Column Generation for the Kidney Exchange Problem

Lucie Pansart, Hadrien Cambazard, Nicolas Catusse, Gautier Stauffer

12^{th} *International Conference on MOdeling, Optimization and SIMlation - MOSIM18 - June 27-29 2018*
Toulouse - France "The rise of connected systems in industry and services"

Column Generation for the Kidney Exchange Problem

Lucie Pansart, Hadrien Cambazard, Nicolas Catusse

Univ. Grenoble Alpes, CNRS, Grenoble INP*, G-SCOP
38000 Grenoble, France
{firstname.name}@grenoble-inp.fr

Gautier Stauffer

The Centre of Excellence in Supply Chain (CESIT)
KEDGE Business School
33000 Bordeaux France
gautier.stauffer@kedgebs.com

ABSTRACT: *The Kidney Exchange Problem (KEP) aims at finding the best exchanges in a barter market where agents are patients with a willing but incompatible donor. (Abraham et al. 2007) introduced a natural (exponential) integer programming formulation called the cycle formulation that they could solve efficiently by a branch-and-price approach. Recently, several countries allowed for the participation of altruistic donors in the exchanges. The corresponding variant of KEP is harder to solve as the pricing problem becomes $\mathcal{NP}$-complete. In this work, we study and experiment a column generation approach that takes into account altruistic donors. We use advanced techniques to circumvent the $\mathcal{NP}$-hardness of the pricing problem and show that the corresponding method can provide excellent guaranteed feasible solutions in a small amount of time.*

KEYWORDS: *Kidney exchange problem, column generation, integer programming, color coding*

1 INTRODUCTION

One in every thousand Europeans is facing end-stage renal disease for which the most effective treatment is kidney transplant. For such a patient, waiting for a deceased donor can be tedious while living donation is a viable alternative. Of course, in order to avoid certain excesses, living donation conditions are very strict and are defined by specific laws, for example, only a close relative is authorized to donate his or her kidney to a specific patient. Living donation is faced with a major hurdle in the potential incompatibilities that can occur between the patient and his or her donor: it is risky to perform an incompatible transplant as it would significantly reduce the patient's immune defenses. The use of paired donation programs can lead to higher graft survival rates (NHS 2017). A paired donation program makes exchanges between incompatible patient-donor pairs (a patient and a close relative willing to donate one kidney but medically incompatible with the patient). If a pair is included in an exchange, the donor gives one kidney to another – compatible – patient and the patient receives a kidney from another – compatible – donor of the exchange. Paired donation programs can also involve altruistic donors defined as singletons who anonymously and charitably give a kidney. Programs performing paired donations grow continuously, sometimes aiming at encompassing several countries, as their effectiveness is strongly dependent on the number of participants. Emerging international and European programs require to tackle pools of the order of thousands of patients. The problem of finding the "best" exchanges in a program is a challenging mathematical problem, in particular the question of developing efficient algorithms capable of tackling a large number of patients and donors at a time.

We focus in this study on algorithms providing good upper and lower bounds in order to be able to assess the quality of our results. (Abraham et al. 2007) developed an exponential integer programming formulation, the cycle formulation, for the problem containing only patient-donor pairs. They solve the model with a branch-and-price algorithm that performs well. This model yet fails to accommodate with the addition of altruistic donors. Informally (see section 2.2 for details), the linear relaxation of the cycle formulation is solved with a technique referred to as column generation, which iteratively add well-chosen variables. Identifying a variable to add is known as the pricing problem and is $\mathcal{NP}$-complete in the presence of altruistic donors (Plaut et al. 2016b) in contrast with (Abraham et al. 2007). In this paper, we propose to combine local search and a randomized method, called color coding, to solve the pricing problem efficiently. We also use a relaxation of the pricing problem to compute an upper bound even if the pricing step is not solved optimally.

The rest of this section introduces the problem, its modeling, and a short literature review. In section 2, we present the integer programming formulation used

*Institute of Engineering Univ. Grenoble Alpes

in this article as well as the column generation algorithm. The next two sections describe the algorithms used to solve the pricing problem (section 3) and the Kidney Exchange Problem (section 4). Finally experimental results are reported in section 5.

1.1 Definition of the problem

Paired donation. A basic Kidney Paired Donation Program (KPD) involves incompatible patient-donor pairs. It creates exchanges between some of the pairs such that the patient of a pair receives the kidney from the donor of another pair. Figure 1 shows the simplest case involving only two pairs.

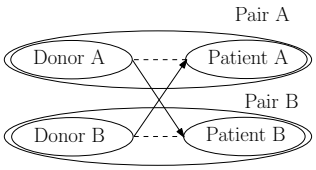

Figure 1 – An exchange of size 2. Dashed lines show incompatibility and arrows show the exchange.

A KPD can create several exchanges among the pool of pairs, called *cycles of donation*, of any size. However, the withdrawal of a donor after the transplant of his or her patient means that in another pair a donor already gave a kidney while his or her patient still waits for a transplant. To avoid this unacceptable situation, all operations of a cycle must be executed simultaneously. As a cycle requires twice as many operating rooms and surgical teams than pairs – one for each patient and each donor –, in practice cycles must be of a limited size $k \in \mathbb{N}$, typically $k = 2, 3, 4$.

Domino paired donation. When KPD involves *altruistic donors*, these latter create *chains of donation*. In this case, called *Domino Paired Donation*, the simultaneity is still desirable as the withdrawal of a donor breaks an exchange, but it is no more required since the critical situation described in the previous paragraph will never happen. Thus, some donors can be *bridge donors*: their patient will receive a kidney before they give their own. The last donor of a chain can either give a kidney to the *waiting list* or be considered later as an altruistic donor. As a result, there is no consensus on the need of a limit on the size of a chain, but when it exists, this limit is usually greater than k. In this paper we consider chains with at most $l \in \mathbb{N}$ transplants. Figure 2 illustrates the main types of chains.

Model. The Kidney Exchange Problem (KEP) is the problem of finding the *best* exchanges in a KPD, where *best* is defined by hospitals or transplant agencies.

Let us define P to be the set of patient-donor pairs and N the set of altruistic donors. The compatibility graph of a KEP is the directed graph $G = (V = P \cup N, A)$, where $A = \{(uv)$ if donor of u is compatible with patient of v, $\forall u \in V, v \in P\}$. We assume we are given a weight function $w : A \to \mathbb{R}^+$. w_{uv} represents the medical benefit of transplanting the kidney of donor u to patient v. A simple chain (resp. cycle) is a chain (resp. a cycle) with no repeated vertices (except the starting and ending vertices). We use the term **valid chain** to refer to a simple chain in G initiated by an altruistic donor with at most l arcs and **valid cycle** to refer to a simple cycle in G of length at most k. Chains and cycles must be simple since a donor (resp. a patient) can give (resp. receive) only one kidney. The term **valid path** refers to either a valid cycle or chain. The weight w_p of any path p is defined as the sum of its arcs weights. Figure 3 illustrates an example of the model. Formally, the KEP is: given G, w, k and l, find a set of disjoint valid paths of maximum weight in G. Note that this solution does not necessarily cover the maximum number of vertices since it depends on the weight function.

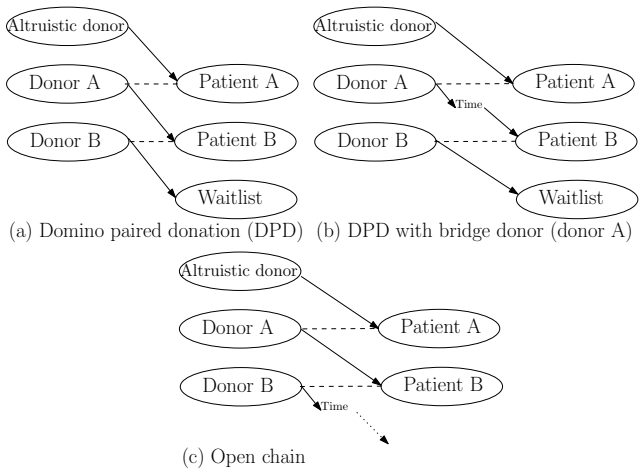

Figure 2 – The main types of chains. Dashed lines show incompatibility and arrows show the exchange. The dotted line shows a potential future exchange, not known at this stage of the program.

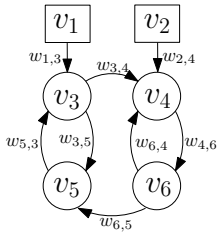

Figure 3 – An example of compatibility graph G. Squares represent altruistic donors and circles represent incompatible patient-donor pairs. Many solutions are possible, depending on l and k, e.g.:
{chain $(v_1v_3v_5)$ and chain $(v_2v_4v_6)$} or {chain $(v_1v_3v_5)$ and cycle (v_4v_6)} or cycle $(v_3v_4v_6v_5)$} ...

1.2 Literature review

Since we focus on integer programming approaches in this paper, the following section will mainly review literature on this topic. Other approaches and variants were studied these past years. The reader can find comprehensive literature reviews in (Mak-Hau 2017) and (Gentry et al. 2011).

The idea of kidney exchange programs was introduced by (Rapaport 1986) and developed for the first time in 1991 in South Korea (Kwak et al. 1999). Later on, the Netherlands, the UK and Spain started their program, followed by many others, including France and the USA. This problem is currently an important research topic in Europe (see (COST 2016)).

The KEP was proved $\mathcal{NP}$-complete by (Abraham et al. 2007) when $k > 2$ even in the case of KPD with pairs only (without altruistic donors). This basic case was widely studied and several integer programs were proposed. Standard formulations are exponential so branch-and-price approaches were developed by (Abraham et al. 2007) and (Klimentova et al. 2014), while (Constantino et al. 2013), (Mak-Hau 2017) and (Dickerson et al. 2016) introduced alternative compact formulations.

When chains are allowed, the problem is more complex. Indeed, the linear relaxation is harder to compute because the pricing problem is $\mathcal{NP}$-complete (Plaut et al. 2016b). To the best of our knowledge, no correct branch-and-price algorithm including altruistic donors was published (previous propositions of (Glorie et al. 2014) and (Plaut et al. 2016a) turned out to be incorrect, as the authors explained in (Plaut et al. 2016b)). Finally, (Anderson et al. 2015) described algorithms using integer programming formulations based on the traveling salesman problem. Compact formulations either do not scale up to large instances or provide poor upper bounds (Mak-Hau 2017). As we hope to construct international programs, including several hundreds or thousands of patients, we focus on the natural exponential formulation.

2 INTEGER PROGRAMMING

We introduce in this section an integer program directly adapted from the cycle formulation of (Abraham et al. 2007) and the framework of the column generation.

2.1 Path formulation

Let us define $\mathcal{P}$ as the set of all valid paths in G. The decision variables of the *Path Formulation* (*PF*) are:

$$\forall p \in \mathcal{P},\ x_p = \begin{cases} 1 \text{ if path } p \text{ is chosen in the exchange} \\ 0 \text{ otherwise} \end{cases}$$

$$z^* = max \sum_{p \in \mathcal{P}} w_p x_p \tag{1}$$

$$s.t. \sum_{p \in \mathcal{P}: v \in p} x_p \leq 1 \qquad \forall v \in V \tag{2}$$

$$x_p \in \{0,1\} \qquad \forall p \in \mathcal{P} \tag{3}$$

Constraints (2) impose that each vertex is taken at most once, namely donors and patients are involved in zero or one exchange. The objective (1) is to maximize the overall weight of exchanges. The linear relaxation of (*PF*) is noted (PF_{LP}) and composed of equations (1), (2) and :

$$x_p \geq 0\ \forall p \in \mathcal{P} \tag{4}$$

2.2 Column Generation

In (*PF*) the number of variables grows exponentially with k and l. To tackle this problem, a prevailing solution is to use a branch-and-price algorithm which mixes brand-and-bound with column generation. Column generation is used to solve the linear relaxation of (*PF*) as explained below.

2.2.1 Principle

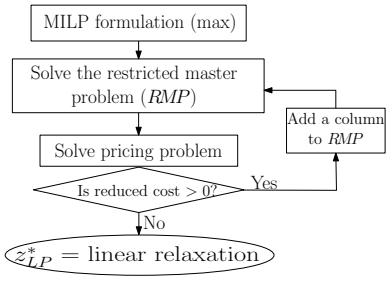

Figure 4 – Framework of the column generation

The general framework of the column generation is shown in figure 4. It is a form of the simplex algorithm; indeed, the simplex algorithm does not need all variables to be explicitly included in the model: the required variables can be iteratively added, until it is proved that no more variable is required. Thus, the linear relaxation of (*PF*) can be solved starting from a subset of valid paths – this problem is called the *restricted master problem* –, and this subset grows as we generate new variables (*i.e.*, new columns). Generating a new variable aims at improving the objective value of the current basis in the simplex algorithm. Finding such a column is called the *pricing problem* or *slave problem*. In our case, the pricing problem is the following: find a valid path of positive reduced cost or prove none exists. Thus, the pricing problem is to find a valid path of maximal reduced cost. The reduced cost of a path p is given by $rc_p = w_p - \sum_{v \in p} \alpha_v$

where α_v is the dual value of vertex v associated to constraints (2) in the linear relaxation of (*PF*).

2.2.2 Complexity of the pricing problem

Suppose we are given G, w, l and $a_v \forall v \in V$.

Cycles. The *cycle pricing problem* is the problem of finding a valid cycle of maximal reduced cost in G. It can be solved in polynomial time in $|V|$ with a modified Bellman-Ford algorithm (Glorie et al. 2012).

Chains. The *chain pricing problem* is the problem of finding a valid chain of maximal reduced cost in G. It is $\mathcal{NP}$-complete (Plaut et al. 2016b).

2.2.3 Our algorithm

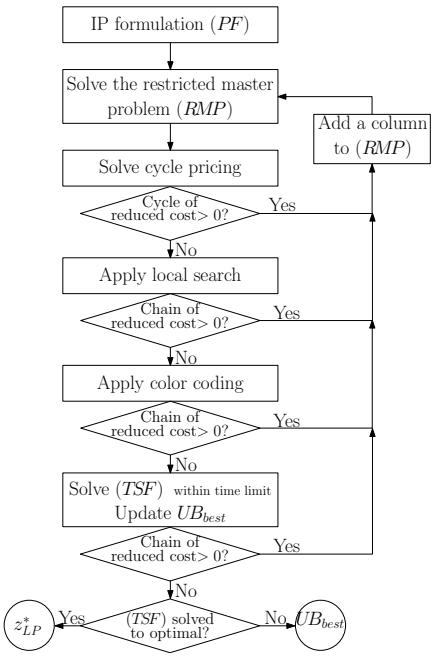

Figure 5 – The column generation algorithm

As the ChPP is $\mathcal{NP}$-complete, some efforts have to be made to solve this problem. In the next section, we will present different ways to solve it: an exact method – an integer programming formulation called Time-Staged Formulation (*TSF*) – and two non-exact methods – a local search heuristic and a randomized method called color coding (Alon et al. 1995).

We solve the pricing problem with algorithms of increasing complexity, as shown in figure 5. We try to solve it with the fastest methods first, which can be sufficient to find lots of columns. When no column is found with these algorithms, the exact method is used, solving (*TSF*) within a small time limit. As we will see in section 4.1, this provides a new upper bound on z^* and possibly a new column or the proof of the optimality of z^*_{LP}. The best upper bound UB_{best} is kept all along the course of the algorithm and is updated when (*TSF*) is solved.

3 SOLVING THE PRICING PROBLEM

In this section we study how to solve the ChPP which is the difficult part of the pricing problem. We construct a graph $G' = (V' = V \cup \{s,t\}, A')$ where s is a source and t a sink. We construct A' with A, A_s the set of arcs from the source to all altruistic donors and A_t the set of arcs from all pairs to the sink, namely $A_s = \{(sv) : \forall v \in N\}$, $A_t = \{(vt) : \forall v \in P\}$ and $A' = A \cup A_s \cup A_t$. Given the dual values α_v $\forall v \in V$, we define the arc function:

$$rc : A' \to \mathbb{R}$$

$$a \to rc_a = \begin{cases} -\alpha_v & \text{if } a = (sv) \in A_s \\ 0 & \text{if } a = (vt) \in A_t \\ w_{uv} - \alpha_v & \text{if } a = (uv) \in A \end{cases}$$

The reduced cost of a path, defined previously as $rc_p = w_p - \sum_{v \in p} \alpha_v$ is also $rc_p = \sum_{(uv) \in p} rc_{uv}$. We look for a valid chain p^* of maximal reduced cost: $rc_{p^*} = rc^* = \max_{p \in \mathcal{P}} rc_p$. It is a simple longest path problem for which a dynamic programming approach adapted from (Held & Karp 1962) can be developed, but does not scale up to the instances we aim at solving. In fact, it is enough for the column generation algorithm to find a valid chain of positive (not maximal) reduced cost, so the use of heuristic methods is reasonable. However, to prove that the column generation finished, an exact method is necessary. Thus, we present in this section two non-exact methods and an integer programming to handle the ChPP.

3.1 Color coding (CD)

(Alon et al. 1995) proposed a randomized algorithm to find simple paths of a given length, called *color coding*. We use this method to find a solution of ChPP. It follows two steps:

1. randomly color each vertex $v \in V$ with a color $c_v \in \{1, 2, ..., C\}$, where $C \in \mathbb{N}$ is the fixed number of colors. Note that s and t are not colored.
2. solve the problem of finding a *colorful chain* – *i.e.*, a valid chain using each color at most once – of maximal reduced cost.

Step 2 is solved with a dynamic program, defined by:

$$g^*(\mathcal{C}, v) = \max_{\substack{u \in V' : c_u \in \mathcal{C} \setminus \{c_v\} \\ (uv) \in A'}} \{g^*(\mathcal{C} \setminus \{c_v\}, u) + rc_{uv}\}$$

$g^*(\mathcal{C}, v)$ is the maximal reduced cost of a chain starting at s, visiting exactly one vertex of each color of $\mathcal{C}$ and finishing in v. The maximal reduced cost of a colorful chain is given by $\max_{\substack{\mathcal{C} \subseteq \{1,...,C\}, v \in P \\ |\mathcal{C}| \leq l+1}} g^*(\mathcal{C}, v)$

This dynamic programming has a space complexity of $\mathcal{O}(|V|2^C)$ and a time complexity of $\mathcal{O}(|V|^2 2^C)$.

In order to get an optimal chain p^* by color coding, step 1 must, by chance, color p^* with different colors. In this case, p^* would be colorful and be returned by step 2. The probability that this event occurs is $Pr\left[p^* \text{ is colorful }\right] = \frac{\#\text{ colorful chains}}{\#\text{ colored chains}} = \frac{C!/(C-l)!}{C^l} = \frac{C!}{(C-l)!C^l} = \rho$. To increase the chance of coloring an optimal chain with different colors, the algorithm must be repeated several times.

Let X be a random variable denoting the number of iterations needed for p^* to be colorful for the first time. X follows a geometric distribution, so the probability that this event occurs before the i^{th} iteration is $Pr\left[X \leq i\right] = 1 - (1-\rho)^i$. We want to repeat the algorithm enough times to guarantee a high probability of having a colorful optimal chain. Let $\epsilon \in [0,1]$, we seek i such that $Pr\left[X \leq i\right] = 1 - \epsilon \Leftrightarrow i = \frac{ln(\epsilon)}{ln(1-\rho)}$.

Choice of the parameters. The efficiency of the color coding algorithm is determined by C, the number of colors, and t, the number of iterations. As they grow, the probability of finding the optimal chain grows with them, as well as the computation time. For given l and ϵ (the maximal allowed probability to fail), their values are chosen to minimize the theoretical complexity. More precisely, let $\overline{C} \in \mathbb{N}$ be the maximal authorized number of colors (for computational reasons), we choose:

$$(C,t) = \underset{C \in [\![l,\ldots,\overline{C}]\!], t \in \mathbb{N}}{\arg\min} \{t \times 2^C s.t.\ t = \frac{ln(\epsilon)}{ln(1-\rho)}\}$$

The steps 1 and 2 are then repeated t times and the best colorful valid path is returned with a probability of $1-\epsilon$ to be optimal.

3.2 Local search (LS)

C and t grow as l does to guarantee the wanted probability of success making the color coding suffering from long computation times. In this case, the leading alternative is the use of heuristics. We developed a local search which starts with a random chain and try to improve it with three kinds of movements. They aim at increasing the reduced cost of the chain, keeping it valid, by inserting, removing or exchanging 1, 2 or 3 vertices. The local search stops when it reaches a local minimum, *i.e.,* when no movements improves the current path.

3.3 Integer programming (*TSF*)

The methods described above run fast but when they find no valid chain of positive reduced cost, no conclusion can be made. If such a chain exists, we want to find it; if it does not exist, we want to prove it. Thus, an exact method must be implemented to address this issue. We state the problem as a time-staged integer programming formulation (*TSF*). We define $I = \{0,1,...,l+1\}$, $I^* = \{1,...,l+1\}$ and the decision variables: $\forall (uv) \in A'$, $i \in I$

$$r_{uv}^i = \begin{cases} 1 \text{ if } (uv) \text{ is the } i^{th} \text{ arc in the path} \\ 0 \text{ otherwise} \end{cases}$$

$$rc^* = max \sum_{i \in I} \sum_{(uv) \in A'} rc_{uv} r_{uv}^i \tag{5}$$

$$s.t. \quad \sum_{v \in N} r_{sv}^0 = 1 \tag{6}$$

$$\sum_{i \in I \setminus \{0,1\}} \sum_{u \in P} r_{ut}^i = 1 \tag{7}$$

$$\forall v \in V, \forall i \in I : \sum_{\substack{u \in V': \\ uv \in A'}} r_{uv}^i = \sum_{\substack{u \in V': \\ uv \in A'}} r_{vu}^{i+1} \tag{8}$$

$$\forall i \in I : \sum_{(uv) \in A'} r_{uv}^i \leq 1 \tag{9}$$

$$\forall v \in V : \sum_{i \in I} \sum_{\substack{u \in V': \\ uv \in A'}} r_{uv}^i \leq 1 \tag{10}$$

$$\forall uv \in A',\ \forall i \in I : r_{uv}^i \in \{0,1\} \tag{11}$$

Constraints (6) and (7) ensure that the first chosen arc leaves the source and that the sink is reached. Constraints (8) are flow constraints guaranteeing that if a vertex is reached by the i^{th} arc then it is left with the $i+1^{th}$. Constraint (9) imposes that only one arc is taken at stage i and constraint (10) forbids a vertex to be taken more than once. Note that the solution given by (*TSF*) may contain $l+2$ arcs but two of them are fictive (one from the source and one to the sink).

4 SOLVING THE KEP

The following section describes how the column generation can be further exploited to provide a feasible guaranteed solution to (*PF*) without the use of a complete branch-and-price method.

4.1 Dual and lagrangian bounds

Define (PF_β) as the linear program composed of equations (1), (2), (4) and the redundant constraint:

$$\sum_{p \in \mathcal{P}} x_p \leq \frac{|V|}{2} \tag{12}$$

This constraint is valid for every KEP. Indeed it is stating that the number of paths is at most half of the number of pairs, which is implied by the fact that a path involves at least two vertices. Let β be the

dual value associated with constraints (12). The dual program of (PF_β) is :

$$(D) \quad min \sum_{v \in V} \alpha_v + \frac{|V|}{2}\beta \tag{13}$$

$$s.t. \sum_{v \in p} \alpha_v + \beta \geq w_p \qquad \forall p \in \mathcal{P} \tag{14}$$

$$\alpha_v \geq 0 \qquad \forall v \in V \tag{15}$$

$$\beta \geq 0 \tag{16}$$

Solving optimally the i^{th} pricing problem provides $rc^* = \max_{p \in \mathcal{P}} \left(w_p - \sum_{v \in p} \alpha_v^i \right) \geq 0$. Thus, setting β^i to rc^* provides a feasible solution (α^i, β) of (D), where α^i is the vector of dual values of the i^{th} restricted master problem. By weak duality, any feasible solution (α, β) produces an upper bound of z^*_{LP} so that $\sum_{v \in V} \alpha_v + \beta \frac{|V|}{2} \geq z^*_{LP}$. Furthermore, as the restricted master problem is solved optimally, we know that $\sum_{v \in V} \alpha_v^i = z^i_{LP}$. Thus, $z^i_{LP} + rc^* \frac{|V|}{2} = UB^i_D$ is a dual (upper) bound of z^*_{LP} at iteration i.

When rc^* cannot be computed (recall that the pricing problem is $\mathcal{NP}$-complete), we compute rc^*_{LP} the value of the linear relaxation of (*TSF*). It is an upper bound of rc^* so $z^i_{LP} + rc^*_{LP} \frac{|P|}{2} = UB^i$ is also an upper bound of z^*_{LP}.

Consequently, these bounds are upper bounds of the optimal value of (*PF*): $z^* \leq UB^i_D \leq UB^i$. Note that when $rc^* = 0$ the dual bound is equal to the primal value, which proves that (PF_{LP}) is solved optimally.

4.2 Solving of (PF)

Even if z^*_{LP} was not computed within the time limit, the algorithm always returns an upper bound UB_{best} of z^*, as well as a set $\mathcal{P}' \subseteq \mathcal{P}$ of valid paths. When the column generation stops, we solve the integer program (*PF*) restricted to $\mathcal{P}'$ thus providing a feasible solution $\underline{z}$ for the problem: this is a lower bound. The quality of this feasible can be assessed against the upper bound UB_{best}. As we will see in section 5, it seems unnecessary to implement the branch-and-price since the gap between the upper and the lower bounds is small for most instances, ensuring that the feasible solution is close to the optimal solution.

4.3 Pre-processing

The size of the graph can be reduced to improve the efficiency of all the algorithms. We use a simple filtering method that removes arcs that cannot belong to any valid path, because they are "too far" from other vertices. Let d_{uv} be the shortest distance (in number of arcs) between u and v, $\forall u, v \in V$.

$R_N = \{(uv) \in A : \forall r \in N \ d_{ru} \geq l\}$ is the set of arcs that cannot belong to a valid chain.
$R_V = \{(uv) \in A : d_{vu} \geq k\}$ is the set of arcs that cannot belong to a valid cycle. $R = R_N \cap R_V$ is the set of arcs than cannot belong to any valid path and thus $A = A \setminus R$.

5 NUMERICAL EXPERIMENTS

5.1 Benchmark

Instances were generated with an online data set generator[1]. We choose the parameters in order to vary δ the density of the compatibility graph, $|P|$ the number of patient-donor pairs and γ the percentage of altruistic donors in relation to $|P|$. These parameters take the following values: $\delta = 5\%, 10\%$, $\gamma = 1\%, 5\%, 10\%$ and $|P| = 100, 200, 400$, leading to 18 classes of instances, each of them containing 10 instances. The probability of success at finding the optimal chain by color coding was set to 90%.

5.2 Experimental setup

We used the CPLEX Java API (version 12.6) to solve the different linear programs. Experiments were performed on an Intel®Xeon®CPU E5-2440 v2 @ 1.90GHz processor and 32 GB of RAM, with a memory limit of 8 GB of RAM. The time limit was set to 900 seconds. The time limit to solve (*TSF*) was set to 90 seconds.

5.3 Results

In this section, we use the following notations (average and number out of 10).

- CPU: average time in seconds to compute z^*_{LP}
- #LR: number of instances where z^*_{LP} is computed
- #OPT: (resp. %OPT) number (resp. percentage) of instances where z^* is computed
- GAP: average gap between upper and lower bounds for (*PF*)

Table 1 presents the results for $k = 3$ and $l = 4$. It shows that the computation time increases as the number of pairs in the instance grows. As expected, this increase is stronger when the graph is denser. The results also demonstrate that the path formulation provides a linear relaxation of high quality. Small gaps between lower and upper bounds ($< 0.3\%$ in average for $|P| = 100, 200, 400$) are reported and the proof of optimality is done for more than 35% of the instances within 2 minutes of computation time (15 seconds in average). This supports the idea that implementing the complete branch-and-price algorithm

[1] Available at http://www.dcs.gla.ac.uk/jamest/kidney-webapp/#/generator

δ	γ	$\|P\|$	CPU	# LR	# OPT	GAP
5%	1%	100	0.1	**10**	10	**0%**
		200	1.2		4	**0.47%**
		400	25.7		2	**0.44%**
		800	784	9	0	4.46%
	5%	100	0.3	**10**	7	**0.30%**
		200	3.6		3	**0.20%**
		400	78		2	**0.28%**
		800	-	0	0	77.01%
	10%	100	0.3	**10**	8	**0.10%**
		200	4.9		7	**0.13%**
		400	115		5	**0.05%**
		800	-	0	0	80.34%
10%	1%	100	0.4	**10**	5	**0.57%**
		200	6.9		1	**0.41%**
		400	124		1	**0.14%**
		800	-	0	0	83.87%
	5%	100	1.3	**10**	4	**0.55%**
		200	17.3		0	**0.37%**
		400	300		0	**0.44%**
		800	-	0	0	85.45%
	10%	100	2.1	**10**	4	**0.38%**
		200	31		1	**0.17%**
		400	587		0	**0.2%**
		800	-	0	0	86.43%

Table 1 – $k = 3$ and $l = 4$

may not be necessary. We also tested the algorithm for instances where $|P| = 800$, but z^*_{LP} was calculated for only 9 instances (with $\delta = 5\%$ and $\gamma = 1\%$). For all the other instances the final gap was 81.8% in average, but this high gap is due to the inability to compute the linear relaxation, as the lower gap of 4.46% demonstrates. Figure 6 shows the evolution of the upper bound (UB_{best}) and the lower bound (value of the restricted master problem) as the columns are added for a specific instance with 800 pairs, a density of 5% and 5% of altruistic donors. The upper bound is not calculated when cycles only are added.

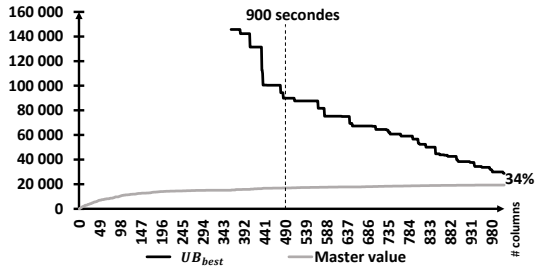

Figure 6 – Evolution of the value of (RMP) and UB_{best} with a time limit of 1 hour for $|P| = 800$, $\delta = 5\%$ and $\gamma = 5\%$

Computation times are higher for $l = 7$ and $l = 10$, but the algorithms behaves similarly. Due to the difficulty of the pricing problem when l is that big, the linear relaxation z^*_{LP} may not be computed within the time limit. Figure 7 outlines the difficulty to compute z^*_{LP} by showing the decreasing number of instances where it is computed and the increasing time to obtain it. We summarized in table 2 the average gap and the percentage of solved instances depending on the computation (or not) of the linear relaxation.

	$l = 7$		$l = 10$	
	GAP	% OPT	GAP	% OPT
z^*_{LP}	0.4%	34%	0.4%	44%
no z^*_{LP}	25.9%	-	29.2%	-

Table 2 – Average gap and percentage of solved instances depending on the computation of z^*_{LP}

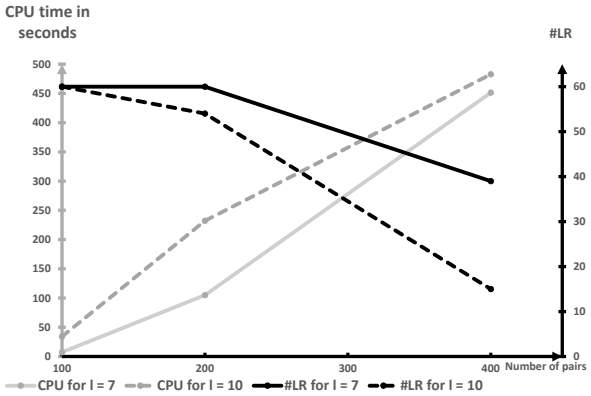

Figure 7 – Number of instances (out of 60) where z^*_{LP} is computed (in black) and average CPU time to compute it (in grey), for $l = 7$ (solid lines) and $l = 10$ (dashed lines).

l	Cycles	Chains	CyPP	LS	CD	(TSF)
4	161	129	0.08	0.05	0.31	1.06
7	263	174		0.25	4.69	3.21
10	354	149		0.64	-	5.65

Table 3 – Number of columns added and average time of pricing algorithms (in seconds)

In average, over all the instances, solving the pricing problems uses more than 90% of the total computation time. Table 3 details the average times, for one iteration, of the different pricing algorithms as l grows. This highlights that the bottleneck remains the computation of each chain pricing problem.

These results show that our algorithm is efficient to find good lower and upper bounds. When the column generation ends before the time limit, the gap is almost (and sometimes is) zero. However, when the time limit is reached before computing the linear relaxation, the gap may deteriorate, because the best dual bound kept all along the column generation algorithm is poor. The improvement of this dual bound – *e.g.*, by updating $\frac{|V|}{2}$ which is a poor upper

bound on the number of paths in a solution or thanks to a stronger integer programming formulation than (*TSF*) – could provide a better final gap even when the linear relaxation is not provided.

6 CONCLUSION

In this paper we studied a column generation algorithm for the kidney exchange problem. It uses the exponential formulation derived from the formulation of (Abraham et al. 2007), which was never strongly exploited up to now to handle altruistic donors.

The results show an encouraging track to explore, as the column generation algorithm provides integer solutions of good (guaranteed) quality in a reasonable amount of time and without the need of branching. However, the solved instances are smaller than the real or expected pools in the USA or in Europe, the target being several hundreds, nay thousands of patients. Further work needs to be done to stabilize the column generation algorithm and thus reduce the number of iterations, which is an important limitation noticed in the results, as well as the need to speed up the pricing step. This includes work on the dual bound described in the previous section as well as development of new algorithms for the chain pricing problem, *e.g.*, derandomization of the color coding, stronger integer formulations, more efficient heuristics or dynamic programming approach with filtering to overcome memory issues.

REFERENCES

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

Alon, N., Yuster, R. & Zwick, U. (1995). Color-coding, *Journal of the ACM (JACM)* **42**(4): 844–856.

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**(3): 663–668.

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**(1): 57–68.

COST (2016). http://www.cost.eu/COST_Actions/ca/CA15210.

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

Gentry, S. E., Montgomery, R. A. & Segev, D. L. (2011). Kidney paired donation: fundamentals, limitations, and expansions, *American Journal of Kidney Diseases* **57**(1): 144–151.

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**(4): 498–512.

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

Held, M. & Karp, R. M. (1962). A dynamic programming approach to sequencing problems, *Journal of the Society for Industrial and Applied Mathematics* **10**(1): 196–210.

Klimentova, X., Alvelos, F. & Viana, A. (2014). A new branch-and-price approach for the kidney exchange problem, *International Conference on Computational Science and Its Applications*, Springer, pp. 237–252.

Kwak, J., Kwon, O., Lee, K. S., Kang, C. M., Park, H. Y. & Kim, J. (1999). Exchange-donor program in renal transplantation: a single-center experience, *Transplantation proceedings*, Vol. 31, Elsevier, pp. 344–345.

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**(1): 35–59.

NHS (2017). Annual report on living donor kidney transplantation.

Plaut, B., Dickerson, J. P. & Sandholm, T. (2016a). Fast optimal clearing of capped-chain barter exchanges, *AAAI Conference on Artificial Intelligence (AAAI)*.

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. T. (1986). The case for a living emotionally related international kidney donor exchange registry., *Transplantation proceedings*, Vol. 18, pp. 5–9.