induced by the two-index model described in Problem (2). A subset of their convexifications are also facets of the master problem in branch-and-cut-and-price, as discussed below.

Proposition 3. For $K = 3$, the convexification of Constraint (6) induced by a path $p = (i_1, i_2, i_3)$ such that $(i_3, i_1) \notin A$ is a facet of the convex hull of Constraints (7b) and (7c).

Proof. For $K = 3$ and a path $p = (i_1, i_2, i_3)$ such that $(i_3, i_1) \notin A$, Constraint (6) specializes to

$$x_{i_1,i_2} + x_{i_1,i_3} + x_{i_2,i_3} \leq 1.$$

Convexifying this constraint leads to

$$\sum_{c \in C} (a_c^{i_1,i_2} + a_c^{i_1,i_3} + a_c^{i_2,i_3}) y_c \leq 1. \tag{14}$$

No cycle can simultaneously use the edges (i_1, i_2) and (i_1, i_3) as it would require the cycle to have two outgoing edges from i_1. Similarly, (i_1, i_3) and (i_2, i_3) cannot be used simultaneously. Since $K = 3$, if (i_1, i_2) and (i_2, i_3) are used simultaneously, they must belong to the cycle (i_1, i_2, i_3, i_1), which is not valid because $(i_3, i_1) \notin A$. Hence, (i_1, i_2) and (i_2, i_3) also cannot be used simultaneously. Consequently, $a_c^{i_1,i_2} + a_c^{i_1,i_3} + a_c^{i_2,i_3} \in \{0, 1\}$, and so, Constraint (14) is a clique inequality and is facet-defining on the master problem. □

5 Implementation Contributions

This section presents several contributions on the implementation of the branch-and-cut-and-price model of the CCMCP.

5.1 A Separator for the Sailboat Cuts for $K = 3$

The sailboat cut specialized to $K = 3$ is

$$x_{i_1,i_2} + x_{i_2,i_3} + x_{i_1,i_3} - x_{i_3,i_1} \leq 1 \tag{15}$$

for some path $p = (i_1, i_2, i_3)$. Proposition 4 argues that this constraint can only be violated if $x_{i_1,i_3} > 0$.

Proposition 4. Constraint (15) is violated in the master problem only if $x_{i_1,i_3} > 0$.

Proof. In the case of $x_{i_1,i_3} = 0$ (or $(i_1, i_3) \notin A$), Constraint (15) realized as

$$x_{i_1,i_2} + x_{i_2,i_3} - x_{i_3,i_1} \leq 1,$$

which cannot be violated. In order for a violation, $x_{i_1,i_3} > 0$. □

Algorithm 1 presents a novel separation algorithm for Constraint (15). In step 1, the separator recovers a feasible solution $\hat{\mathbf{x}}$ to the linear relaxation of the two-index model equivalent to the solution $\hat{\mathbf{y}}$ of the linear relaxation of the master problem by projection. By Proposition 4, step 2 loops through all edges that skip i_2 in p. Step 3 finds a vertex $i_2 \in V$ that forms p. Step 4 computes the left-hand side of Constraint (15). Step 5 determines if the constraint is violated. If so, step 6 adds a row to the master problem.

Algorithm 1: Separator for sailboat cuts when $K = 3$

```
1 $\hat{x}_{i,j} \leftarrow \sum_{c \in C} a_c^{i,j} \hat{y}_c$ for all $(i,j) \in A$
2 for all $(i_1, i_3) \in A$ such that $\hat{x}_{i_1,i_3} > 0$
3     for all $i_2 \in V$ such that $(i_1, i_2) \in A \wedge (i_2, i_3) \in A$
4         $lhs \leftarrow \hat{x}_{i_1,i_2} + \hat{x}_{i_2,i_3} + \hat{x}_{i_1,i_3} - \hat{x}_{i_3,i_1}$
5         if $lhs > 1$ then
6             create Constraint (15) induced by $(i_1, i_2, i_3)$
7         end
8     end
9 end
```

5.2 A Separator for the Sailboat Cuts for $K > 3$

In the original work, sailboat inequalities were separated by exhaustively generating all cycles and then determining whether they induce a constraint that is violated by the current solution of the linear relaxation. We develop a new exact separator based on depth-first tree search to find paths p that induce a violated sailboat cut.

The pseudocode is provided in Algorithm 2. Step 1 calculates the same projection as before. Steps 2 and 3 construct a subgraph whose edges have positive value in the solution of the linear relaxation. Step 4 loops through all vertices as the root in the depth-first search. Step 5 initializes an array storing the vertices of p. Steps 6 and 7 initialize a stack of pairs of a vertex and its depth in the depth-first search. Step 8 is the main loop. Step 9 pops an element off the stack. Step 10 appends the vertex to the partial path. Steps 11 to 14 create a row in the master problem if the vertices in the array p specify a path inducing a violated sailboat inequality. Steps 15 to 17 continue the depth-first search while the length is not violated.

5.3 Separators for the Tournament and TNT Cuts

The remaining cuts are separated using a depth-first search algorithm similar to Algorithm 2. In previous work, they were also separated by exhaustively generating

Algorithm 2: Separator for sailboat cuts when $K > 3$

```
$\hat{x}_{i,j} \leftarrow \sum_{c \in C} a_c^{i,j} \hat{y}_c$ for all $(i,j) \in A$
$A' \leftarrow \{(i,j) \in A : \hat{x}_{i,j} > 0\}$
$G' \leftarrow (V, A')$
for all $s \in V$
    $p \leftarrow$ empty array of length $K$
    $stack \leftarrow$ empty stack storing $(i,k) \in V \times \{1, \dots, K\}$
    $stack.push((s,1))$
    while $\neg stack.empty()$
        $(i,k) \leftarrow stack.pop()$
        $p[k] \leftarrow i$
        if $k = K$ then
            $lhs \leftarrow \sum_{s=1}^{K-1} \sum_{t=s+1}^{K} \hat{x}_{p[s],p[t]} - \hat{x}_{p[K],p[1]}$
            if $lhs > K - 2$ then
                create Constraint (6) induced by $(p[1], \dots, p[K])$
            end
        end
        if $k < K$ then
            for all $j \in V$ such that $(i,j) \in A \wedge j \notin \{p[1], \dots, p[k]\}$
                $stack.push((j, k+1))$
            end
        end
    end
end
```

the path inducing the constraint and then checking whether the constraint is violated.

5.4 A Heuristic Pricer

Recall that the objective value of the solution to the linear relaxation of the master problem is a valid dual bound if and only if an exact pricer proves that every column not yet in C' has non-positive reduced cost. Even though a complete pricer is necessary for obtaining a dual bound, faster and/or polynomial-time heuristic pricers can be used to find columns with positive reduced cost since adding any such column will change the basis, moving one step closer towards linear optimality. Only when the heuristic pricers fail to find columns with positive reduced cost must a (usually exponential-time) complete pricer be called to verify that no additional columns improve the objective value.

The discussion below presents a cubic-time heuristic pricing algorithm, and proves that it is, in fact, exact when $K = 3$. The pricer finds elementary cycles with positive reduced cost by solving longest path problems on directed acyclic graphs (DAGs) using a label setting algorithm. The pricer loops through every vertex $s \in V$ and attempts to find a longest path from the source vertex s to a duplicated sink vertex s in a position-expanded DAG $\bar{G}(s)$ induced by s.

The linear-time longest path algorithm for a DAG is well-established, and hence, is not repeated here. However, the graph $\bar{G}(s)$ upon which it is called is uniquely designed for the CCMCP. The graph $\bar{G}(s)$ is constructed by expanding the original graph G in a manner similar to the time-expanded graphs that appear in many optimization problems. For a vertex $s \in V$, define $\bar{G}(s) = (\bar{V}(s), \bar{A}(s))$, where

$$\begin{aligned}\bar{V}(s) = &\{(s,1)\} \cup \\ &\{(i,k) : i > s, i \in V, k \in \{2,\dots,K\} \cup \\ &\{(s,K+1)\}\end{aligned}$$

and

$$\begin{aligned}\bar{A}(s) = &\{((s,1),(i,2)) : i > s, (s,i) \in A\} \cup \\ &\{((i,k),(j,k+1)) : i > s, j > s, (i,j) \in A, k \in \{2,\dots,K-1\}\} \cup \\ &\{((i,k),(s,K+1)) : i > s, (i,s) \in A, k \in \{2,\dots,K\}\}.\end{aligned}$$

Associate with every arc $((i,k),(j,k+1)) \in \bar{A}(s)$ a weight $\bar{w}_{((i,k),(j,k+1))} = \bar{w}_{i,j}$. Figure 5 illustrates an example of constructing $\bar{G}(s)$. Restricting the search to only vertices numbered greater than the source vertex ensures that different permutations of the same cycle are not repeated (Johnson, 1975).

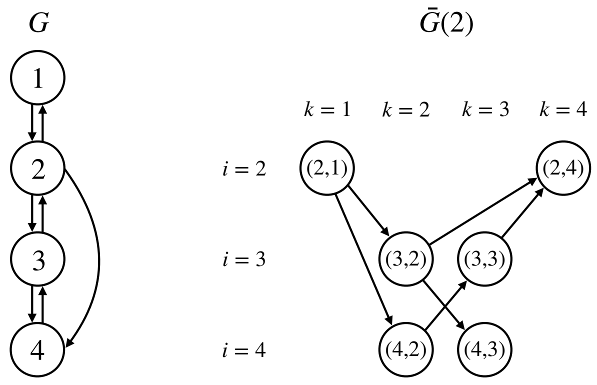

Figure 5: Example of a graph G and $\bar{G}(2)$ for $K = 3$.

The pricer then simply finds a longest path from the source $(s, 1)$ to the sink $(s, K + 1)$. By exploiting the topological ordering of the acyclic structure, every vertex and every edge is considered exactly once. Therefore, a longest path can be found in time linear in the number of vertices and edges in $\bar{G}(s)$, i.e., quadratic time $O(VK + AK)$ due to the position-expansion. Since the heuristic pricer loops through every source vertex $s \in V$, it has overall cubic complexity $O(V^2K + VAK)$.

The heuristic pricer may find paths that visit (i, k) and (i, k') for some $k' \neq k$, i.e., a non-elementary cycle that visits i more than once. In this case, it simply discards the cycle; therefore, it is incomplete. The proposition below argues that cycles are never discarded when $K = 3$, and hence, the heuristic pricer is actually complete for $K = 3$.

Proposition 5. The heuristic pricer is exact for $K = 3$.

Proof. In the case of $K = 3$, the algorithm begins by expanding from the source $(s, 1)$ towards some vertex $(i, 2)$, i.e., the vertex i in the second position of a cycle. Next, it either closes the cycle by expanding $(i, 2)$ to the sink $(s, 4)$ or continues to some other vertex $(j, 3)$, at which point it can only expand to the sink. Since $(i, i) \notin A$, then $j \neq i$, and hence, i and j appear at most once in the cycle. Therefore, the heuristic pricer exactly solves Problem (8) when $K = 3$. □

6 The Implementation

The implementation of the branch-and-cut-and-price model of the CCMCP includes all the components introduced in Sections 4 and 5. This section discusses the remaining components not yet presented. These components are not novel.

6.1 A Complete Pricer

We implement a basic exact pricer that solves the longest path problem with resource constraints using exhaustive enumeration similar to the pricer invented by Abraham et al. (2007). It first computes the reduced cost $\bar{w}_{i,j}$ of every arc $(i, j) \in A$. Next, it loops through each vertex $s \in V$ as the source. Let $\tilde{G}(s) = (\tilde{V}, \tilde{A})$ be a subgraph of G where $\tilde{V} = \{s, \ldots, |V|\}$ and $\tilde{A} = \{(i, j) \in A : i, j \in \tilde{V}\}$. That is, $\tilde{G}(s)$ contains the vertices of G with an index at least s.

The pricer then searches $\tilde{G}(s)$ using a trie rooted at s. Whenever s appears as a leaf with depth between 3 and K, the path from the root to the leaf represents a cycle $c = (s, \ldots, s)$. If the sum of the reduced costs of its edges is positive, it can potentially improve the current master problem solution, and hence, is added as a new column. Using depth-first search, every possible cycle is examined for a positive reduced cost.

6.2 Branching Rule

It is well-known that branching on the cycle variables of the Dantzig-Wolfe master problem causes considerable difficulty in the pricing problem. Therefore, it is common to branch on the variables of the original arc-based model. We implement a branching rule that branches on these arc variables using pseudocosts (Benichou et al., 1971; Achterberg et al., 2005).

Every arc $(i, j) \in A$ is associated with a down pseudocost and an up pseudocost, which represent some measure of the improvement to the dual bound upon fixing $x_{i,j} = 0$ and $x_{i,j} = 1$ respectively. After optimizing the linear relaxation of the root node, the pseudocosts are initialized using strong branching, which tentatively branches on fractional arc to compute their down and up pseudocosts. Since there may be a large number of fractional arcs, we perform strong branching only on a subset of the arcs, with the down and up pseudocosts of the remaining arcs initialized to their average value.

7 Experimental Results

This section presents experimental results.

7.1 Experiments Set-up

The branch-and-cut-and-price (BCP) model is implemented in SCIP 6.0.2 with CPLEX 12.9 as the underlying linear programming solver. The model includes the TNT, tournament and sailboat inequalities, the heuristic pricer introduced in

Section 5.4, the exact pricer mentioned in Section 6.1, and the pseudocosts arc branching rule presented in Section 6.2. The exact pricer is ignored if $K = 3$ since the heuristic pricer is proven to be complete. The number of fractional arcs on which strong branching occurs is set as a parameter to 100.

BCP is compared against the state-of-the-art PICEF and PIEF. The implementation of PICEF and PIEF are retrieved from the original authors. It calls Gurobi 7.5.2 as a black-box mixed integer programming solver. Note that SCIP is roughly eight to nine times slower than Gurobi on benchmark problems according to the evaluations by Mittelmann (2019).

The three models are evaluated on the set of instances generated by Dickerson et al. (2012) and made publicly available on the the PrefLib repository (Mattei and Walsh, 2013). These instances are generated to capture the essence of real kidney exchange problems. Eighty of these instances are applicable to the CCMCP since the others involve chains, i.e., they are instances for the CCCCP. The 80 instances range from 16 to 2048 vertices and vary between 10% to 32% arc density. Each instance is solved for $K = 3$ and $K = 4$, totaling 160 runs.

All models are run on a single thread on an Intel Xeon E5-2660 V3 CPU at 2.6 GHz. Each run is given a time limit of 30 minutes and a memory limit of 4 GB.

7.2 Results and Analysis

Figure 6 plots the percentage of the instances optimized after a particular duration. On $K = 3$, BCP solves all 80 instances in under one minute. PICEF solves 65 instances, with the last of these 65 instances solved after 21 minutes. PIEF solves 64 instances, with the last of these solved after 13 minutes. On $K = 4$, BCP again outperforms PICEF and PIEF by solving 69 instances compared to 50 by both PICEF and PIEF. Overall, BCP solves more instances, despite the nearly order-of-magnitude difference in performance between SCIP and Gurobi.

The instances with 256 or fewer vertices appear to be trivial: all three models solve them in a few seconds at most. Table 1 compares several statistics over the instances with 512 or more vertices for $K = 3$.

Generating the large matrix in PIEF and PICEF caused major issues. PIEF and PICEF failed to build the model within one hour and was subsequently terminated respectively on 16 and 12 of the instances. This issue did not arise in BCP, which begins with zero columns.

On the instances that PIEF and PICEF solve, they are usually solved during presolve or in the root node. However, this is still very time-consuming for the larger instances. In particular, PICEF scales very poorly, despite not including the time taken to generate all cycles in the statistics reported in the table. BCP solves all instances except instance 276 in several seconds, and almost always at the root

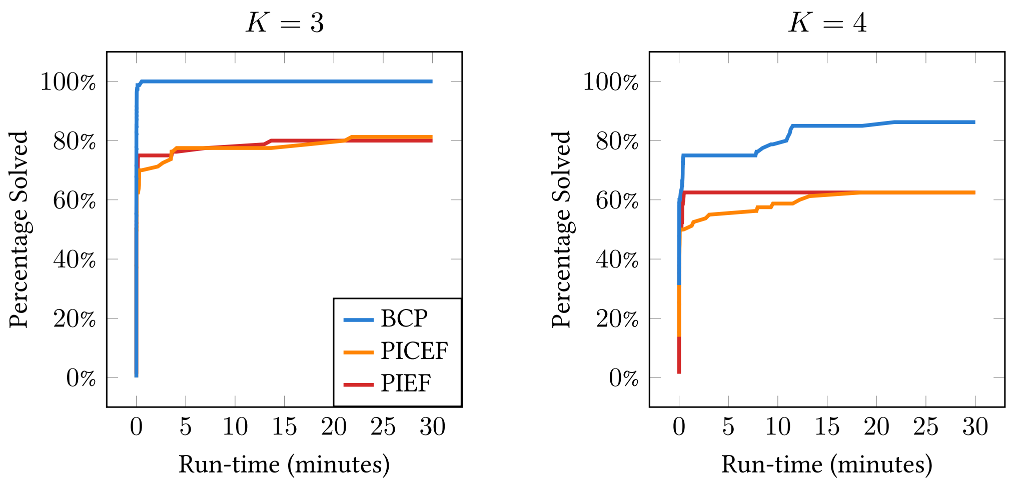

Figure 6: Percentage of instances solved after a given duration. Higher is better.

node, which indicates that the Dantzig-Wolfe reformulation attains a strong linear relaxation.

			PIEF					PICEF					BCP				
Instance	$\lvert V\rvert$	Density	Status	Time	Nodes	Obj	Bound	Status	Time	Nodes	Obj	Bound	Status	Time	Nodes	Obj	Bound
191	512	27.03%	Optimal	11	1	351	351	Optimal	17	1	351	351	Optimal	7	6	351	351
192	512	27.50%	Optimal	7	1	337	337	Optimal	16	1	337	337	Optimal	0	1	337	337
193	512	24.31%	Optimal	7	1	300	300	Optimal	161	1	300	300	Optimal	0	1	300	300
194	512	25.93%	Optimal	9	1	313	313	Optimal	214	1	313	313	Optimal	0	1	313	313
195	512	25.78%	Optimal	8	0	322	322	Optimal	212	1	322	322	Optimal	0	1	322	322
196	512	24.65%	Optimal	8	1	313	313	Optimal	130	0	313	313	Optimal	0	1	313	313
197	512	25.60%	Optimal	9	1	334	334	Optimal	17	1	334	334	Optimal	0	1	334	334
198	512	25.12%	Optimal	6	0	332	332	Optimal	14	1	332	332	Optimal	0	1	332	332
199	512	24.67%	Optimal	9	1	313	313	Optimal	16	1	313	313	Optimal	0	1	313	313
200	512	24.09%	Optimal	7	0	312	312	Optimal	18	1	312	312	Optimal	0	1	312	312
231	1024	25.82%	Failed	–	–	–	–	Optimal	210	1	658	658	Optimal	0	1	658	658
232	1024	25.53%	Failed	–	–	–	–	Optimal	245	1	662	662	Optimal	0	1	662	662
233	1024	25.07%	Failed	–	–	–	–	Timed Out	1800	0	567	12974210	Optimal	0	1	635	635
234	1024	24.79%	Optimal	777	1	641	641	Optimal	1267	1	641	641	Optimal	0	1	641	641
235	1024	25.51%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	0	1	660	660
236	1024	25.39%	Optimal	819	1	655	655	Failed	–	–	–	–	Optimal	0	1	655	655
237	1024	24.09%	Optimal	422	1	597	597	Optimal	1046	1	597	597	Optimal	0	1	597	597
238	1024	25.39%	Failed	–	–	–	–	Timed Out	1800	0	612	12556027	Optimal	0	1	671	671
239	1024	25.80%	Failed	–	–	–	–	Timed Out	1800	0	606	13078507	Optimal	0	1	673	673
240	1024	24.63%	Optimal	219	1	639	639	Optimal	1309	1	639	639	Optimal	0	1	639	639
271	2048	24.59%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	2	1	1284	1284
272	2048	24.39%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	2	1	1249	1249
273	2048	23.84%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	2	1	1232	1232
274	2048	23.91%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	1	1	1242	1242
275	2048	25.02%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	2	1	1301	1301
276	2048	25.13%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	33	8	1311	1311
277	2048	25.91%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	1	1	1316	1316
278	2048	24.24%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	1	1	1268	1268
279	2048	24.81%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	2	1	1271	1271
280	2048	25.33%	Failed	–	–	–	–	Failed	–	–	–	–	Optimal	1	1	1295	1295

Table 1: Statistics comparing PIEF, PICEF and BCP on the instances with 512 or more vertices and $K = 3$. For each instance, the table reports the instance ID, the number of vertices, the arc density, and for each of the models, the status, the total run-time in seconds, the number of branch-and-bound nodes, the primal (lower) bound and the dual (upper) bound.

For $K = 4$, PIEF and PICEF failed to construct the matrix on all the instances with 512 or more vertices. Table 2 shows the statistics for BCP, which successfully solves all but one of the instances with up to 1024 vertices, and times out on all instances with 2048 vertices. Even though BCP outperforms PIEF and PICEF in general, it is clear that BCP cannot solve huge instances.

A very significant finding is that the complete pricer did not find a column with positive reduced cost at any stage of the runs. This indicates that the heuristic pricer explores a sufficiently large portion of the search space, and that the exact pricer only needs to verify that the heuristic pricer found all fractionally-optimal columns. The reason for such an excellent result is not yet understood.

It was hoped that the various families of cuts would substantially tighten the linear relaxation but this is not the case: the tournament, TNT and sailboat cuts are rarely separated. Of the 160 runs, cuts were separated in only 23. For these 23 runs, the number of rows generated in each family, and also the number of columns, is reported in Table 3. Of the three families of inequalities, the sailboat inequalities are separated most often. This finding corroborates with the results by Mak-Hau (2018), which concluded that the tournament and TNT cuts are ineffective when the sailboat cuts are present in the two-index model. However, a new finding is that even the sailboat cuts appear to provide minimal benefits in the Dantzig-Wolfe reformulation, presumably due to the tight relaxation.

8 Future Directions

As the seminal study into branch-and-cut-and-price for the CCMCP, many directions for future work are available.

The outstanding performance of the heuristic pricer indicates that better pricing algorithms warrant further investigation. This research direction is particular important since the bottleneck in column generation usually lies in the pricing algorithm.

The pricers presented in this paper and in every other paper lack dominance rules. Without dominance rules, every cycle must be excluded either via bounds or via enumeration (e.g., Irnich and Desaulniers, 2005). It is well-established in the literature for VRPs that generating a large number of columns in each round of pricing together with strong dominance rules are essential for high-performance. One CCMCP pricer from the literature uses mixed integer programming, which provides bounds but cannot generate a large number of columns. Other researchers, including us, use a tree search that enumerates every possible cycle without considering bounds. Dominance rules will prevent every cycle from being examined. Preliminary experiments with some dominance rules showed that it was slower

			BCP				
Instance	$\|V\|$	Density	Status	Time	Nodes	Obj	Bound
191	512	27.03%	Optimal	24	1	352	352
192	512	27.50%	Optimal	26	1	337	337
193	512	24.31%	Optimal	18	1	300	300
194	512	25.93%	Optimal	22	1	313	313
195	512	25.78%	Optimal	23	1	322	322
196	512	24.65%	Optimal	15	1	313	313
197	512	25.60%	Optimal	22	1	334	334
198	512	25.12%	Optimal	19	1	332	332
199	512	24.67%	Optimal	21	1	313	313
200	512	24.09%	Optimal	23	1	312	312
231	1024	25.82%	Optimal	1308	1	659	659
232	1024	25.53%	Optimal	661	1	662	662
233	1024	25.07%	Timed Out	1800	2	633	635
234	1024	24.79%	Optimal	508	1	641	641
235	1024	25.51%	Optimal	678	1	660	660
236	1024	25.39%	Optimal	652	1	655	655
237	1024	24.09%	Optimal	470	1	597	597
238	1024	25.39%	Optimal	673	1	672	672
239	1024	25.80%	Optimal	691	1	673	673
240	1024	24.63%	Optimal	558	1	639	639
271	2048	24.59%	Timed Out	1800	1	1196	–
272	2048	24.39%	Timed Out	1800	1	1212	–
273	2048	23.84%	Timed Out	1800	1	1154	–
274	2048	23.91%	Timed Out	1800	1	1202	–
275	2048	25.02%	Timed Out	1800	1	1205	–
276	2048	25.13%	Timed Out	1800	1	1220	–
277	2048	25.91%	Timed Out	1800	1	1255	–
278	2048	24.24%	Timed Out	1800	1	1235	–
279	2048	24.81%	Timed Out	1800	1	1235	–
280	2048	25.33%	Timed Out	1800	1	1289	–

Table 2: Statistics for BCP on the instances with 512 or more vertices and $K = 4$.

K	Instance	Columns	Tournament	TNT	Sailboat
3	118	345	0	0	2
3	151	616	0	0	1
3	153	703	0	0	2
3	158	746	0	1	2
3	191	1582	8	16	4
3	192	1498	0	0	1
3	195	1318	0	0	2
3	197	1438	0	0	1
3	231	3235	4	8	5
3	233	3085	0	0	1
3	234	3651	0	0	1
3	237	2866	0	0	4
3	238	3329	3	2	3
3	240	3584	0	0	2
3	271	6782	0	0	1
3	272	7833	0	0	1
3	273	6752	0	0	2
3	275	7212	0	0	1
3	276	7079	0	0	1
3	279	6230	0	0	3
4	35	56	0	0	1
4	75	128	0	0	2
4	231	3722	0	0	1

Table 3: Number of columns found by the heuristic pricer and the number of cuts generated.

than the depth-first search pricer due to very few opportunities for dominance since the paths are very short, unlike VRPs. For this reason, developing dominance rules that are effective in practice could be difficult.

Other families of cuts should also be studied. The tournament, TNT and sailboat inequalities are discovered in previous work by studying the polyhedron of an arc-based formulation. These cuts become robust cuts in the master problem after applying Dantzig-Wolfe reformulation. The experiments show that many instances are solved at the root node without any cuts being separated, indicating that the relaxation is very tight, and hence, suggests that robust cuts in general are unlikely to prove beneficial. The natural avenue is to devise non-robust cuts, i.e., inequalities over the set packing master problem. Preliminary experiments evaluating the impact of the subset-row cuts (Constraint (12)) using the same naive separator from the original article, which enumerates all subsets S such that $|S| = 3$, found that the inequalities drastically worsened performance. Since the subset-row inequalities parametrized by $|S| = 3$ reason about incompatibilities across the columns that use the three vertices in S, the cuts are not expected to be effective because the CCMCP naturally has small values of K, which is exactly the number of non-zero coefficients in each column (before additional rows). Nevertheless, other non-robust cuts should be developed as they can reason about the set packing structure.

Branching rules are another avenue for future research. Preliminary experiments on a Ryan-Foster branching rule (Foster and Ryan, 1976) showed worse performance in general due to the additional considerations required in the pricing problem. Efficient implementations of pricing algorithms that can tolerate such branching rules should be considered in future work.

9 Conclusion

This paper implemented the first branch-and-cut-and-price (BCP) model of the Cardinality-constrained Multi-cycle Problem (CCMCP), which is one of the two graph optimization problems used in modeling kidney exchange. The model includes a new heuristic pricing algorithm and new separators for the convexifications of existing inequalities over arc-based formulations. This paper also generalized a family of inequalities developed in prior work and presented minor polyhedral results demonstrating that a subset of one of the convexified families of cuts, namely the sailboat cuts, are facet-defining on the master problem.

BCP, which includes all the components described above, is shown to empirically outperform the state-of-the-art position-indexed chain-edge formulation (PICEF) and the position-indexed edge formulation (PIEF). BCP solves 149 of 160 benchmark instances, compared to 115 by PICEF and 114 by PIEF. BCP achieves these remarkable results despite the eight to nine times performance difference in comparing the academic solver SCIP against the commercial solver Gurobi.

An inconvenient conclusion from this study is that the various families of robust cuts are not as important as hoped. Very few of the constraints are separated in practice, indicating the tightness of the Dantzig-Wolfe reformulation. Future work should explore the impact of non-robust inequalities since recent advances in Vehicle Routing Problems are attributed to the development of new families of non-robust cuts, e.g., Costa et al. (2019).

Acknowledgements

We would like to thank the three anonymous reviewers, whose comments have substantially improved this paper.

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, ACM, New York, NY, USA. pp. 295–304.

Achterberg, T., Koch, T., Martin, A., 2005. Branching rules revisited. Operations Research Letters 33, 42 – 54.

Anderson, R., Ashlagi, I., Gamarnik, D., Roth, A.E., 2015. Finding long chains in kidney exchange using the traveling salesman problem. Proceedings of the National Academy of Sciences 112, 663–668.

Applegate, D.L., Bixby, R.E., Chvatál, V., Cook, W.J., 2006. The Traveling Salesman Problem: A Computational Study. Princeton University Press.

Ascheuer, N., Fischetti, M., Grötschel, M., 2000. A polyhedral study of the asymmetric traveling salesman problem with time windows. Networks 36, 69–79.

Benichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribiere, G., Vincent, O., 1971. Experiments in mixed-integer linear programming. Mathematical Programming 1, 76–94.

Biró, P., Manlove, D.F., Rizzi, R., 2009. Maximum weight cycle packing in directed graphs, with application to kidney exchange programs. Discrete Mathematics, Algorithms and Applications 01, 499–517.

Codato, G., Fischetti, M., 2004. Combinatorial Benders' cuts, in: Bienstock, D., Nemhauser, G. (Eds.), Integer Programming and Combinatorial Optimization, Springer Berlin Heidelberg. pp. 178–195.

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.

Costa, L., Contardo, C., Desaulniers, G., 2019. Exact branch-price-and-cut algorithms for vehicle routing. Transportation Science doi:10.1287/trsc.2018.0878.

Desaulniers, G., Desrosiers, J., Solomon, M.M., 2005. Column Generation. Springer US.

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, ACM. pp. 25–42.

Dickerson, J.P., Procaccia, A.D., Sandholm, T., 2012. Optimizing kidney exchange with transplant chains: Theory and reality, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, International Foundation for Autonomous Agents and Multiagent Systems. pp. 711–718.

Foster, B.A., Ryan, D.M., 1976. An integer programming approach to the vehicle scheduling problem. Operational Research Quarterly 27, 367–384.

Fukasawa, R., Longo, H., Lysgaard, J., De Aragão, M.P., Reis, M., Uchoa, E., Werneck, R.F., 2006. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming 106, 491 – 511.

Glorie, K.M., van de Klundert, J.J., Wagelmans, A.P.M., 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.

Irnich, S., Desaulniers, G., 2005. Shortest Path Problems with Resource Constraints. Springer. chapter 2. Column generation, pp. 33–65.

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.

Johnson, D., 1975. Finding all the elementary circuits of a directed graph. SIAM Journal on Computing 4, 77–84.

Klimentova, X., Alvelos, F., Viana, A., 2014. A new branch-and-price approach for the kidney exchange problem, in: Murgante, B., Misra, S., Rocha, A.M.A.C., Torre, C., Rocha, J.G., Falcão, M.I., Taniar, D., Apduhan, B.O., Gervasi, O. (Eds.), Computational Science and Its Applications – ICCSA 2014, Springer International Publishing. pp. 237–252.

Lam, E., Le Bodic, P., Harabor, D., Stuckey, P.J., 2019. Branch-and-cut-and-price for multi-agent pathfinding, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), International Joint Conferences on Artificial Intelligence Organization. pp. 1289–1296.

Letchford, A.N., Salazar-González, J.J., 2006. Projection results for vehicle routing. Mathematical Programming 105, 251–274.

Lübbecke, M.E., Desrosiers, J., 2005. Selected topics in column generation. Operations Research 53, 1007–1023.

Mak-Hau, V., 2017. 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.

Mak-Hau, V., 2018. A polyhedral study of the cardinality constrained multi-cycle and multi-chain problem on directed graphs. Computers & Operations Research 99, 13 – 26.

Mattei, N., Walsh, T., 2013. PrefLib: A library for preferences http://www.preflib.org, in: Perny, P., Pirlot, M., Tsoukiàs, A. (Eds.), Algorithmic Decision Theory, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 259–270.

Mitchell, J.E., 2010. Branch and Cut, in: Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc.

Mittelmann, H.D., 2019. Benchmarking optimization softwarea (hi)story, in: EURO 2019. URL: http://plato.asu.edu/talks/euro2019.pdf.

Nemhauser, G.L., Wolsey, L.A., 1999. Integer and Combinatorial Optimization. Wiley-Interscience.

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., 2014. Improved branch-cut-and-price for capacitated vehicle routing, in: Lee, J., Vygen, J. (Eds.), Integer Programming and Combinatorial Optimization: 17th International Conference, IPCO 2014, Bonn, Germany, June 23-25, 2014. Proceedings, Springer International Publishing. pp. 393–403.

Plaut, B., Dickerson, J.P., Sandholm, T., 2016. Hardness of the pricing problem for chains in barter exchanges. CoRR abs/1606.00117. URL: http://arxiv.org/abs/1606.00117, arXiv:1606.00117.

Røpke, S., 2012. The solomon instances are solved! URL: https://www.gerad.ca/colloques/ColumnGeneration2012/presentations/session7/Ropke.pdf. presentation at the International Workshop on Column Generation 2012.

Røpke, S., Cordeau, J.F., 2009. Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science 43, 267–286.

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.

Spoorendonk, S., Desaulniers, G., 2010. Clique inequalities applied to the vehicle routing problem with time windows. INFOR: Information Systems and Operational Research 48, 53–67.

Vigo, D., Toth, P., 2014. Vehicle Routing: Problems, Methods, and Applications. Second ed., Society for Industrial and Applied Mathematics.