Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Econometrics & Operations Research

A Modular Matheuristic Algorithm for Finding Warm-Start Solutions to the Kidney Exchange Problem

Daniel Enzlin (533483de)

Supervisor:	Roby Cremers
Second assessor:	Albert Wagelmans
Date final version:	12th July 2024

Abstract

Kidney transplants are life-saving surgeries that require a person to donate their kidney to the patient. However, the donor's kidney is often not compatible with that of the patient. For this reason, Kidney Exchange Programs were created where incompatible patient-donor pairs are pooled together and matched with others to exchange kidneys between pairs. However, finding the maximum amount of matches possible within this pool of pairs is an NP-complete problem, and exactly solving the problem to optimality quickly becomes infeasible as the size of the pool of pairs grows. Increasing the size of these pools increases the number of possible transplantations, so developing new exact Mixed Integer Linear Programming formulations of the problem that can decrease solving time is an active area of research. Two of these formulations are reproduced and evaluated, after which a matheuristic algorithm is proposed that creates heuristic solutions to be used as warm-start solutions for any current and future formulation of the problem. A warm-start is a sub-optimal solution that is supplied to an MILP solver to reduce solving time. When constructing the matheuristic algorithm, a modular approach is taken that includes the use of a Greedy matheuristic and a dynamic randomized Local Search algorithm. Computational tests show that providing the warm-starts to an exact solver allows it to find optimal and near-optimal solutions for pools that are up to thrice as big as the largest pool currently solvable with the two reproduced exact formulations.

Contents

1 Introduction

For people suffering from kidney failure, a kidney transplantation is the only option for long-term survival. If a patient can find a person willing to donate a kidney, that kidney must also be compatible with the patients blood group and immune system. Often patients have a donor but they are not compatible. For these situations many countries have set up Kidney Exchange Program, where incompatible pairs of patients and donors are matched with other pairs to exchange their kidneys and make more transplantations possible (Constantino, Klimentova, Viana & Rais, 2013). This exchange can be performed in a way where two pairs trade kidneys, but by increasing the amount of pairs considered for an exchange more possibilities for exchanging are created. A cycle of pairs can be created where each pair donates a kidney to the next pair in the cycle. Finding these cycles in the pool of pairs participating in a kidney exchange program allows for more kidneys to be donated and more lives to be saved. (Abraham, Blum & Sandholm, 2007)

Kidney exchange programs keep track of a pool of pairs that needs to be matched and perform a matching at regular time intervals where they try to match pairs by finding as many of these cycles as possible.

Transplantations in such cyclic exchanges are performed simultaneously to prevent situations where a pair donates a kidney but the donor of the kidney they would receive becomes unable to donate, either willingly or unwillingly. Logistical constraints only allow for a limited number of surgeries to be performed simultaneously, so a limit is often placed on the length of exchange cycles.

The problem of matching as many pairs as possible by finding cycles with a limited length in the pool of pairs is called the k-cycle Kidney Exchange Problem (KEP), where k is the maximum cycle size allowed. For $k = 2$ the problem is solvable in polynomial time, but for $k \geq 3$ the problem is known to be NP-hard, meaning that as the number of pairs in the pool

increases, it quickly becomes difficult to solve exactly in a reasonable amount of time (Abraham et al., 2007). Increasing the pool size solvable to optimality is an ongoing area of research with clear practical applications. Combining disjoint pools of patient-donor pairs allows for matches from one pool to the other, and these new matches are likely to allow the combined pool to match more pairs than if the pools are separate. Multiple European countries currently run independent kidney exchange programs (Constantino et al., 2013) and increasing the solvable pool size is one of the steps that have to be made in order for these programmes to merge, which is something that is actively being discussed (Biró et al., 2019).

The concept of a kidney exchange program exchange program was first proposed by Wallis, Samy, Roth and Rees (2011) has been actively studied since. Abraham et al. (2007) formulated the k-cycle KEP in two ways using Mixed Integer Linear Programming (MILP) where the objective is to maximize the number of matches: the Cycle formulation, which is a set covering formulation where every possible cycle has a decision variable, and the Edge formulation, where the problem is solved using graph theory. Donor-patient pairs are the vertices in a directed graph and they are connected to another node if they can donate to that pair.

Real-world kidney exchange programs often have more complex objectives that include considerations such as prioritizing patients who need a kidney more urgently or minimizing the chance kidneys being rejected by patients' immune systems, but maximizing the number of transplantations is always one of the primary objectives (Mak-Hau, n.d.). As such, improving exact formulations of the k-cycle KEP where the only objective is to maximize the number of transplantations is an active area of research. At the moment of writing the formulation that solves instances of realistic data the fastest is the Edge formulation-based. For the basic k-cycle KEP it outperforms previous formulations in terms of solving times and the maximum size of the pool of pairs it can solve to optimality in reasonable time. Importantly, the linear relaxation of this formulation is also tight, meaning no other formulation's relaxation sets a lower upper bound on the optimal number of solvable matches (Constantino et al., 2013).

Instead of finding a better exact formulation, this paper provides a different way to increase the number of transplantations by providing a way to find warm-start solutions for any current or future formulation. A warm-start solution is a sub-optimal solution to an MILP problem that can be used to reduce the solving time. Depending on the way the problem is solved or which solving software is used, a warm-start solution will be used in different ways, and commercial solvers like GUROBI often do not give details on their proprietary methods. However, the most common way of solving MILP problems is by using branching methods (Clausen, 2003) for which a warm-start solution can significantly reduce solving times (Huang et al., 2021).

To find these warm-start solutions we propose a modular matheuristic algorithm. It is modular in the sense that it consists of multiple independent parts that could be improved or replaced, and it is a matheuristic algorithm in the sense that it uses mathematical programming techniques to find a heuristic solution. The matheuristic algorithm uses an exact formulation of the k-cycle KEP at one point to solve smaller sub-problems, but it is implemented in a way where that formulation can be replaced by any current or future exact formulation so that the algorithm can adapt to new developments in exact formulations of the problem. Every formulation has a limit on how large of a pool it can solve to optimality in reasonable time,

and the goal for this matheuristic algorithm is to increase that limit by shortening solving times through use of a high objective value warm-start solution.

Two formulations from a paper by Constantino et al. (2013) are reproduced to evaluate the effect of warm-starts on different exact formulations of the k-cycle KEP. These are the Cycle Formulation, and the newly proposed Extended Edge formulation, which is a compact version of the Edge formulation by Abraham et al. (2007).

This paper first gives an exact description of the problem and introduce notation. Afterwards the general structure of the matheuristic algorithm is laid out and each part is explained in detail. Lastly, numerical experiments are performed to evaluate its performance and the results are discussed.

2 Problem Description

To define the k-cycle KEP we use a graph theory-based formulation of the problem using by Constantino et al. (2013).

Let $G(V, A)$ be a directed graph where patient-donor pairs that participate in the exchange program are represented by vertices $V = 1, ..., |V|$. Possible matches are respresented by arcs between these vertices. The set of arcs A contains arc (i, j) if the donor of vertex i can donate to the patient of vertex j. The objective is to maximize the total number of transplants, so the Kidney Exchange Problem is defined as finding a set of vertex-disjoint cycles having length at most k in $G(V, A)$ that contains the maximum amount of vertices.

For the rest of this paper the patient-donor pairs will be referred to as vertices, and these vertices are considered matched if they are in a cycle with a length of at most k.

As mentioned by Delorme, García et al. (2023), the Dutch KEP limits cycles to include at most four donors, and the UKLKSS limits cycles to three donors. Increasing the value of k results in longer solving times (Mak-Hau, n.d.) but does not significantly increase the number of matches for $k > 4$ (Abraham et al., 2007). For these reasons and for the sake of conciseness we will only consider the KEP for $k = 4$ in this paper.

3 Methodology

This section first discussed the reproduction of the formulations by Constantino et al. (2013). Afterwards, an extension to that paper in the form of a heuristic algorithm is laid out. The algorithm is comprised of various sub-steps, so a general overview is given before the individual parts are explained in detail.

3.1 Reproduction

The goal of the matheuristic algorithm proposed in this paper is to create high quality solutions that can be used as warm-starts when solving exact formulations of the k-cycle KEP. To evaluate these solution's performance as warm starts we reproduced two exact formulations from a paper by Constantino et al. (2013). The paper contains four formulation, and we chose to reproduce the Cycle and Extended Edge (EE) formulations since these were the two best performing

formulations in the paper's initial small-scale testing and were thus tested with large data sets that better reflect reality. Another reason why these two formulations are reproduced is that they represent the two types of exact formulations of the KEP. Cycle-based formulations see the problem as a collection of all possible cycles from which the best combination must be chosen, and Edge-based formulations see the problem as a graph where vertices representing the pairs must be linked by activating arcs representing a match between two pairs. The full formulations will not be repeated in this paper for the sake of brevity. For this we refer the reader to the work by Constantino et al. (2013).

These two implementations are later also considered for a step in the matheuristic algorithm where an exact formulation is needed to solve a sub-problem. The computational results and findings on the reproduction can be seen in Sections 4.1 and 4.2.

3.2 Matheuristic Algorithm Overview

We first give an overview of the structure of the algorithm. Detailed explanations of the sub-steps and terminology are done in specific sections for each sub-step.

The algorithm consists of two major parts:

First, it finds an initial solution by combining two methods for creating feasible solution that are time efficient but do not produce high quality solutions by themselves. It begins by performing a pairwise matching, which finds the optimal objective value for $k = 2$. Then, an ILP model finds a 2-cycle solution with this objective value and optimal characteristics for the next step, in which this solution is improved using a randomized greedy heuristic.

The second part is a dynamic random local search matheuristic that uses the initial solution from the previous part as a starting point. It iteratively tries to find better solutions by creating a neighbourhood around the current solution and moving to the best neighbouring solution. The neighbours are created by randomly destroying cycles in the current solution and solving the k-cycle KEP for these freed vertices and previously unmatched vertices. The KEP for this sub-set of all pairs is solved exactly. If the search becomes stuck in a local optimum or a plateau the number of cycles that are destroyed is increased to widen the search. This search is halted if a theoretical upper bound is reached. There are multiple of these upper bounds and they can be calculated independently of the local search.

This improved solution found by the local search can now be used as a warm-start solution for an exact formulation of the k-cycle KEP.

3.3 Pre-processing

Depending on the origin of a data instance for the KEP, it is possible that not every patient-donor pair in the pool can be matched. For a pair to be matched in a cycle it must donate and receive a kidney. As such every pair in V that can either only donate or only receive will be removed from the set. It is possible that this removal causes another pair to lose their ability to either donate or receive if the removed pair was their only match. As such the process of removal must be repeatedly performed until every pair left can donate and receive.

3.4 Maximum Pairwise Matching

The first step of constructing the initial solution is finding something called the maximum pairwise matching. The KEP with a cycle length capped at k is known to be NP-Hard for values of $k \geq 3$. However, for $k = 2$ the problem reduces to a maximum matching problem (Abraham et al., 2007). The solution of this problem is thus called the maximum pairwise matching and it is feasible for any k-cycle KEP on the same problem instance since the set of all possible 2-cycles is a subset of all possible k-cycles when $k > 2$. The problem can be solved in polynomial time using Edwards' Blossom algorithm (Edmonds, 1965).

3.5 Greedy Randomized Heuristic

The solution found using maximum pairwise matching is improved using a Greedy Randomized algorithm.

Greedy algorithms are a class of widely used problem-solving matheuristics that iteratively build a solution by always taking the optimal local step. One way of applying this technique to the KEP is to build a solution by iteratively picking the "best" cycle available. The question here is what defines the "best" cycle at a given point. Several methods are devised to define this:

3.5.1 Cycle Scoring Methods

What is needed to effectively use a greedy algorithm is a way to know what makes a cycle more likely to be in an optimal solution. Multiple methods to determine what differentiates "good" cycles found in optimal solutions from the set of all possible cycles are proposed. These all make use of a scoring system.

Each donor and patient are assigned a score on $(0, 1]$ that indicates the fraction of others they can donate to/receive from. A person with a low score is thus harder to match and vice versa. A match between a donor and a patient with two low scores is desirable, since it takes two hard to match people out of consideration while minimally lowering the chance for others to find a match. This idea can be extended to cycles, where score of a cycle is a function of the scores of the matches it consists of.

Three ways of determining the score of a match between a donor and patient are considered:

- taking the sum of the donor and patient's scores,
- taking the product of their scores, and
- taking the maximum of both scores.

Two ways of combining the scores of matches to create scores of cycles are considered:

- taking the sum of the scores of inidvidual matches in the cycle, and
- taking the average of the scores of inidvidual matches in the cycle.

The six possible combinations of scoring matches and cycles each rank cycles in a different way. The methods are compared in Section 4.4.3, and best performing method chosen for the computational results of the next step of the algorithm.

3.5.2 Creating a Starting Point for the Greedy Algorithm

After the maximum objective value through pairwise matching is found and the method of assigning scores to cycles is known, a starting solution for the Greedy algorithm is found. This is done by exactly solving an MILP model that finds a pairwise matching with the same number of matches as the maximum found earlier, but with objective of minimize the scores of the cycles in that pairwise matching. This exact mixed integer linear programming model is based on the Cycle formulation as described in (Constantino et al., 2013).

Given a graph G as described in Section 2, let C be the set of all possible cycles with length 2. We assume that a cycle is a set of arcs. Define a variable z_c for each cycle $c \in C$:

$$z_c = \begin{cases} 1, & \text{if cycle } c \text{ is selected to for the exchange} \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

Denote by $V(c) \subseteq V$ the set of vertices which belong to cycle c. Let s_c be the score of cycle c, and let M be the objective value of the maximum pairwise matching. The model can then be written as follows:

$$\begin{aligned} \text{minimize:} \quad & \sum_{c \in C(k)} s_c z_c \\ \text{subject to:} \quad & \sum_{c: i \in V(c)} z_c \leq 1, \qquad \forall i \in V \\ & \sum_{c \in C} z_c = M, \\ & z_c \in \{0,1\}, \quad c \in C(k) \end{aligned}$$

The idea is that finding a maximum pairwise matching with the lowest cycle scores removes hard to match vertices from consideration and leaves vertices with high scores unmatched, allowing the greedy heuristic to make more matches. The greedy algorithm takes these unmatched vertices as input and tries to find as many additional matches as possible.

The reason that the Greedy algorithm is applied to the vertices left unmatched by this score-optimized maximum pairing is that the Greedy algorithm needs to find all cycles possible with the given vertices to rank them. Finding all cycles for the complete set of vertices quickly becomes impractical as the pool size increases, both because of memory constraints and the amount of time it takes.

3.5.3 Greedy Algorithm Definition

For a given method of defining cycle scores and a set of unmatched vertices, the algorithm for the Greedy Randomized Heuristic is defined as follows:

The algorithm starts by finding all possible cycles with the unmatched vertices and ranks them based on their score in ascending order. A solution is defined as a set of cycles with disjoint vertices, and the algorithm keeps track of the best solution found so far. The best solution is initialized as an empty set of cycles. It also initializes an empty set of all solutions found so far.

After these initializations the following steps are repeated until a user-set time limit is reached:

A new current solution is initialized as an empty set of cycles. A selection of vertices is made by iterating over all possible cycles in ascending order. A cycle is selected if none of its vertices are in the cycles of the current solution, except if selecting this cycle can only create solutions that have already been found. This selection process continues until λ cycles have been selected or the end of the list of cycles is reached. The algorithm then randomly picks one of the selected cycles, adds it to the current solution, and repeats the selection process. This continues until the selection can not find any more cycles with vertices that are not in the current solution. At that point the completed solution is added to the list of found solutions, and if the objective value is better than that of the best solution found so far it becomes the new best solution.

The technique of choosing the next addition to a solution randomly from the best λ options was introduced by Hart and Shogan (1987).

3.6 Dynamic Random Local Search

The final part of the overall algorithm is a Dynamic Random Local Search matheuristic. Local Search Matheuristics are a common method of finding high objective value solutions to MILP problems that are hard to solve exactly (Kleinberg & Tardos, 2005). They work by altering a solution in small ways to create a set of solutions that are similar to the current solution. This set is called a neighbourhood. The matheuristic then takes the neighbouring solution with the highest objective value and uses it as a new starting solution to repeat the process. The idea is that slight variations of a solution with a high objective value will have objective values that are close to that high value, with some hopefully improving upon it. By repeating the process of taking the best solution and slightly altering it the algorithm moves towards increasingly better solutions. The output of the algorithm is the best solution it encountered. The Local Search matheuristic proposed in this paper creates a neighbourhood by semi-randomly destroying cycles in the current solution.

3.6.1 Algorithm

The matheuristic starts by taking an initial solution provided by the first part of the overall algorithm and solving the k-cycle KEP exactly for all vertices it left unmatched. Any exact formulation of the k-cycle KEP can be used for this step, but the formulation that is solved the quickest is obviously preferred. The solution of this sub-problem is a set of cycles that are disjoint from the cycles in the initial solution, so the two solutions can be combined. Variables representing the current solution and best solution found so far are initialized, and their starting values are set to this combined solution.

The algorithm then iteratively tries to find better solution. In every iteration it creates one neighbour for every cycle in the current solution by destroying that cycle and σ random others. The vertices that made up the destroyed cycles are added back to the pool of unmatched vertices, after which the k-cycle KEP is solved for just this pool. Any exact formulation of the k-cycle KEP can be used for this step, but the formulation that is solved the quickest should be used to increase the number of iterations within the given time limit. The cycles of the optimal solution to this sub-problem plus the cycles in the current solution that were not destroyed then form a

neighbouring solution. The neighbour with the highest objective value is chosen as the current solution for the next iteration. If this solution has a higher objective value than the previous best solution it is stored. This algorithm iterates until a user-set time limit is reached or until a solution with an objective value equal to an upper bound on the problem is found.

The reason for creating one neighbour per cycle in the current solution and destroying that cycle is that it takes away some randomness from the process. In this way every cycle is at least destroyed once per neighbourhood which creates a more even spread around the current solution. This reduces the chance of having bad luck and missing that one cycle that would lead to an improvement.

Parameter σ starts with a value of α and is increased by α if τ subsequent iterations have not found a solution with a higher objective value than the previous best. Here α and τ are user-set parameters.

3.6.2 Upper Bounds

Because the matheuristic can stop once an upper bound on the objective value has been reached, it is useful to find one that is as low as possible. There are several options that vary in the time needed to compute them and how tight they are to the optimal solution of the k-cycle KEP. Three options for upper bounds are considered in this paper.

One upper bound is the optimal objective value to a relaxation of the k-cycle KEP where $k \to \infty$. This problem is solvable in polynomial time (Abraham et al., 2007). This upper bound will be referred to as the unlimited-k bound.

The other two upper bounds are given by the optimal values of the linear relaxations of the Cycle and Extended Edge formulations. Constantino et al. (2013) prove that the linear relaxation of the Cycle formulation always gives an upper bound that is lower than or equal to that of other exact formulations. However, the Cycle formulation has an exponentially increasing amount of variables as the pool size increases, whereas the EE formulation is compact in its number of variables (Constantino et al., 2013). This suggests that solving the linear relaxation in reasonable time is possible for larger pool sizes with the EE formulation than with the Cycle formulation.

3.7 Use as a Warm-Start

The solution produced is feasible for any formulation of the k-cycle KEP, which means it provides a lower bound on the objective value and a starting solution for solvers. Providing a lower bound to a solver that uses any kind of branching like branch-and-bound, branch-and-price, or branch-and-cut can significantly reduce solving time as branches of the search can be pruned (Huang et al., 2021). It can also be used as a starting point for other heuristics. If the solution produced by this algorithm is equal to the optimal value of the linear relaxation of a formulation, solving the unrelaxed model can be skipped entirely.

4 Results

Just like the methodology this section is structured into two main sections. The first concerns the results of the reproduction, and the second concerns the results of the heuristic algorithm.

4.1 Reproduction Results

The Cycle and Extended Edge formulations of the KEP are implemented and used to perform the same numerical experiments as in the work by Constantino et al. (2013). The results of these experiments are compared to those in the original paper and any findings are discussed.

4.1.1 Data

For the sake of acurracy the same four types of data as in the reproduced paper are used to test the formulations. The first is blood-test type data (Saidman, Roth, Sönmez, Ünver & Delmonico, 2006), made publicly available by Dickerson, Procaccia and Sandholm (2012). The other three types of data are obtained by using a data generation process proposed in the reproduced paper.

4.1.2 Computational Results

The computational results are shown in the same format as in the original paper for ease of comparison. These computations and all other computations in this paper were performed using the commercial Gurobi solver on a Java Virtual Machine with 16GB of RAM and a AMD Ryzen 7 3700X processor overclocked to 4.10 GHz. In the original paper the computations are done using a single core of a Quad-Core Intel Xeon processor at 2.66 GHz, with 16GB of RAM.

The table 1 contains results for the Cycle and EEF reproductions. For every combination of k and n, or one row in the table, 10 instances were generated. Column n denotes the amount of pairs in the data instances and column k denotes the maximum length of a cycle. Results are only shown for data instance sizes where at least on of the Just as in the original paper the C (Cycle) section has the following columns:

- t_c, the average time it took to find all cycles in the graph per data instance
- T, the average solving time
- $\#opt$, the number of instances solved to optimality within The time limit of 1800 seconds. The number in parentheses shows the amount of instances that could be created within memory constraints. A blank value means all 10 instances were solved to optimality.
- gap, the average LP gap achieved by a particular formulation, is defined as $\frac{UB-Opt}{UB} * 100\%$ where UB is the upper bound found by solving the linear relaxation of the formulation, and Opt is the optimal value found using the unrelaxed formulation.

4.2 Reproduction Findings

The main conclusions of the original paper stemmed from solving large instances of four different types of data with both formulations, where it was found that the Cycle formulation dominated

Table 1: Computational Results of the Reproduction

n	k	C				EE		C				EE	
		t_c	T	#opt	gap	T	gap	t_c	T	#opt	gap	T	gap
Blood-type test								*Medium density test instances*					
16	3	0	0		0.0	0	0.1	0	0		0.1	0	0.0
32		0	0		0.0	0	0.0	0	0		0.0	0	0.0
64		0	0		0.0	0.1	0.0	0	0.1		0.0	1.0	0.0
128		0	0.1		0.0	0.5	0.1	0	0.9		0.0	23.2	0.0
256		0.1	1.0		0.0	16.3	0.1	0.9	23.0		0.0	-	-
512		1.3	7.3		0.0	-	-	12.9	207.4		0.0	-	-
1024		16.7	97.3		0.0	-	-	154.0	782.6	7 (9)	0.0	-	-
2048		224.7	1225.5	8 (10)	0.0	-	-	-*	-*		-	-	-
16	4	0	0		0.0	0	0.1	0.1	0		0.0	0.2	0.0
32		0	0		0.0	0	0.0	0	0.1		0.0	0.2	0.0
64		0	0		0.0	0.4	0.0	0.1	3.5		0.0	4.4	0.0
128		0	1.6		0.0	12.6	0.0	7.7	139.1		0.0	338.4	0.0
256		4.8	57.5		0.0	-	-	-*	-*		-	-	-
512		95.0	1612.5	1 (1)	0.0	-	-						
16	5	0	0		0.0	0	0.0	0	0.1		0.0	0	0.0
32		0	0		0.0	0.1	0.0	0	3.8		0.0	0	0.0
64		0.1	1.1		0.0	1.6	0.1	6.3	183.5		0.0	3.6	0.0
128		3.6	49.2		0.0	96.3	0.0	-*	-*		-	-	-
16	6	0	0		0.0	0	0.0	0	0.1		0.0	0	0.0
32		0	0.2		0.0	0.1	0.0	2.3	81.8		0.0	0.2	0.0
64		0.5	11		0.0	1.7	0.0	-*	-*		-	-	-
Low density test instances								*High density test instances*					
16	3	0.1	1.2		0.0	0	0.0	0	0		0.0	0	0.0
32		0	0.1		0.2	0	0.0	0	0.1		0.0	0	0.0
64		0	0.0		0.0	0.2	0.0	0.0	0.1		0.0	2.0	0.0
128		0	0.0		0.0	1.2	0.0	0.4	4.0		0.0	60.6	0.0
256		0	1.7		0.0	29.4	0.0	2.3	23.4		0.0	-	-
512		0.8	19.4		0.0	-	-	33.0	264.2		0.0	-	-
1024		11.9	167.5		0.0	-	-	-*	-*		-*	-	-
2048		205.3	1360.1	1 (2)	0.0	-	-						
16	4	0.0	0.0		0.1	0.0	0.2	0	0		0.0	0	0.0
32		0	0.1		0.0	0.1	0.0	0.1	0.4		0.0	0.4	0.0
64		0	0.3		0.0	0.6	0.0	0.6	10.2		0.0	3.6	0.0
128		0	5.3		0.0	61.2	0.0	16.6	257.6		0.0	134.8	0.0
256		3.4	78.1		0.0	-	-	-*	-*		-*	-*	-*
512		82.9	1235.0	9 (10)		-	-						
16	5	0	0		0.0	0	0.14	0.0	0.1		0.0	0	0.0
32		0	0		0.0	0.2	0.0	0.5	11.1		0.0	0.4	0.0
64		0	2.1		0.0	7.0	0.0	-*	-*		-*	3.8	0.0
128		2.3	106.2		0.0	186.6	0.0	-*	-*		-*	-*	-*
16	6	0	0		0.1	0	0.2	0	0.4		0.0	0	0.0
32		0	0.3		0.0	0.4	0.0	9.8	82.2		0.0	0	0.0
64		0.6	31.6		0.0	4.6	0.0	-*	-*		-*	-*	-*

* Solving failed because of memory constraints instead of the limit on running time

the EEF in the blood-type and low density test instances, except when $k = 6$, and that the EEF proved superior for medium and high density data with $k > 3$. These conclusions will now be reviewed given the reproduced test results.

4.2.1 Effect of Different Methods for Cycle Detection

When comparing our results to those in the original paper, the first thing that stands out is the significantly smaller amount of time it took to find all cycles in the reproduction compared to the time it took in the original paper by Constantino et al. (2013). Solving the Cycle formulation first requires finding every possible cycle with a length of at most k. In the reproduction an adaptation of Johnson's Algorithm (Johnson, 1975) is used. Johnson's Algorithm finds all simple cycles of any length in a directed graph in $O(((|V| + |E|)|C|)$ time, where $|C|$ is the number of possible cycles. Hawick and James (2008) adapted this algorithm to only find cycles of length k or less. The Java code used to implement this algorithm was code made publicly available by Michail, Kinable, Naveh and Sichi (2020). In the original paper it is not said which method or algorithm is used to find all cycles, but the computational results in Table 1 seem to show that the algorithm by Hawick and James used in this paper is faster, as finding all possible cycles often takes orders of magnitude more time to find in the original paper, especially for high values of k. For example, when using blood-test type instances of size 64 the time it took 0.5 seconds on average to find all 6-cycles in our reproduction. In the paper by Constantino et al. (2013), finding all 6-cycles took 370 seconds on average for the same data type and instances of size 70, an increase by a factor of around 700. The same comparison for 3-cycles and instance sizes of 1024 and 1000 results in a smaller increase of a factor of around 20.

The authors also did not attempt to solve instances with more than 3 million cycles, noting that:"*the number of paths associated with the edge formulation increases sharply for larger values of* k". It is not mentioned why these instances were not studied. Most likely it was because of memory constraints or because the time it took to find that many cycles exceeded a certain time limit.

Our results offer some insights on this. Larger instances than those in the original paper were solved with the Cycle formulation for every combination of data type and k. Instances containing up to 49 million cycles were solved to optimality. This was done with the same amount of RAM as used in the original paper, suggesting that a time limit, not a lack of available memory, was the reason for not studying larger data instances. Then, it is likely that the use of an inefficient method to find all possible cycles was the bottleneck for studying larger instances with the Cycle formulation in the original paper.

4.2.2 Solving Times

Looking at the solving times of the actual Integer Programming models once the cycles have been found, we see that the difference in solving time is considerably smaller, with the solving times being roughly between two and ten times as long in the original paper. Unlike the difference in time for finding cycles, this difference is small enough that it could be caused by the difference in GPU frequency (2.2 GHz vs. 4.15 GHz), the use of multiple CPU cores, and/or advances in solver software in the eleven years since the paper was published.

4.2.3 Revisiting the Conclusions

The main conclusion of the original paper was that the Cycle formulation outperforms the EEF for blood-type and low density data instances when $k < 6$, and for medium and high density data instances when $k < 5$.

With the improved cycle detection algorithm the Cycle formulation outperforms the EEF even more in regards to solving time and solvable instance size for blood-type and low density instances, but it is still inferior for $k = 6$. For the medium and high density tests the EEF is now only the clear best option when $k > 4$, instead of $k > 3$. It should be noted however that every one of the Cycle tests that failed did so because of memory constraints, and that the memory used for the tests was relatively low for simulation standards.

A potential reason for the EEF 's outperformance at high values of k could be that with the Cycle formulation the amount of cycles and by extension the amount variables grow exponentially in k, causing memory issues. As seen in the results of this reproduction, this limits k and the size of the donor pool that can be solved in reasonable time with the Cycle formulation. The EEF does not have this issue. It creates at most $|V|$ copies of the original graph, and in those graphs, edges are only included if there is a cycle that contains that edge and the index-vertex. Since an edge can only be included once in a graph, the number of edges per graph-copy is at most $|V^2|$ edges in the worst case scenario where every vertex is connected to every other vertex. Since there is a constant amount of variables and restrictions per edge added, this results in a memory complexity of $\mathcal{O}(|V|^3)$ which is independent of k.

So while the Cycle formulation is faster for low values of k and low-density data instances, the sharp increase in the number of possible cycles for higher density data and higher values of k make it ineffective under those conditions. The EEF is slower in most cases, including the cases that are most representative of real-world data where $k \leq 4$ and density is lower, but its memory properties make it usable for larger instance sizes.

4.2.4 Use for Local Search Sub-Problem

The reproduced results give us two main take-aways for the formulations' use in the Local Search step of the matheuristic algorithm. The first is that the Cycle Formulation quickly becomes ineffective as instance sizes grow because of memory constraints. When k is set to 4 and the lowest density data is chosen, it can solve data instances containing up to 512 pairs. The second take-away is that when the Cycle Formulation *can* be used, it is faster than the EEF when solving the KEP regardless of data instance density for the cycle lengths considered in this paper, i.e, $k \leq 4$.

The reproduction data 1 displays these findings: for almost all data types and values of $k \leq 4$ the solving time of the EEF was the limiting factor on pool size. In contrast, the expanding the pool size of the Cycle formulation was limited by memory constraints for every data instance and value of k.

These results suggest that the Cycle formulation should be used in the Local Search step for as long as possible within memory constraints.

4.3 Matheuristic Algorithm Data

Accessing real Kidney Exchange program data is often difficult because of its medical and thus confidential nature, so over the years multiple data instance generators have been developed to accurately simulate real world Kidney Exchange data.

According to a literature survey by Mak-Hau (n.d.) the Saidman-generator (Saidman et al., 2006) is often used. It uses distributions of relevant patient traits like blood-type and panel antibody reactivity, and also includes the probability of husband-wife donor pairs, who have a higher chance of being a match than a random pair. Recently, Delorme et al. (2022) examined instances created by the Saidman-generator provided by Dickerson et al. (2012) and found that they "*differ from real-world instances in a range of parameters (such as edge density of the graph of potential kidney exchanges)*". Using these observations they find that optimal solutions to Saidman-generated data instances always have an objective value equal to the lowest of three specific upper bounds. A matheuristic to quickly find these bounds is provided, making solving the problem by MILP programming unnecessary. Using data from the UK Living Kidney Sharing Scheme they devise a way of generating instances that more accurately represent real world data ("delorme-generator"), and they "*confirm that [their] new upper bounds are not tight on these instances, meaning that our matheuristic does not provide a guarantee of optimality anymore, as is usually required in a KEP*".

As such, the data instances used in this thesis will be those generated by way of (Delorme et al., 2022). One of the authors also provides an online generator for these data instances (https://wpettersson.github.io/kidney-webapp//generator). As recommended in the paper, all instances are generated with the following pre-set settings found at the bottom of the web-page:

- *Use donor blood-group distributions from paper, determined by patient blood group*
- *Use SplitPRA values for determining cPRA*
- *Use BandXMatch-PRA0 to determine compatibility*

Additionally, tuning was enabled for instances of size 1000 for 100 iterations on blood groups and cPRA distributions but not on the number of donors per patient.

For every instance size ten instances were generated.

4.4 Results for the Matheuristic Algorithm

Before getting computational results on the whole matheuristic algorithm, the various parts are tested to determine which parameter values and other settings perform best.

4.4.1 Pre-processing

When generating problem instances using the delorme-generator it was found that in all instances a number of vertices could not be matched, and these vertices were removed by the pre-processing step. The average number of removed vertices was around 25% of vertices for all instance sizes. The pre-processing step took less than 0.1 seconds for all instances.

4.4.2 Pairwise Matching Results

The maximum pairwise matching problem described in Section 3.4 was solved using the pre-processed data. The average time to needed to find the maximum pairwise matching was less than 0.05 seconds for every instance size. The performance of this step is measured by the amount of vertices it can match from the pool of matchable vertices. The average percentage of matchable vertices it matched over the 10 instances per instance size is shown in Table 4.4.3.

4.4.3 Greedy Algorithm Results

As described in Section 3.5, a way of ranking cycles must be chosen before the Greedy algorithm can improve upon the pairwise matching. The six proposed methods for assigning scores to cycles were tested. Through trial and error testing it was found that the best method was to determine the score of matches by taking the product of the donor and patient's scores, and then setting the score of a cycle to the average of the scores of its matches. For all values of λ this method matched the most vertices, and the most vertices were matched overall for $= 50$. It was also found that running the Greedy algorithm for longer than a minute did not significantly improve the best objective value. This is a positive result since the intended function of the Greedy algorithm is to quickly generate a decent solution which can then be improved by the slower Local Search algorithm.

Once a scoring method is chosen the step between the pairwise matching and the Greedy algorithm described in Section 3.5.2 can be performed. It finds a pairwise matching with the maximum objective value such that the sum of the scores of the cycles in the solution is minimized. The Greedy algorithm is then applied to the vertices that are not matched in this solution. For every instance size Table 4.4.3 shows the average number of vertices matched by the maximum pairwise matching and the average number of vertices the Greedy algorithm matched when it was performed afterwards. The table also shows the average number of vertices matched per instance size if the Greedy algorithm is applied on the data before a portion of it is matched by the pairwise matching.

Table 2: The percentage of vertices matched by pairwise and/or Greedy matching

Instance size	1000	1250	1500	1750	2000	2250	2500	2750	3000
Pairwise matching %	37.0	38.4	39.9	41.7	42.1	42.7	41.6	45.1	46.2
Greedy % after pairwise	6.4	6.1	4.8	4.8	5.0	4.7	4.3	4.1	3.5
Pairwise & Greedy %	43.3	44.5	44.7	46.5	47.1	47.4	45.9	49.2	49.6
Only Greedy matching %	52.4	54.4	-	-	-	-	-	-	-

We see that the percentage of vertices matched by the pairwise matching slowly increases towards 50% as the instance size increases, and that the percentage of additional vertices matched by the Greedy algorithm seems to be decreasing as instance sizes grow.

Looking at the number of vertices matched when only using the greedy algorithm we see that it finds better solutions than by combining the pairwise matching and the Greedy algorithm on average but that it hits the memory limit for instance sizes above 1500, and is thus not usable to increase the pool size.

4.4.4 Local Search and Warm-Start Results

Using the found initial solutions we create improved solutions using the Local Search matheuristic algorithm. The value of these solutions as a warm-start solution to exact formulations of the k-cycle KEP was then evaluated.

Before being able to run the Local Search Matheuristic, we must choose an exact formulation of the k-cycle KEP to solve the sub-problems that create neighbouring solutions and choose parameters α and τ. For this paper we implemented the Cycle and Extended Edge formulations. Delorme, Manlove and Smeets (2023) performed experiments where the vertices are ranked in the match-matrix based on their degree, and found that the EE formulation can be improved by sorting the vertices either ascendingly or descendingly. We tested the four options on delorme-generator data and found that the Cycle formulation was solved the fastest. Testing these formulations was done with the GUROBI solver's default parameters. Out of the three possible variants of the Extended Edge formulation, the Extended Edge formulation with vertices ordered in descending order (EE-D) was solved the fastest. As such, the Cycle formulation will be used to solve the sub-problems in the Local Search algorithm, and the final solutions will be evaluated as warm-starts for both the Cycle and EE-D formulations. Trial-and-error tests showed that the Local Search performed best when $\alpha = 5$ and $\tau = 3$ so these parameter values will be used. The largest instance size that any of the formulations could solve was 1000, so this will be used as the starting instance size when testing the matheuristic algorithm.

Table 4.4.4 shows the results of running the complete matheuristic algorithm and using the produced solutions as warm-start solutions for the Cycle and EE-D formulations. The three upper bounds mentioned in Section 3.6.2 are also calculated. If a solution found by the matheuristic algorithm reached one of these bounds is was optimal by definition. Else, its optimality must be proven in the last step of the test when the exact formulations are solved.

The tests were performed on ten instances per instance size, and when solving the formulations with a warm-start the GUROBI solver's default settings were used except that for settings concerning heuristics. By, default the solver will try to find a heuristic solution as a lower bound before starting the solve. These lower bounds were found to be of poor quality compared to the matheuristic algorithm and as such this setting was turned off to save time. The time limit set on finding an initial solution using pairwise matching and the Greedy algorithm was set to 60 seconds, and the time limit set on the Local Search algorithm was 1800 seconds. Afterwards, the Cycle and EE-D were given 1800 seconds to solve the k-cycle KEP using these warm-starts.

One important finding that influenced the way the results are displayed needs to be addressed first. It was found that for every tested instance size the warm-start solution was not improved by the two exact formulations. However, it was also found that for most instances the matheuristic algorithm reached the optimal value by itself, which is confirmed when the solution reaches an upper bound. So while the exact formulations did not improve any solutions, only solving their relaxations was often enough to prove optimality. If the linear relaxations can not be found within 1800 seconds, and the matheuristic solution does not reach the upper bound found by solving the KEP for unlimited k, it is unknown whether the the solution is optimal.

For every instance size n the columns of Table () are divided in four groups in the following way:

- The Matheuristic category has three columns: *opt* shows the number of times the matheuristic algorithm reached an upper bound and thus found optimal an optimal solution. T_max shows the average time in seconds that the algorithm needed to reach the maximum in those cases. For instances where optimality is not proven because algorithm did not reach an upper bound, column *gap* shows the average distance of the algorithms best found solution value to the lowest upper bound.

- The Unlimited-k category concerns the unlimited-k upper bound. Column *opt* shows how many times the unlimited-k upper bound was equal to the optimal objective value, out of the times the optimal value was found. For the cases where the optimal value was not found, column *lowest* shows how many times the unlimited-k bound was equal to the lowest found upper bound.

- The EE-D category concerns the bound found by solving the linear relaxation of the Extended-Edge formulation with vertices ordered in descending order. Column $found$ shows how many times the linear relaxation was able to be solved within time and memory limits, and T_{max} shows the average time needed in seconds. Column *opt* shows the number of times the bounds were equal to the optimal objective values, out of the times both the upper bound and the optimal value were found. 0he Cycle category is the same as the EE-D category except that the linear relaxation of the Cycle formulation is solved instead.

	Matheuristic			Unlimited-k		EE-D relaxation		Cycle relaxation	
n	opt	T_{max}	avg gap	opt	lowest	found	T_{max}	found	T_{max}
1000	10	410.0	-	5/10	5/10	10	40.1	10	71.8
1250	10	429.2	-	6/10	6/10	10	86.7	10	132.0
1500	9	423.6	1.0	6/9	7/10	10	201.1	10	496.5
1750	10	677.6	-	8/10	8/10	10	396.3	4	805.7
2000	7	763.3	1.34	7/7	9/10	10	1335.5	0	-
2250	8	935.8	1.0	8/8	10/10	10	1335.5		
2500	7	874.9	1.0	7/7	8/8	8	1239.3		
2750	3	1095.0	1.3	3/3	-/-	0	-		
3000	7	1282.7	2.0	7/7	-/-	0	-		

Table 3: Numerical results of using the warm-start solution for the Cycle and EE-D formulations

Starting from the largest instance size that exact formulations can solve without a warm start, the instance size was increased until the relaxations of the EE-D and Cycle formulations could not be calculated within 1800 seconds anymore.

When running the matheuristic algorithm for 1800 seconds we see that it reaches an optimal solution for a majority of instances. Furthermore, for the rest of the solutions it found it was not proven that they were sub-optimal. The exact formulations were unable to improve or confirm the (sub-)optimality of any warm-start solution that had an objective value lower than the solution of their linear relaxation. The exact formulations functioned more as a kind of "check", since optimality was only confirmed when the matheuristic already reached an upper bound by itself.

We also see that the Cycle formulation quickly fails because of memory constraints. As mentioned in Section 3.6.2, the Cycle formulation's linear relaxation always provides equal or

lower upper bounds than that of the EE formulation. More matheuristic solution could be shown to be optimal if a formulation is used that is as tight as the Cycle formulation, but does not have memory problems.

Another interesting finding is that the unlimited-k upper bound was either equal to the upper bounds set by the linear relaxations of the exact formulation or higher by exactly 1, while being much easier to find as shown earlier in this section. Column *lowest* in Table (??) shows that the unlimited-k bound is the equal to the lowest bound for a majority of instances.

In general these are positive results. Optimal solutions or solutions that are very close to it are found for instance sizes that are up to three times as large as the largest instance size that the exact formulations can solve on their own within the same amount of time.

5 Conclusion

In this paper we proposed a modular matheuristic algorithm to find warm-start solutions for exact formulations of the k-cycle KEP and performed numerical experiments. The matheuristic algorithm consists of two major parts: first an initial solution is created by improving an easily findable solution to a restricted version of the problem using a fast-working Greedy algorithm. This initial solution is then used as the starting point for a dynamic randomized Local Search algorithm that makes use of an exact formulation of the problem to solve sub-problems. The use of these solutions as warm-starts was tested with reproductions of the Cycle and Extended Edge formulations by Constantino et al. (2013).

The goal of the algorithm was to provide warm-start solutions to exact formulations such that they could solve larger problem instances in reasonable time than they could without using the warm-start solutions. In this regard our matheuristic algorithm partly succeeded. Optimal or near-optimal solutions were found by the algorithm for instances that were up to three times as large as the largest instance size that is solvable to optimality using the exact formulations. However, the exact algorithms were only able to confirm the optimality of warm-start solutions. If a warm-start solution was not already optimal by itself the exact formulations could not use the warm-start to find a solution with a higher objective value.

The modular structure of the matheuristic algorithm was devised with the intention that it allows for gradual improvement through future research, and here we provide some suggestions. The Greedy algorithm can be improved by researching better ways for it to rank cycles and by performing in-depth analysis of its user-set parameters instead of the trial-and-error tests performed for the results of this paper. The Local Search algorithm could also benefit from more extensive parameter analysis, but most importantly it should be tested with more recent and faster-solving formulations of the k-cycle KEP, since its iteration time is largely determined by how quickly k-cycle KEP sub-problems are solved. Lastly, this algorithm could be evaluated for use with more complex and realistic versions of the KEP, like the KEP with multiple hierarchical objectives, or the KEP with altruistic donors.

6 Code and Data Instances

The source code that was used to find all numerical results in this paper and the data instances used are available at `https://github.com/denzlin/BaThesis`.

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* (p. 295–304). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/1250910.1250954` doi: 10.1145/1250910.1250954

Biró, P., Haase-Kromwijk, B., Andersson, T., Ásgeirsson, E. I., Baltesová, T., Boletis, I., ... van der Klundert, J. (2019, July). Building kidney exchange programmes in europe—an overview of exchange practice and activities. *Transplantation*, *103*(7), 1514-1522. doi: 10.1097/TP.0000000000002432

Clausen, J. (2003). Branch and bound algorithms-principles and examples.. Retrieved from `https://api.semanticscholar.org/CorpusID:16580792`

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. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0377221713004244` doi: https://doi.org/10.1016/j.ejor.2013.05.025

Delorme, M., García, S., Gondzio, J., Kalcsics, J., Manlove, D. & Pettersson, W. (2023). New algorithms for hierarchical optimization in kidney exchange programs. *Operations Research*, *0*(0).

Delorme, M., García, S., Gondzio, J., Kalcsics, J., Manlove, D., Pettersson, W. & Trimble, J. (2022). Improved instance generation for kidney exchange programmes. *Computers Operations Research*, *141*, 105707. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0305054822000107` doi: https://doi.org/10.1016/j.cor.2022.105707

Delorme, M., Manlove, D. & Smeets, T. (2023). Half-cycle: A new formulation for modelling kidney exchange problems. *Operations Research Letters*, *51*(3), 234-241. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0167637723000329` doi: https://doi.org/10.1016/j.orl.2023.02.009

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* (pp. 711–718).

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, *17*, 449–467. doi: 10.4153/CJM-1965-045-4

Hart, J. & Shogan, A. W. (1987). Semi-greedy heuristics: An empirical study. *Operations Research Letters*, *6*(3), 107-114. Retrieved from `https://www.sciencedirect.com/science/article/pii/0167637787900216` doi: https://doi.org/10.1016/0167-6377(87)90021-6

Hawick, K. A. & James, H. A. (2008). Enumerating circuits and loops in graphs with self-arcs and multiple-arcs. In *International conference on foundations of computer science.* Retrieved from https://api.semanticscholar.org/CorpusID:5171345

Huang, L., Chen, X., Huo, W., Wang, J., Zhang, F., Bai, B. & Shi, L. (2021). *Branch and bound in mixed integer linear programming problems: A survey of techniques and trends.* Retrieved from https://arxiv.org/abs/2111.06257

Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, *4*(1), 77-84. Retrieved from https://doi.org/10.1137/0204007 doi: 10.1137/0204007

Kleinberg, J. & Tardos, E. (2005). *Algorithm design* (1st ed.). Pearson Education.

Mak-Hau, V. (n.d.). On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. , *33*(1), 35–59. Retrieved from https://doi.org/10.1007/s10878-015-9932-4 doi: 10.1007/s10878-015-9932-4

Michail, D., Kinable, J., Naveh, B. & Sichi, J. V. (2020, May). Jgrapht—a java library for graph data structures and algorithms. *ACM Trans. Math. Softw.*, *46*(2).

Saidman, S. L., Roth, A. E., Sönmez, T., Ünver, M. U. & Delmonico, F. L. (2006, March 15). Increasing the opportunity of live kidney donation by matching for two- and three-way exchanges. *Transplantation*, *81*(5), 773–782. doi: 10.1097/01.tp.0000195775.77081.25

Wallis, C. B., Samy, K. P., Roth, A. E. & Rees, M. A. (2011, 03). Kidney paired donation. *Nephrology Dialysis Transplantation*, *26*(7), 2091-2099. Retrieved from https://doi.org/10.1093/ndt/gfr155 doi: 10.1093/ndt/gfr155