New algorithms for hierarchical optimisation in kidney exchange programmes

Maxence Delorme[(1)], Sergio García[(1)], Jacek Gondzio[(1)], Joerg Kalcsics[(1)], David Manlove[(2)], William Pettersson[(2)]

(1) *School of Mathematics, University of Edinburgh, United Kingdom*
(2) *School of Computing Science, University of Glasgow, United Kingdom*

Corresponding author `maxence.delorme@ed.ac.uk`, phone +44 0131 650 5870

Technical Report - ERGO-20-005

Abstract

Kidney exchange programmes (KEPs) across the world help match donors and recipients to identify kidney transplantations. Almost all KEPs use a hierarchical set of objectives to determine an optimal set of transplants to perform, and integer linear programming is often used to find such optimal matchings. In this work, we identify the barriers in existing mathematical models and we propose innovative techniques to remove these barriers, vastly reducing solution times and allowing us to significantly increase the potential size of KEPs.

Our techniques include two methods to avoid unnecessary variables, as well as a diving algorithm that reduces the need to solve multiple complex integer linear programming models while still guaranteeing the optimality of a final solution. We also show that it is possible to transition between two distinct existing formulations (namely the cycle formulation and the position-indexed chain-edge formulation) between the optimisation of two successive objective functions. We use this technique to devise a new algorithm which, among other features, intelligently exploits the different advantages of the prior two models.

We demonstrate the performance of our new algorithms with extensive computational experiments modelling the UK KEP, where we show that our improvements reduce running times by three orders of magnitude compared to the cycle formulation. We also provide substantial empirical evidence that the new methodology offers equally spectacular improvements when applied to modelling the objectives used by Spain and the Netherlands, suggesting that our approach is not just viable, but a significant performance improvement, for many different KEPs across the globe.

Keywords: Kidney exchange programme, hierarchical optimisation, exact algorithms, objective diving, preprocessing.

1 Introduction

In 2016 over 750 million people world wide were impacted by impaired kidney function due to Chronic Kidney Disease (CKD) [7], and over 1.2 million deaths in 2017 were attributable to CKD [32]. Despite best efforts by researchers, no cure exists for CKD, and for a patient in the

final stage of CKD (called end stage renal disease), the options are limited to either dialysis or transplantation. Of these, dialysis is more expensive, and offers both a worse quality of life and a worse life expectancy for the patient [6]. Transplantation is therefore the better option for a patient, but requires that a donor kidney be found. Donor kidneys come from either a deceased or living donor, with better outcomes for patients who receive a transplant from a living donor [24, 40]. However, to receive a kidney from a living donor, a patient, also called a donor recipient or simply recipient, must find a willing and medically compatible donor. A *Kidney Exchange Programme* (KEP) increases the rate of living donor kidney transplantation by alleviating the requirement that recipients find a medically compatible donor [33, 34]. As the life expectancy of a recipient on dialysis is typically 5 years [1], not only does each transplant facilitated by a KEP transform a recipient's quality of life, it is clear that KEPs truly are life-saving OR applications.

Recipients wishing to join a KEP will pair up with one or more willing donors, who may or may not be medically compatible with the recipient. At certain intervals a KEP will perform a *matching run*, taking all donors and recipients that have entered the system, creating a graph representing all potential transplants, and determining a set of *exchanges* to perform. The simplest exchange is two sets of paired donors and recipients (d_1, r_1) and (d_2, r_2), where d_1 is paired with r_1 and d_2 is paired with r_2. Such an exchange is called a 2-way exchange, and within this exchange donor d_1 donates a kidney to recipient r_2, and donor d_2 donates a kidney to recipient r_1. Due to their nature, exchanges such as these are called *cycles*. A cycle containing n such pairs is said to be a cycle of size n. Some KEPs allow the participation of *non-directed donors*. These donors are willing to donate a kidney to a recipient in the KEP despite not being paired with any recipient themselves. These donations will then trigger further donations from a donor whose recipient has received a kidney transplant. Such structures are called *chains*, and usually end with a donation to a deceased-donor waiting list.

The goal of a KEP is to perform matching runs that will provide the best outcome for the donors and recipients involved. That is, in each matching run the KEP will need to find the "best" set of exchanges. An obvious desirable goal for a KEP is to maximise the number of transplants, but often many solutions are found that each attain some maximal number of transplants. Recent publications show that KEPs across Europe all have slightly differing criteria for breaking such ties [8, 9]. Due to the number of potential solutions found, often this tie breaking involves multiple criteria. These can include the number of transplants between a donor and recipient with identical blood groups, how long a given recipient has either been on dialysis, or waiting in a KEP, the tissue-type sensitivity of the recipients in each matching, or logistical components such as how many different hospitals or transplantation centres must be involved. Some of these criteria can also be combined in a scoring or weighting function, with each potential transplant being given some score based on some pre-determined function. Tie breaking is often performed hierarchically (sometimes known as lexicographically). The criteria are presented as an ordered list of functions $(f_1, f_2, \ldots, f_n)$ which must all be optimised in turn. That is to say, to solve such a problem, a solver will find a solution x_1 that optimises the first objective f_1, achieving some optimum $f(x_1) = o_1$, and then add the constraint $f_1(x) = o_1$ before moving onto the next objective. This is repeated until an optimal solution is found.

If only 2-way exchanges are allowed in a KEP, and all optimality criteria can be expressed

as a score or weight applied to each potential transplant, then an optimal solution can be found in polynomial time by modelling the problem as a weighted matching problem [35]. However even if only 3-way exchanges are allowed in addition to 2-way exchanges, it is NP hard to even determine a set of exchanges that maximises the number of transplants [3]. As such, optimal sets of exchanges are often found using Integer Linear Programming (ILP) techniques.

The first two ILP models for determining an optimal set of exchanges were the *edge formulation* and *cycle formulation* [36]. Since then, new compact models have been introduced [14], including the *Position-Indexed Cycle Edge Formulation* (PICEF) [20] which we use for testing. Other techniques for solving such problems include branch-and-price [3] and branch-and-price-and-cut [27]. A survey and comparison of ILP models for maximising the number of transplants is given in [29]. Further studies have considered models that maximise the expected number of transplants given some failure rate of arcs [26, 4, 13, 31], methods of maximising potential fallback solutions if arcs can fail [39], and models that consider a dynamic, or online, KEP — the evolution of a KEP over time [17, 19, 12, 22]. Other approaches have been used for finding exchanges, including parameterised complexity [28] and randomized mechanisms [11, 10]. However, in most, if not all, such studies the models and techniques are compared solely on the time it takes to solve for a single objective, namely determining a set of exchanges that maximises the number of transplants. This is a useful metric for comparing different models, but does not reflect real-world applications which consider other metrics as well [8, 9].

One avenue for increasing the number of transplants arranged by KEPs is to increase the number of donor-recipient pairs in the pool. As exact OR techniques are used to find optimal exchanges, merging two or more distinct pools is guaranteed to be at least as good as keeping them split, and will likely increase the number of transplants. However, current ILP formulations for KEPs struggle to solve these larger pools, either because the formulation cannot model certain aspects of the KEP (e.g., we will see later on that PICEF cannot model one of the UK objective functions), or because the formulation creates models that are too large to be tractable by real-world solvers (e.g., the cycle formulation struggles past 300 donor-recipient pairs as shown in our experiments).

1.1 Existing kidney exchange programmes

Many existing European KEPs are based on finding sets of exchanges that are optimal according to hierarchical sets of objectives. This is the case for the following three KEPs in Europe [8, 9], and we now describe their objectives as follows.

- The UK KEP, run by NHS Blood and Transplant (NHSBT) and called the UK Living Donor Kidney Sharing Scheme (UKLKSS), which optimises over the following five objectives hierarchically:

  1. maximise the number of effective 2-way exchanges[1],
  2. maximise the number of transplants,
  3. minimise the number of 3-way exchanges,

[1] An effective 2-way exchange is a n-way exchange containing an embedded 2-way exchange.

4. maximise the number of cross arcs[2], and
5. maximise the sum of the scores.

- The Spanish KEP, run by Organización Nacional de Trasplantes, which optimises over the following five objectives hierarchically:
  1. maximise the number of transplants,
  2. maximise the number of distinct exchanges selected,
  3. maximise the number of cross arcs,
  4. maximise the number of highly sensitised patients selected[3], and
  5. maximise the sum of the scores.
- The Netherlands KEP, run by Nederlandse Transplantatie Stichting, which optimises over the following six objectives hierarchically:
  1. maximise the number of transplants,
  2. maximise the number of transplants between a donor and recipient with the same blood group,
  3. prioritise the transplants to hard-to-match patients[3],
  4. minimise the length of the largest cycle selected,
  5. maximise the number of distinct transplant centres involved in any one cycle, and
  6. maximise the longest waiting time experienced by any selected recipient.

The fifth objective of the Netherlands KEP aims to spread the logistical cost of an exchange over a broader region, while the sixth objective gives preference to those recipients who have been waiting the longest. We note that the UKLKSS currently limits chains to have at most three donors, while the Dutch KEP has an upper bound of four, and the Spanish KEP has no hard upper bound. In this paper we will extend this cap on the number of donors in a chain in the UK to four. We do this in anticipation of likely future changes to the UKLKSS that should increase the number of kidney transplants performed in the UK. In such a scenario, the UKLKSS objectives would have to be revised. Whilst a new set of objectives has not been ratified by NHSBT, for the purposes of this paper we will consider the following set of objectives that we believe could be appropriate to the setting where longer chains are permitted: maximise the number of transplants, minimise the number of chains of length four, minimise the number of 3-way exchanges and chains of length three, maximise the number of cross arcs, and maximise the sum of the scores.

Some KEPs, like the United Network for Organ Sharing in the US, only use a single objective when determining an optimal set of exchanges, but do so using a complex weighting formula to take into consideration factors such as recipient ages, blood types, or waiting times [2].

[2]Cross arcs, which we define formally in Section 2, represent a form of "fault tolerance".

[3]Definitions of highly-sensitised or hard to match patients vary across different KEPs.

1.2 Our contribution

After introducing the necessary definitions for KEPs, we review two well-known ILP models from the literature in Section 2, namely the cycle formulation and the position-indexed chain-edge formulation. As controlling the model size is a critical issue in these two formulations, we describe in Section 3 three new techniques that dramatically reduce the number of variables in the two models, and thus deliver a step change in their performance: a cycle/chain deactivation algorithm that removes unnecessary variables because they cannot belong to any optimal assignment, a diving algorithm to remove the necessity to solve complex ILP models, and a dominated chain detector to remove variables that can only deteriorate the objective functions. We also show that it is possible to transition from PICEF to the cycle formulation between the optimisation of two objective functions, allowing hybridised algorithms using both PICEF and the cycle formulation. We then show in Section 4 that our approaches can be up to three orders of magnitude faster than the basic cycle formulation on a large set of instances for the UKLKSS, and we show that our tools can be adapted to tackle other European KEPs such as those running in Spain or in the Netherlands. Some conclusions are drawn in Section 5 and we outline some future research directions.

1.3 Other kidney exchange algorithms in the literature

In this paper we compare our new algorithms against the cycle formulation and PICEF. Among other algorithms for KEPs that have been proposed in the literature, we remark that (i) the *edge-assignment formulation* and the *extended edge formulation* [15] were shown by the authors to be computationally outperformed by the cycle formulation for realistic instances and were not able to solve any instance with 300 patients or more, (ii) the formulation based on the *prize-collecting travelling salesman problem* [5] was empirically shown to be less effective than the extended edge formulation (see [29]), and (iii) the promising branch-and-cut-and-price from Lam and Mak-Hau [27] did not consider non-directed donors. Other algorithms were tested in [20], but in almost all cases, PICEF turned out to be the most effective. It is worth mentioning that some of the previously mentioned methods work particularly well under specific conditions, such as when the cycle size limits or chain length limits are very large (e.g., the experiments in [15] were performed on instances with cycles of size up to 6, and those in [29] were performed on instances with chains of length up to 20, when limited), or when instances display a high proportion of compatibility. These conditions, however, are not present in the real-world scenarios we study.

2 Background

2.1 Definitions

We will use *recipient*, or r, to refer to a person who has chronic kidney disease and is waiting for a transplant, and *donor*, or d, to refer to a donor willing to donate a kidney. This avoids the ambiguity of using the term patient, as both the donors and recipients may, depending on which exchanges are selected, undergo surgery and be patients. A donor in a KEP may either be paired with a recipient, forming a *recipient/donor pair* where the donor is only willing to donate if their paired recipient also receives a kidney, or they may be a *non-directed* donor (sometimes

called an *altruistic* donor) who is willing to donate a kidney without having an identified paired recipient.

Given a recipient r and a donor d (possibly but not necessarily from the same recipient/donor pair), we say that r and d are *compatible* if the donation of a kidney from d to recipient r is deemed medically viable. Otherwise, we say that r and d are *incompatible*. Incompatibilities may arise from blood typing, tissue typing, or any of a number of other reasons as determined by specialists.

We represent compatibilities by a *compatibility graph*, a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ whose vertex set $\mathcal{V}$ contains one vertex for each non-directed donor, one vertex for each recipient, and a dummy *sink* vertex S, which represents a donation being made to a deceased-donor waiting list, although it can also represent a *bridge* donation that may act as a non-directed donor in the next matching run. By associating only recipients to vertices, we allow for scenarios in which a recipient is paired with multiple donors. This can occur as different donors may be compatible with a given recipient, and so pairing with multiple donors may improve a recipient's chance of being matched.

The arc set $\mathcal{A}$ of $\mathcal{G}$ contains all the arcs (u, v) where a donor corresponding to u (either the non-directed donor u, or a paired donor of the recipient u) is compatible with the recipient v, as well as one arc from every vertex to S (except the loop from S to S). Where a recipient r_1 has two (or more) donors, both of whom are compatible with a common recipient r_2, we introduce parallel arcs. In certain scenarios, each arc may also be given a score or weight which is associated with the corresponding transplantation.

Given this definition of a compatibility graph, for any vertex v representing recipient r, we will use "recipient v" to refer to r, and "donor v" to refer to any of the donors paired with recipient r. Note that any ambiguity in selecting the correct donor can be ascertained if an arc leaving v is selected, or is irrelevant if no arc leaving v is selected.

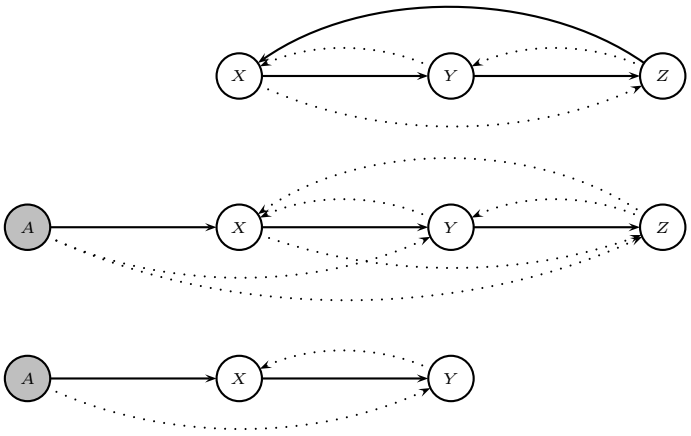
A feasible *matching* M is a subgraph of $\mathcal{G}$ in which each non-directed donor has no incoming arc and one outgoing arc, and each recipient has exactly one outgoing and one incoming arc. A matching can be decomposed into connected subgraphs called *exchanges*. These exchanges come in two particular types – cycles and chains – which we define now. A *cycle* is a subgraph of M that contains recipients $r_1, r_2, \ldots, r_k$ such that there is an arc from r_i to r_{i+1} for $i \in \{1, 2, \ldots, k-1\}$, as well as one arc from r_k to r_1. Such a cycle has length k. Upper bounds on the lengths of cycles in KEPs are common, as to maintain the integrity of a KEP all transplants relating to a given cycle should be performed simultaneously. A *chain* is a directed path in M starting at a non-directed donor, containing a further k recipients $r_1, r_2, \ldots, r_k$, and terminating at S. Such a chain is said to have length $k+1$ (i.e., we count the number of arcs when determining the length of a chain) and counts as $k+1$ transplants. Note that chains of length one contain a single arc from a non-directed donor directly to S. While it would seem that such a chain (corresponding to a transplantation from a non-directed donor to a recipient on a deceased-donor waiting list) is not strictly related to a KEP, such chains are included in the modelling to allow a fair comparison between alternative solutions (for example between one solution comprising a chain of length 3 and another solution comprising a chains of length 1 and a pairwise exchange). This will be particularly important for the cycle formulation which we introduce in Section 2.2.

Unlike cycles, chains can be performed non-simultaneously while still ensuring that each

donor does not donate a kidney until after their paired recipient has received a kidney, so limits on their lengths tend to be more relaxed. Limits to chain lengths are still sometimes enforced, as the dynamic nature of a KEP means a recipient may be better off waiting for a later matching run rather than being towards the end of a very long chain.

One important feature of the UKLKSS is the concept of a *cross arc*. Given an exchange consisting of some set of vertices V and some set of arcs A, a *cross arc* is any arc that is between two vertices of V but is not in A. Such an arc adds robustness to the exchange. If part of the exchange fails, either because a donor or recipient was unable or unwilling to proceed with the procedure, or an identified transplant was determined to be incompatible, then a cross arc may allow part of this exchange to continue. Such fallback options are useful as they are easy to detect, do not require any re-optimisation, and also guarantee that any recipients who are not in the failed part of an exchange are still matched. This is important for the well-being of both donors and recipients, as offering and then withdrawing a donation is traumatic to people in such situations. We detail in Figure 1 all the possible cross arcs in cycles of size three and chains of length three and four. Grey nodes represent non-directed donors while white nodes are used for recipient/donor pairs.

Figure 1: Example of cross arcs (dotted lined) in cycles of size 3 and chains of length 3 and 4



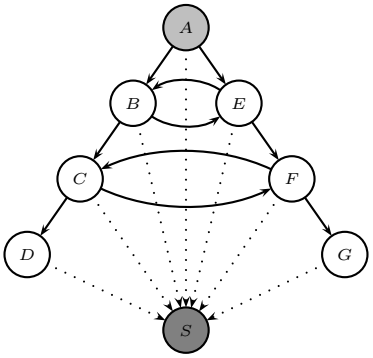
Let us consider for example that chain $A \to X \to Y$ was selected in a matching, and say that recipient/donor pair X was unable to participate in the exchange because the health of the donor would not allow him to go through a surgical intervention. Then, cross-arc $A \to Y$ would still allow recipient/donor pair Y to be matched.

In the following, we introduce an example that will be used throughout the paper to describe the behaviours of our algorithms.

Example 1. *Let us consider the following instance with one non-directed donor A, 6 recipients B, C, D, E, F, and G, each of whom is paired with one donor, and the compatibility graph described in Figure 2. Where relevant, all arcs have unitary score. For conciseness, where it is obvious we are referring to a donor, we will use e.g. donor B to refer to the donor paired with recipient B.*

Non-directed donor A can give their kidney to (or is compatible with) the recipients B and E. Donor E is compatible with recipient B and donor B is compatible with recipient E. As a result, vertices B and E can form the cycle of size two $[B, E]$. Another cycle of size two is $[C, F]$. As

Figure 2: Compatibility graph for Example 1



donor E is compatible with recipient F, and donor F is compatible with recipient G, non-directed donor A and vertices E, F, and G can form the chain of length four $A \to E \to F \to G \to S$. Other chains of length four are, for example, $A \to E \to B \to C \to S$ (which includes cross arc $B \to E$) or $A \to B \to C \to D \to S$. The latter chain can also be shortened to form, for example, the chain of length three $A \to B \to C \to S$ or the chain of length two $A \to B \to S$.

2.2 Cycle formulation

The cycle formulation for KEPs was proposed by Roth, Sönmez, and Ünver [36]. Even though the name refers explicitly to "cycles", it can handle chains as well. In the rest of the paper, we use the original name "cycle formulation" when referring to the model and we use "cycles/chains" when we refer to a generic variable of the model that can be either a cycle or a chain. If one of the techniques we propose is specific to one of the two structures, then we clearly identify it by only using the term "chain" or "cycle". For the sake of clarity, we omit the last edge "$\to S$" when talking about chains in the cycle formulation.

In the cycle formulation, a list of every feasible cycle/chain needs to be found beforehand. As our approach is tailored to the UK kidney exchange program, we assume that every feasible cycle has size at most three (i.e., it involves at most three recipient/donor pairs), and that every feasible chain has length at most four (i.e., it involves exactly one non-directed donor and at most three recipient/donor pairs). Let us consider the following notation:

- $\mathcal{N}$ and $\mathcal{P}$ are the set of non-directed donors and recipient/donor pairs, respectively.
- $\mathcal{C}$ is the set of feasible cycles/chains, $\mathcal{C}_{XL}$ is the set of feasible chains with length four, and $\mathcal{C}_L$ is the set of feasible cycle/chains with length three.
- For a cycle/chain c, we call $B(c)$ the number of cross arcs in c, $S(c)$ the sum of the arc scores included in c, and $V(c)$ the set of vertices (either in $\mathcal{N}$ or $\mathcal{P}$) which belong to c.

Let us also introduce binary decision variables x_c that take value 1 if cycle/chain c is selected, and 0 otherwise ($c \in \mathcal{C}$) and let us consider the following objective functions (see Manlove and

O'Malley [30]):

$$\max \quad z_1 = \sum_{c \in \mathcal{C}} |V(c)| \; x_c \tag{1}$$

$$\min \quad z_2 = \sum_{c \in \mathcal{C}_{XL}} x_c \tag{2}$$

$$\min \quad z_3 = \sum_{c \in \mathcal{C}_L} x_c \tag{3}$$

$$\max \quad z_4 = \sum_{c \in \mathcal{C}} B(c) \; x_c \tag{4}$$

$$\max \quad z_5 = \sum_{c \in \mathcal{C}} S(c) \; x_c. \tag{5}$$

Let us also denote by KEP_{z_k} the problem of solving objective function k ($k = 1, \ldots, 5$) to optimality and let us define $\bar{z}_1, \ldots, \bar{z}_5$ the optimal objective values obtained when solving $\text{KEP}_{z_1}, \ldots,$ KEP_{z_5}. The generic cycle model (called F_k^C hereafter) to solve KEP_{z_k} is as follows:

$$(F_k^C) \quad \min / \max \quad z_k \tag{6}$$

$$\text{s.t.} \quad \sum_{c \in \mathcal{C}: v \in V(c)} x_c \leq 1, \quad \forall v \in \mathcal{N} \cup \mathcal{P}, \tag{7}$$

$$z_j = \bar{z}_j, \qquad \forall j = 1, \ldots, k-1, \tag{8}$$

$$x_c \in \{0, 1\}, \qquad \forall c \in \mathcal{C}. \tag{9}$$

Objective functions (1)-(5) maximise the number of transplants, minimise the number of chains of size four selected, minimise the number of cycles/chains of size three selected, maximise the number of cross arcs in the selected cycles/chains, and maximise the sum of the arc scores in the selected cycles/chains, respectively. These objective functions correspond to a reasonable extension of the current UKLKSS objectives (see Manlove and O'Malley [30]) to scenarios allowing for chains of size four. Constraints (7) ensure that donors and recipients appear in at most one of the selected cycles/chains, and constraints (8) make sure that optimal values $\bar{z}_1, \ldots, \bar{z}_{k-1}$ found at previous iterations are not degraded when optimising objective function z_k.

We report in Table 1 the outputs obtained by the cycle formulation with an ILP solver when applied to Example 1. The first four columns give an identifier for each cycle/chain c, its length, the number of cross arcs in c, and the set of vertices that compose the cycle/chain. The five following columns contain the value taken by x_c after solving the cycle formulation with each of the five objective functions.

Example 1. (resumed) *The optimal solution value for KEP_{z_1} is 5 and can be obtained by selecting cycles $[C, F]$ and $[B, E]$, and the chain A. The optimal solution values for KEP_{z_2},…,KEP_{z_5} are 0, 0, 0, and 4, respectively. These values can be obtained by selecting the same set of cycles and chains.*

2.3 Position-indexed chain-edge formulation

The Position-Indexed Chain-Edge Formulation (PICEF) for KEPs was proposed by Dickerson et al. [20]. In the following, we present a different (but equivalent) description of PICEF than

Table 1: Results of the cycle formulation with an ILP solver for Example 1

id	len.	# c. a.	comp.	$\bar{z}_1 = 5$	$\bar{z}_2 = 0$	$\bar{z}_3 = 0$	$\bar{z}_4 = 0$	$\bar{z}_5 = 4$
0	2	0	$[B, E]$	1	1	1	1	1
1	2	0	$[C, F]$	1	1	1	1	1
2	1	0	A	1	1	1	1	1
3	2	0	$A \to B$	0	0	0	0	0
4	3	2	$A \to B \to E$	0	0	0	0	0
5	4	2	$A \to B \to E \to F$	0	0	0	0	0
6	3	0	$A \to B \to C$	0	0	0	0	0
7	4	1	$A \to B \to C \to F$	0	0	0	0	0
8	4	0	$A \to B \to C \to D$	0	0	0	0	0
9	2	0	$A \to E$	0	0	0	0	0
10	3	2	$A \to E \to B$	0	0	0	0	0
11	4	2	$A \to E \to B \to C$	0	0	0	0	0
12	3	0	$A \to E \to F$	0	0	0	0	0
13	4	1	$A \to E \to F \to C$	0	0	0	0	0
14	4	0	$A \to E \to F \to G$	0	0	0	0	0

the one proposed in [20]. The main idea behind PICEF is to handle cycles and chains in two separate structures. While there is still a complete enumeration of every feasible cycle, each chain is now represented by a path in a graph $\mathcal{G}'$ with $3|\mathcal{P}|+|\mathcal{N}|+1$ nodes. We differentiate each of the three copies of set $\mathcal{P}$ with an index ℓ where $\ell = 1, 2, 3$. A path initiates with a non-directed donor and ends with the dummy node S. The graph $\mathcal{G}'$ is composed of 4 subgraphs:

- Subgraph 1 is a copy of the compatibility graph that only includes the arcs coming from the non-directed donors. Additional arcs link each non-directed donor to the sink node S.
- Each of subgraphs 2 and 3 is a copy of the compatibility graph that only includes the arcs coming from "activated" recipient/donor pairs. An activated recipient/donor pair in Subgraph 2 (respectively, 3) is a pair that has at least one incoming arc in Subgraph 1 (respectively, 2). Additional arcs link each activated recipient/donor pair to the sink node S.
- Subgraph 4 only contains arcs that link each activated pair to the sink node S.

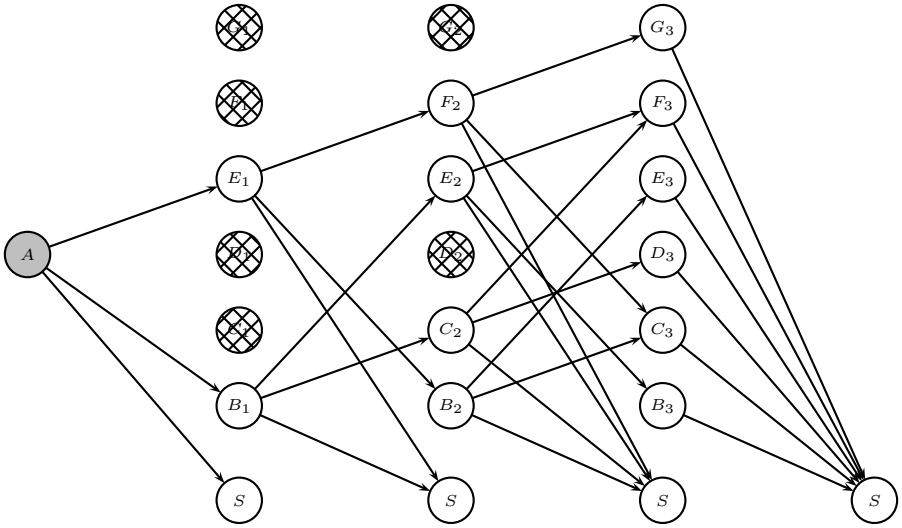
We provide in Figure 3 the graph required to model the chain structure in PICEF for Example 1.

Example 1. (resumed) *As A is compatible with the recipients B and E, nodes B_1 and E_1 are activated in the second subgraph. Since donor B is compatible with recipients C and E, and since donor E is compatible with recipients B and F, nodes B_2, C_2, E_2, and F_2 are activated in the third subgraph. An interesting observation is that every feasible chain listed in Table 1 can be reconstructed through a path in Figure 3. The opposite is not true since some paths in Figure 3 are infeasible (e.g., $A \to B_1 \to E_2 \to B_3 \to S$). Note that sink node S is duplicated in the figure for the sake of clarity.*

Let us consider the following additional notation:

- $\mathcal{C}'$ is the set of feasible cycles.
- $\mathcal{G}' = (\mathcal{V}', \mathcal{A}')$ is the graph structure required to model the chains in PICEF

Figure 3: Chain structure for Example 1 in PICEF, non-activated nodes are crosshatched



- Vertex set $\mathcal{V}'$ contains the $|\mathcal{N}|$ non-directed donors, 3 copies of the recipients, and the sink node S to model the end of the chain.
- $\mathcal{A}' = \{(u, v) : u, v \in \mathcal{V}\}$ is the set of arcs. We call $\delta^+(v)$ (respectively, $\delta^-(v)$) the subset of arcs emanating from (respectively, entering) vertex v.
- $\mathcal{A}'_v$ contains all the arcs emanating from v
  - in the first subgraph if v is an altruistic donor
  - in the last three subgraphs, if v is a recipient/donor pair.

By introducing a binary variable y_{uv} taking value 1 if arc (u, v) is selected and 0 otherwise, KEP_{z_1} can be modelled as follows with PICEF:

$$(F_1^P) \qquad \max z_1 = \sum_{c \in \mathcal{C}'} |V(c)|\, x_c + \sum_{(u,v) \in \mathcal{A}} y_{uv} \tag{10}$$

$$\text{s.t.} \sum_{c \in \mathcal{C}' : m \in V(c)} x_c + \sum_{(u,v) \in \mathcal{A}_m} y_{uv} \leq 1, \quad \forall m \in \mathcal{N} \cup \mathcal{P}, \tag{11}$$

$$\sum_{(v,w) \in \delta^+(v)} y_{vw} - \sum_{(u,v) \in \delta^-(v)} y_{uv} = \begin{cases} 1 & \text{if } v \in \mathcal{N}, \\ -|\mathcal{N}| & \text{if } v = S, \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

$$x_c \in \{0, 1\}, \qquad \forall c \in \mathcal{C}', \tag{13}$$

$$y_{uv} \in \{0, 1\}, \qquad \forall (u, v) \in \mathcal{A}. \tag{14}$$

Objective function (10) maximises the number of exchanges while constraints (11) make sure that non-directed donors do not initiate more than one chain, and that recipients appear in at most one of the selected cycles or chains. Constraints (12) are the flow conservation constraints. Objective functions z_2, z_3, and z_5 could also easily be handled by PICEF:

- If we call $\mathcal{A}_{XL}$ the set of arcs in the fourth subgraph, then the second objective function in PICEF becomes $\min z_2 = \sum_{c \in \mathcal{C}'_{XL}} x_c + \sum_{(u,v) \in \mathcal{A}_{XL}} y_{uv}$.

- If we call $\mathcal{A}_L$ the set of arcs in the third subgraph entering node S, then the third objective function in PICEF becomes $\min z_3 = \sum_{c \in \mathcal{C}'_L} x_c + \sum_{(u,v) \in \mathcal{A}_L} y_{uv}$.
- If we call $S(u,v)$ the score of arc (u,v), and if that score is set to 0 for the arcs entering node S, then the fifth objective function in PICEF becomes $\max z_5 = \sum_{c \in \mathcal{C}'} S(c)\ x_c + \sum_{(u,v) \in \mathcal{A}} S(u,v)\ y_{uv}$.

Modelling z_4 with PICEF is more challenging as there is no trivial way to count the number of cross arcs in a chain. One possibility is to generate a different graph structure for each non-directed donor, so that we can keep track of all the recipients included in a given chain, and thus, determine the number of cross arcs. However, preliminary experiments showed that such an approach is not interesting as it dramatically increases the number of variables and constraints involved in PICEF, and by extension, the time required to solve the model to optimality.

3 New algorithms for hierarchical optimisation in KEPs

In this section, we describe several algorithms aimed at optimising KEPs with hierarchical optimisation considering the UK set of objective functions $z_1, \ldots, z_5$. Since z_4 maximises the number of cross arcs in the selected cycles and chains, the cycle formulation is the most appropriate starting point. Indeed, even though PICEF was shown to have a better computational behaviour (see [20]) than the cycle formulation, it does not have a simple way to count the number of cross arcs.

3.1 Cycle/chain deactivation

The main drawback of the cycle formulation is its large number of variables: there are $O(|\mathcal{P}|^3)$ cycles when the cycle size is at most three, and $O(|\mathcal{N}||\mathcal{P}|^3)$ chains when the chain length is at most four. The resulting ILP models become intractable (because of their size) for the instances we aim to solve, where $|\mathcal{P}| \in [50, 1400]$ and $|\mathcal{N}| = f|\mathcal{P}|$ with $f \in [0.01, 0.20]$. Inspired by an innovative approach for the bin packing problem (see Delorme and Iori [18]), we propose a cycle/chain deactivation algorithm which uses the information obtained after solving the continuous relaxation of the cycle formulation to set the value of some cycles/chains to 0. In the following, we call this relaxation and its objective value $L(F_k^C)$, where $k = 1, \ldots, 5$, depending on the objective function $z_1, \ldots, z_5$ we are optimising.

After solving $L(F_1^C)$ with the complete set of cycles/chains, we obtain the linear solution $\bar{\xi}_1$. Our goal is now to find an integer solution of value $L_1 = \lfloor L(F_1^C) \rfloor$. To this aim, we gather in a set $\mathcal{C}_1$ all cycles/chains whose reduced cost[4] is less than or equal to $L_1 - L(F_1^C) - \epsilon$ (where ϵ is set to a very small value), and restrict the F_1^C model by setting the variables associated with these cycles/chains to 0. Indeed, selecting one or more cycles/chains in $\mathcal{C}_1$ would imply a solution value $\bar{z}_1$ strictly smaller than L_1. We then solve F_1^C. If no solution of value L_1 is found, we decrease L_1 by one unit, update $\mathcal{C}_1$, and iterate. Once a solution of value L_1 is found, it is guaranteed to be optimal. Note that this approach is effective because we empirically observed

[4]The reduced cost of a variable may be interpreted as the amount of penalty one would have to pay to introduce one unit of that variable into the solution, see [16]

that $\lfloor L(F_1^C) \rfloor$ is always equal or close to $\bar{z}_1$, and thus, only few iterations are required. The same approach can be used for F_4^C.

In the case of F_2^C, which is a minimisation problem, we try instead to find an integer solution of value $L_2 = \lceil L(F_2^C) \rceil$ and we gather in $\mathcal{C}_2$ all cycles/chains whose reduced cost is greater than or equal to $L_2 - L(F_2^C) + \epsilon$. The same applies to F_3^C.

Preliminary tests showed that it was not effective to apply this procedure when solving F_5^C. Indeed, as the fifth objective function deals with scores, $\lfloor L(F_5^C) \rfloor$ can be far away from $\bar{z}_5$. In addition, only few cycles/chains are still activated after solving F_4^C, so the interest of reducing even further the number of activated cycles/chains is limited. An overview of the overall cycle deactivation algorithm is presented in Algorithm 1, and the outputs it obtains with an ILP solver when applied to Example 1 are presented in Table 2. The first four columns still give an identifier for each cycle/chain c, its length, the number of cross arcs in c, and the set of vertices that compose the cycle/chain. Columns $L(F_k^C)$ contain the value (in sub-column "V") and the reduced cost (in sub-column "RC") taken by x_c after solving the continuous relaxation of the cycle model for objective function k where $k = 1, 2, 3, 4$. Columns L_k contain the value (in sub-column "V") taken by x_c after solving the cycle model for objective function k where $k = 1, 2, 3, 4, 5$. An "x" indicates that the cycle/chain was deactivated when the model was solved.

Algorithm 1 `cycle/chain deactivation algorithm`

1: $L_1 = \lfloor L(F_1^C) \rfloor$
2: Deactivate cycles/chains $\mathcal{C}_1$ with reduced cost $r_1 \leq L_1 - L(F_1^C) - \epsilon$ and let $\bar{z}_1 := F_1^C$
3: **if** $\bar{z}_1 \neq L_1$ **then** $L_1 = L_1 - 1$, reactivate cycles/chains $\mathcal{C}_1$, and go back to step 2
4: $L_2 = \lceil L(F_2^C) \rceil$
5: Deactivate cycles/chains $\mathcal{C}_2$ with reduced cost $r_2 \geq L_2 - L(F_2^C) + \epsilon$ and let $\bar{z}_2 := F_2^C$
6: **if** $\bar{z}_2 \neq L_2$ **then** $L_2 = L_2 + 1$, reactivate cycles/chains $\mathcal{C}_2$, and go back to step 5
7: $L_3 = \lceil L(F_3^C) \rceil$
8: Deactivate cycles/chains $\mathcal{C}_3$ with reduced cost $r_3 \geq L_3 - L(F_3^C) + \epsilon$ and let $\bar{z}_3 := F_3^C$
9: **if** $\bar{z}_3 \neq L_3$ **then** $L_3 = L_3 + 1$, reactivate cycles/chains $\mathcal{C}_3$, and go back to step 8
10: $L_4 = \lfloor L(F_4^C) \rfloor$
11: Deactivate cycles/chains $\mathcal{C}_4$ with reduced cost $r_4 \leq L_4 - L(F_4^C) - \epsilon$ and let $\bar{z}_4 := F_4^C$
12: **if** $\bar{z}_4 \neq L_4$ **then** $L_4 = L_4 - 1$, reactivate cycles/chains $\mathcal{C}_4$, and go back to step 11
13: $\bar{z}_5 := F_5^C$

Example 1. (resumed) *Algorithm 1 determines that* $L_1 = \lfloor L(F_1^C) \rfloor = 6$ *and thus, puts chains 2-7 and 9-13 in* $\mathcal{C}_1$ *as they all have a reduced cost equal to -1. In other words, every feasible continuous solution containing one of any cycle/chain in* $\mathcal{C}_1$ *must have* $\bar{z}_1 \leq 5$. *The same holds for any feasible integer solution. With integrality constraints, the best solution obtained is* $\bar{z}_1 = 4$, *which is strictly smaller than* L_1. *Thus, the algorithm decreases* L_1 *to 5, reactivates all the chains, and finds an integer solution of value 5 by selecting cycle 1 and chain 4.*

When it switches to z_2, *the algorithm finds that* $L_2 = \lceil L(F_2^C) \rceil = 0$ *and puts chains 5, 7, 8, 11, 13, and 14 in* $\mathcal{C}_2$. *With integrality constraints, the best solution obtained is* $\bar{z}_2 = 0$, *which matches the bound. The chains in* $\mathcal{C}_2$ *are deactivated for the rest of the algorithm.*

When it arrives at z_3, *the algorithm finds that* $L_3 = \lceil L(F_3^C) \rceil = 0$ *and puts chains 4, 6, 10, and 12 in* $\mathcal{C}_3$. *With integrality constraints, the best solution obtained is* $\bar{z}_3 = 0$, *which matches the bound. The chains in* $\mathcal{C}_3$ *are deactivated for the rest of the algorithm.*

Table 2: Results of the cycle/chain deactivation obtained by an ILP solver for Example 1

id	len.	# c. a.	comp.	z_1				z_2			z_3			z_4			z_5
				$L(F_1^C)=6$		$L_1=6$	$L_1=5$	$L(F_2^C)=0$		$L_2=0$	$L(F_3^C)=0$		$L_3=0$	$L(F_4^C)=0$		$L_4=0$	
				V	RC	V	V	V	RC	V	V	RC	V	V	RC	V	V
0	2	0	$[B,E]$	0.5	0	1	0	0.52	0	1	0.69	0	1	0.68	0	1	1
1	2	0	$[C,F]$	0.5	0	1	1	0.83	0	1	1	0	1	1	0.14	1	1
2	1	0	A	0	-1	x	0	0.19	0	1	0.37	0	1	0.36	0	1	1
3	2	0	$A \to B$	0	-1	x	0	0.16	0	0	0.31	0	0	0.32	0	0	0
4	3	2	$A \to B \to E$	0	-1	x	1	0.15	0	0	0	1	x	x	x	x	x
5	4	2	$A \to B \to E \to F$	0	-1	x	0	0	1	x	x	x	x	x	x	x	x
6	3	0	$A \to B \to C$	0	-1	x	0	0.17	0	0	0	1	x	x	x	x	x
7	4	1	$A \to B \to C \to F$	0	-1	x	0	0	1	x	x	x	x	x	x	x	x
8	4	0	$A \to B \to C \to D$	0.5	0	0	0	0	0.58	x	x	x	x	x	x	x	x
9	2	0	$A \to E$	0	-1	x	0	0.16	0	0	0.31	0	0	0.32	0	0	0
10	3	2	$A \to E \to B$	0	-1	x	0	0	0	0	0	1	x	x	x	x	x
11	4	2	$A \to E \to B \to C$	0	-1	x	0	0	1	x	x	x	x	x	x	x	x
12	3	0	$A \to E \to F$	0	-1	x	0	0.17	0	0	0	1	x	x	x	x	x
13	4	1	$A \to E \to F \to C$	0	-1	x	0	0	1	x	x	x	x	x	x	x	x
14	4	0	$A \to E \to F \to G$	0.5	0	0	0	0	0.58	x	x	x	x	x	x	x	x

When processing the fourth objective z_4, the algorithm finds that $L_4 = \lfloor L(F_1^C) \rfloor = 0$, but does not find any chain or cycle to put in $\mathcal{C}_4$ as all the reduced costs are equal to 0. With integrality constraints, the best solution obtained is $\bar{z}_4 = 0$, which matches the bound.

When it reaches z_5, only 5 cycles/chains out of the 15 are still activated. The algorithm finds a solution with score 4 that selects cycles $[C,F]$, $[B,E]$, and the chain that contains the non-directed donor A.

3.2 Diving algorithm

After preliminary tests on Algorithm 1, we observed that steps 2 and 5 took most of the computational effort. This is not surprising as only few cycles/chains are deactivated at these steps, and solving exactly an ILP with many variables can be time-consuming.

However, the only ILP solution that is relevant for the problem is the one solved at step 13 as it is the one that gives the set of cycles and chains that should be selected to optimise the five objective functions. The other four ILP models only give binary indications about whether or not a solution of value L_k exists, where $k = 1, 2, 3, 4$.

The idea of the diving algorithm is to first make the assumption that a solution of value L_k exists (where $k = 1, 2$) and thus, to skip the ILP models of steps 2 and 5 necessary to obtain $\bar{z}_1$ and $\bar{z}_2$. The diving algorithm may backtrack and correct that assumption in case the model of step 8 is infeasible. Preliminary tests showed that it was not expedient to extend the assumption to L_3 as it involved significantly more backtracks. An overview of the diving algorithm is presented in Algorithm 2.

Algorithm 2 starts by solving $L(F_1^C)$, $L(F_2^C)$, and $L(F_3^C)$, and deactivates the corresponding sets of cycles/chains $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ (steps 2-6, 15-17). It then solves F_3^C and tries to find an integer solution where the three first objectives functions are equal to L_1, L_2, and L_3 (step 18).

Algorithm 2 `Diving algorithm`

1: $\Lambda_2 = 0, \Lambda_3 = 0$ ▷ Keep track of the number of failures
2: $L_1 = \lfloor L(F_1^C) \rfloor$ ▷ Assumption 1: $z_1 = L_1$
3: Deactivate cycles/chains $\mathcal{C}_1$ with reduced cost $r_1 \leq L_1 - L(F_1^C) - \epsilon$
4: $L_2 = \lceil L(F_2^C) \rceil$ ▷ Assumption 2: $z_2 = L_2$
5: **if** $\Lambda_2 < \bar{\Lambda}_2$ **then** ▷ If we trust Assumption 1
6: Deactivate cycles/chains $\mathcal{C}_2$ with reduced cost $r_2 \geq L_2 - L(F_2^C) + \epsilon$
7: **else** ▷ If we doubt Assumption 1
8: Get $\bar{z}_2 = F_2^C$ ▷ Solve F_2^C exactly, with no $\mathcal{C}_2$ cycle/chain deactivation
9: **if** F_2^C is infeasible **then** ▷ Assumption 1 was wrong
10: $\Lambda_2 = 0, L_1 = L_1 - 1$, reactivate cycles/chains $\mathcal{C}_1$, and go back to step 3 ▷ update Ass. 1
11: **else** ▷ Assumption 1 was right
12: Deactivate cycles/chains $\mathcal{C}_2$ with reduced cost $r_2 \geq \bar{z}_2 - L(F_2^C) + \epsilon$
13: **end if**
14: **end if**
15: $L_3 = \lceil L(F_3^C) \rceil$
16: **if** $\Lambda_3 < \bar{\Lambda}_3$ **then** ▷ If we trust Assumption 2
17: Deactivate cycles/chains $\mathcal{C}_3$ with reduced cost $r_3 \geq L_3 - L(F_3^C) + \epsilon$
18: Get $\bar{z}_3 = F_3^C$
19: **if** $\bar{z}_3 \neq L_3$ **then**
20: $L_3 = L_3 + 1$, $\Lambda_3 = \Lambda_3 + 1$, reactivate cycles/chains $\mathcal{C}_3$, and go back to step 16
21: **end if**
22: **else** ▷ If we doubt Assumption 2
23: get $\bar{z}_3 = F_3^C$ ▷ Solve F_3^C exactly, without $\mathcal{C}_3$ cycle/chain deactivation
24: **if** F_3^C is infeasible **then** ▷ Assumption 2 was wrong
25: $\Lambda_3 = 0, \Lambda_2 = \Lambda_2 + 1, L_2 = L_2 + 1$, reactivate cycles/chains $\mathcal{C}_2$, and go back to step 5
26: **else** ▷ Assumption 2 was right
27: Deactivate cycles/chains $\mathcal{C}_3$ with reduced cost $r_3 \geq \bar{z}_3 - L(F_3^C) + \epsilon$
28: **end if**
29: **end if**
30: $L_4 = \lfloor L(F_4^C) \rfloor$ ▷ Step 4 is unchanged
31: Deactivate cycles/chains with reduced cost $r_4 \leq L_4 - L(F_4^C) - \epsilon$ and get $\bar{z}_4 = F_4^C$
32: **if** $\bar{z}_4 \neq L_4$ **then** $L_4 = L_4 - 1$ and go back to step 31
33: get $\bar{z}_5 = F_5^C$

If it succeeds, it then solves the fourth objective function as in the cycle/chain deactivation algorithm (steps 30-32) and the fifth as in the cycle formulation (step 33). If it fails, it records the failure in the variable Λ_3, increases L_3 (step 20), updates $\mathcal{C}_3$ and tries again (steps 17-18).

Once it has reached a given number of failures $\bar{\Lambda}_3$, the algorithm solves F_3^C without deactivating any cycle/chain in $\mathcal{C}_3$ and without any consideration on the bound L_3 (step 23). If F_3^C is feasible, the algorithm re-deactivates the appropriate cycles/chains in $\mathcal{C}_3$ (step 27) and moves on to the fourth objective function. If F_3^C is infeasible, the algorithm records the failure in the variable Λ_2, increases L_2, removes any information about $\mathcal{C}_3$ or L_3 (step 25), updates $\mathcal{C}_2$, (step 6) and solves $L(F_3^C)$ again (step 15).

Once it has reached a given number of failures $\bar{\Lambda}_2$, the algorithm solves F_2^C without deactivating any cycle/chain in $\mathcal{C}_2$ and without any consideration on the bound L_2 (step 8). If F_2^C is feasible, the algorithm re-deactivates the appropriate cycles/chains in $\mathcal{C}_2$ (step 12) and moves on to the third objective function. If F_2^C is infeasible, the algorithm decreases L_1, removes any information about $\mathcal{C}_2$ or L_2 (step 10), updates $\mathcal{C}_1$ (step 3), and solves $L(F_2^C)$ again (step 4).

The performance of Algorithm 2 highly depends on the allowed number of failures $\bar{\Lambda}_2$ and $\bar{\Lambda}_3$. If the parameters have values that are too small, the algorithm may have to go to steps 8 and 23 and solve ILP models with no cycle/chain deactivation while it was not necessary. If the parameters have values that are too large, the algorithm may have to loop a significant amount of time in steps 17-21 and steps 6, 15-29 before realising that Assumptions 1 and 2 were wrong. The outputs obtained by Algorithm 2 with an ILP solver when applied to Example 1 are presented in Table 3.

Example 1. (resumed)

Table 3: Results of the diving algorithm obtained by an ILP solver for Example 1, $\bar{\Lambda}_2 = \bar{\Lambda}_3 = 1$

id	len.	# c.	# a.	comp.	$L(F_1^C)$		$L(F_2^C)$		$L(F_3^C)$		F_3^C	F_3^C	F_2^C	$L(F_2^C)$		$L(F_3^C)$		F_3^C	$L(F_4^C)$		F_4^C	F_5^C
					V	RC	V	RC	V	RC	V	V	V	V	RC	V	RC	V	V	RC	V	V
0	2	0		$\{B, E\}$	0.5	0	0.5	0	0.5	0				0.52	0	0.69	0	1	0.68	0	1	1
1	2	0		$\{C, F\}$	0.5	0	0.5	0	0.5	0				0.83	0	1	0	1	1	0.14	1	1
2	1	0		A	0	-1	x	x	x	x	I	I	I	0.19	0	0.37	0	1	0.36	0	1	1
3	2	0		$A \to B$	0	-1	x	x	x	x	N	N	N	0.16	0	0.31	0	0	0.32	0	0	0
4	3	2		$A \to B \to E$	0	-1	x	x	x	x	F	F	F	0.15	0	0	1	x	x	x	x	x
5	4	2		$A \to B \to E \to F$	0	-1	x	x	x	x	E	E	E	0	1	x	x	x	x	x	x	x
6	3	0		$A \to B \to C$	0	-1	x	x	x	x	A	A	A	0.17	0	0	1	x	x	x	x	x
7	4	1		$A \to B \to C \to F$	0	-1	x	x	x	x	S	S	S	0	1	x	x	x	x	x	x	x
8	4	0		$A \to B \to C \to D$	0.5	0	0.5	0	0.5	0	I	I	I	0	0.58	x	x	x	x	x	x	x
9	2	0		$A \to E$	0	-1	x	x	x	x	B	B	B	0.16	0	0.31	0	0	0.32	0	0	0
10	3	2		$A \to E \to B$	0	-1	x	x	x	x	L	L	L	0	0	0	1	x	x	x	x	x
11	4	2		$A \to E \to B \to C$	0	-1	x	x	x	x	E	E	E	0	1	x	x	x	x	x	x	x
12	3	0		$A \to E \to F$	0	-1	x	x	x	x				0.17	0	0	1	x	x	x	x	x
13	4	1		$A \to E \to F \to C$	0	-1	x	x	x	x				0	1	x	x	x	x	x	x	x
14	4	0		$A \to E \to F \to G$	0.5	0	0.5	0	0.5	0				0	0.58	x	x	x	x	x	x	x
				Λ_2	0		0		0		0	1	0	0		0		0	0		0	0
				Λ_3	0		0		0		1	0	0	0		0		0	0		0	0
Variables				L_1	6		6		6		6	6	5	5		5		5	5		5	5
				L_2	$\emptyset$		1		1		1	$\emptyset$	$\emptyset$	0		0		0	0		0	0
				L_3	$\emptyset$		$\emptyset$		0		$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$		0		0	0		0	0
				L_4	$\emptyset$		$\emptyset$		$\emptyset$		$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$		$\emptyset$		$\emptyset$	0		0	0

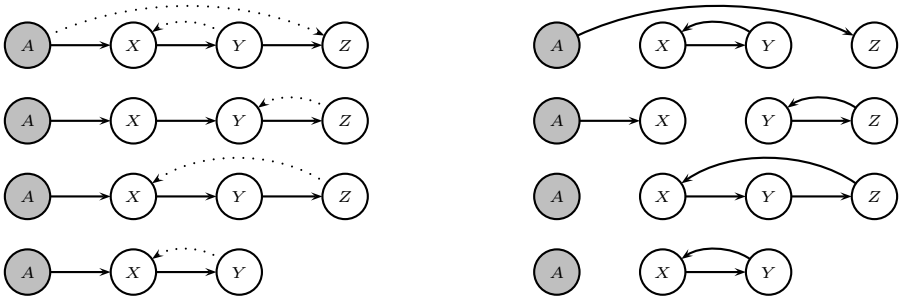
Algorithm 2 determines that $L_1 = \lfloor L(F_1^C) \rfloor = 6$ and thus, puts chains 2-7 and 9-13 in $\mathcal{C}_1$ as they all have a reduced cost equal to -1. It then performs $L_2 = \lceil L(F_2^C) \rceil = 1$, but does not find any chain or cycle to put in $\mathcal{C}_2$ as all the reduced costs are equal to 0. It continues with $L_3 = \lceil L(F_3^C) \rceil = 0$ and does not find any chain or cycle to put in $\mathcal{C}_3$ either. With integrality constraints, the algorithm does not find any feasible solution to F_3^C, and thus, increments variable Λ_3 which is now equal to 1. As 1 is also the limit set by $\bar{\Lambda}_3$, the algorithm removes any assumption it has on L_3, empties $\mathcal{C}_3$, and solves again F_3^C. Since it was not able to find any feasible solution, the algorithm backtracks to the second objective function, increments Λ_2, and resets Λ_3. As Λ_2 is now equal to the limit $\bar{\Lambda}_2$, it removes any assumption it has on L_2, empties $\mathcal{C}_2$, and solves again F_2^C. Once more, no feasible solution could be found, so the algorithm backtracks to the first objective function, sets L_1 to 5, empties $\mathcal{C}_1$, resets Λ_2, and solves again $L_2 = \lceil L(F_2^C) \rceil$, which is now equal to 0. Because of their reduced costs, chains 5, 7, 8, 11, 13, and 14 are put in $\mathcal{C}_2$. The algorithm continues with $L_3 = \lceil L(F_3^C) \rceil = 0$ and and puts chains

4, 6, 10, and 12 in $\mathcal{C}_3$. With integrality constraints, the best solution obtained is $\bar{z}_3 = 0$, which matches the bound. The chains in $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ are deactivated for the rest of the algorithm. Algorithm 2 behaves similarly to Algorithm 1 for the two last objective functions z_4 and z_5, and it finds the same optimal solution containing cycles 0 and 1, and chain 2.

3.3 Dominated chains

Let us consider a chain c. If there exists a combination s of one chain and one cycle involving the same pairs and non-directed donor as the ones involved in c that improves objective function z_2 (respectively, z_3) while preserving z_1 (respectively, z_1 and z_2) with respect to c, then we call c a *dominated* chain and s a *dominating* set. We detail in Figure 4 four generic cases of dominated chains (on the left side) and their respective dominating set (on the right side). For example,

Figure 4: Dominated chains for the UKLKSS and their dominating sets



the chain of length three $A \to X \to Y$ with a cross arc between Y and X contributes to 3 units in z_1, 0 unit in z_2, and 1 unit in z_3. Leaving the altruistic donor A alone and selecting the cycle of size two $[X, Y]$ instead still contributes to 3 units in z_1 and 0 unit in z_2, but now counts as 0 unit in z_3, which is an improvement with respect to selecting the chain $A \to X \to Y$. Similarly, the chain of length four $A \to X \to Y \to Z$ with a cross arc between Z and Y contributes to 4 units in z_1, 1 unit in z_2, and 0 unit in z_3. Using instead chain $A \to X$ and cycle $[Y, Z]$ still contributes to 4 units in z_1, but now counts as 0 unit in z_2 and 0 unit in z_3, which is an improvement with respect to selecting the chain $A \to X \to Y \to Z$. The dominated chains of Example 1 are outlined in Table 4.

Note that the concept of dominated chains could also be extended to take the cycles into account in order to detect some sets of dominating cycles. For example, three cycles of length two $[U, V]$, $[W, X]$, and $[Y, Z]$ should always be selected over two cycles of length three $[U, V, W]$ and $[X, Y, Z]$. However, the detection of such dominated sets of cycles is not trivial, and while removing a dominated chain fixes a variable to 0, removing a dominated set of cycles requires an additional constraint, which goes against our model size reduction paradigm.

3.4 Hybrid algorithm

So far, we have used the cycle formulation because objective function z_4 maximises the number of cross arcs in the selected cycles and chains and no easy modification of PICEF would allow us to count the number of cross arcs in the chain structure. In the hybrid algorithm, we propose the use of PICEF for the three first objective functions z_1, z_2, and z_3, and transition to the cycle

Table 4: Dominated chains identification for Example 1

id	len.	# b. a.	cycle members	is dominated	dominating set
0	2	0	$[B, E]$	n/a	n/a
1	2	0	$[C, F]$	n/a	n/a
2	1	0	A	no	-
3	2	0	$A \to B$	no	-
4	3	2	$A \to B \to E$	yes	{0,2}
5	4	2	$A \to B \to E \to F$	no	-
6	3	0	$A \to B \to C$	no	-
7	4	1	$A \to B \to C \to F$	yes	{1,3}
8	4	0	$A \to B \to C \to D$	no	-
9	2	0	$A \to E$	no	-
10	3	2	$A \to E \to B$	yes	{0,2}
11	4	2	$A \to E \to B \to C$	no	-
12	3	0	$A \to E \to F$	no	-
13	4	1	$A \to E \to F \to C$	yes	{1,9}
14	4	0	$A \to E \to F \to G$	no	-

formulation for the two last objective functions z_4 (with cycle/chain deactivation) and z_5. Note that returning to PICEF when optimising z_5 is not an option since we now have a constraint on the number of cross arcs that needs to be in the solution.

The cycle/chain deactivation algorithm can be extended to PICEF and is now called "cycle/arc deactivation algorithm", as chains are represented in a graph structure in PICEF. The diving algorithm can also be used with PICEF following Algorithm 2 (until step 29). Forbidding dominated chains in PICEF is more challenging than it is in the cycle formulation. Indeed, removing a dominated chain in the cycle formulation simply sets its corresponding decision variable to 0. In PICEF, we need to add a constraint for each dominated chain to force the sum of the decision variables associated with each of its arcs to be less than or equal to the length of the chain minus one. Let us recall that since each chain is uniquely defined by a path from a non-directed donor to the sink node S, forbidding chain $A \to B_1 \to E_2 \to S$ does not forbid $A \to B_1 \to E_2 \to F_3 \to S$, for example. Preliminary tests indicated that adding such constraints dramatically increases the time to solve PICEF, so dominated chains were not forbidden in our PICEF implementation.

Regarding the transition between z_3 and z_4, we need to do a complete graph exploration of the PICEF chain structure, and transform every feasible path in PICEF into a chain in the cycle formulation, provided that the resulting chain is not dominated. The outputs obtained by the extension of Algorithm 2 to PICEF (until line 29) with an ILP solver when applied to Example 1 are detailed in Table 5.

Example 1. (resumed)

We omit the full description of the algorithm outputs, since they are similar to those presented in Table 3. We simply observe that every cycle/chain that is not deactivated in Table 3 at a given step can be reconstructed by some arcs that are not deactivated at that same step in Table 5. Once the third objective function is solved, we do a complete graph exploration of the chain structure to transform every feasible path in PICEF into a chain in the cycle formulation. In

Table 5: Results of the extension of the diving algorithm to PICEF obtained by an ILP solver for Example 1, $\bar{\Lambda}_2 = \bar{\Lambda}_3 = 1$

id	len.	comp.	$L(F_1^C)$		$L(F_2^C)$		$L(F_3^C)$		F_3^C	F_3^C	F_2^C	$L(F_2^C)$		$L(F_3^C)$		F_3^C
			V	RC	V	RC	V	RC	V	V	V	V	RC	V	RC	V
0	2	$[B, E]$	0.5	0	0.5	0	0.5	0				0.53	0	0.69	0	1
1	2	$[C, F]$	0.5	0	0.5	0	0.5	0				0.81	0	1	0	1
2	1	$A \to B_1$	0.5	0	0.5	0	0.5	0				0.34	0	0.31	0	1
3	1	$A \to E_1$	0.5	0	0.5	0	0.5	0				0.47	0	0.31	0	0
4	1	$A \to S$	0	-1	x	x	x	x				0.19	0	0.37	0	0
5	1	$B_1 \to E_2$	0	0	0	0	0	0				0	0	0	1	x
6	1	$B_1 \to C_2$	0.5	0	0.5	0	0.5	0				0.19	0	0	1	x
7	1	$B_1 \to S$	0	-1	x	x	x	x				0.15	0	0.31	0	0
8	1	$E_1 \to B_2$	0	0	0	0	0	0				0.13	0	0	1	x
9	1	$E_1 \to F_2$	0.5	0	0.5	0	0.5	0	I	I	I	0.19	0	0	1	x
10	1	$E_1 \to S$	0	-1	x	x	x	x	N	N	N	0.15	0	0.31	0	0
11	1	$B_2 \to E_3$	0	-1	x	x	x	x	F	F	F	0	1	x	x	x
12	1	$B_2 \to C_3$	0	-1	x	x	x	x	E	E	E	0	1	x	x	x
13	1	$B_2 \to S$	0	-1	x	x	x	x	A	A	A	0.13	0	0	0	0
14	1	$E_2 \to B_3$	0	-1	x	x	x	x	S	S	S	0	1	x	x	x
15	1	$E_2 \to F_3$	0	-1	x	x	x	x	I	I	I	0	1	x	x	x
16	1	$E_2 \to S$	0	-1	x	x	x	x	B	B	B	0	0	0	0	0
17	1	$C_2 \to F_3$	0	-1	x	x	x	x	L	L	L	0	1	x	x	x
18	1	$C_2 \to D_3$	0.5	0	0.5	0	0.5	0	E	E	E	0	0.7	x	x	x
19	1	$C_2 \to S$	0	-1	x	x	x	x				0.19	0	0	0	0
20	1	$F_2 \to C_3$	0	-1	x	x	x	x				0	1	x	x	x
21	1	$F_2 \to G_3$	0.5	0	0.5	0	0.5	0				0	0.7	x	x	x
22	1	$F_2 \to S$	0	-1	x	x	x	x				0.19	0	0	0	0
23	1	$B_3 \to S$	0	0	0	0	0	0				0	0	0	0	0
24	1	$E_3 \to S$	0	0	0	0	0	0				0	0	0	0	0
25	1	$C_3 \to S$	0	0	0	0	0	0				0	0	0	0	0
26	1	$F_3 \to S$	0	0	0	0	0	0				0	0	0	0	0
27	1	$D_3 \to S$	0.5	0	0.5	0	0.5	0				0	0	0	0	0
28	1	$G_3 \to S$	0.5	0	0.5	0	0.5	0				0	0	0	0	0
		Λ_2	0		0		0		0	1		0	0		0	0
		Λ_3	0		0		0		1	0		0	0		0	0
	Variables	L_1	6		6		6		6	6		5	5		5	5
		L_2	$\emptyset$		1		1		1	$\emptyset$		$\emptyset$	0		0	0
		L_3	$\emptyset$		$\emptyset$		0		$\emptyset$	$\emptyset$		$\emptyset$	$\emptyset$		0	0

the example, 15 arcs are still activated:

- $A \to B_1$, $A \to E_1$, $A \to S$ *from subgraph 1*
- $B_1 \to S$, $E_1 \to S$ *from subgraph 2*
- $B_2 \to S$, $E_2 \to S$, $C_2 \to S$, $F_2 \to S$ *from subgraph 3*
- $B_3 \to S$, $E_3 \to S$, $C_3 \to S$, $F_3 \to S$, $D_3 \to S$, $G_3 \to S$ *from subgraph 4*

These arcs allow us to reconstruct chains $A \to S$, $A \to B_1 \to S$, and $A \to E_1 \to S$ in the cycle formulation. These were also the chains that were activated at the fourth objective function in Table 3.

4 Experimental results

We report in this section the outcome of extensive computational experiments aimed at testing the effectiveness of our new algorithms for the UKLKSS. We also show that our methods can be adapted to take into account the objective functions of other countries that employ hierarchical optimisation (see [8]) as illustrated by the application of our approach to the Spanish and the Dutch KEPs.

All our algorithms were coded in `C++` and can be downloaded from `https://github.com/mdelorme2/Hierarchical_Optimisation_Kidney_Exchange_Programmes_Codes`. The experiments were run on an Intel Xeon E5-2680W v3, 2.50GHz with 192GB of memory, running under Scientific Linux 7.5, and Gurobi 7.5.2 was used to solve the ILP models. Parameter ϵ was set to 0.001, and preliminary tests indicated that 1 was a good value for parameters $\bar{\Lambda}_2$ and $\bar{\Lambda}_3$. Each instance was run using a single core and no time limit was imposed, unless specified otherwise. We used the following parameters for Gurobi:

- `Method = 2`, meaning that the barrier algorithm was used to solve both the LP models and the root nodes of the ILP models. Preliminary tests showed that using the barrier algorithm was faster than letting Gurobi choose the algorithm (`Method = -1`).
- `MIPGap = 0` for the ILP models, meaning that the solver terminates with an optimal solution if the gap between the lower bound and the upper bound is equal to 0.
- `Crossover = 0` for the LP models, meaning that the solver does not try to transform the interior solution produced by the barrier algorithm into a basic solution. Our approaches only need the LP optimal values and the reduced costs associated with each variable, not a basic solution. Disabling crossover saved a significant amount of time in our tests.

4.1 Instance generation

In order to generate instances similar to real-world cases, we used the data generator written by Trimble [23]. The generator is derived from the work of Saidman et al. [37] and was used previously in the literature by Klimentova, Pedroso, and Viana [26] and Blum et al. [10]. The generator can be used to replicate any KEP pool provided that the user has an accurate estimation of the following key population parameters:

- donor blood types: proportion of donors in each blood group {O, A, B, AB};
- recipient blood types: proportion of recipients in each blood group {O, A, B, AB};
- donors per recipient: proportion of recipients who have 1, 2, 3, or 4 donors; and
- calculated reaction frequency (or cPRA): the proportion of donors who would not be tissue-type compatible with a given recipient.

We used a large set of data provided by the UKLKSS to create instances with a similar distribution to those solved in their quarterly runs. Note that the above-mentioned generator also used "Additional fields for compatibility with Saidman generator" that we left untouched since the information for these fields were not available in our data, and we added an inner routine to

attribute a score to each compatibility following the score distribution observed in the UKLKSS data.

In addition to the above key population parameters, the generator also requires some information about the number of recipients and the proportion of altruistic donors. In order to have a complete overview of the performance of our algorithms and their scalability, we generated instances with a large range of number of recipients ($|\mathcal{P}| \in \{50, 100, 200, 400, 600, 800, 1000, 1200, 1400\}$) and a few different proportions of non-directed donors ($f \in \{0.01, 0.05, 0.10, 0.15, 0.20\}$ where $|\mathcal{N}| = f|\mathcal{P}|$). For each pair $(|\mathcal{P}|, f)$, we generated 30 instances resulting in $9 \times 5 \times 30 = 1350$ instances in total. All the instances can be downloaded from the online repository `http://researchdata.gla.ac.uk/id/eprint/1016`.

4.2 Randomly generated instances with the UKLKSS objectives

We first tested each of our new approaches on a subset of the randomly generated instances and we compared their results with those obtained by the cycle formulation, which is currently in use by the UKLKSS. The subset contains 450 instances with a number of recipients $|\mathcal{P}| \in \{50, 100, 200, 400, 600\}$ and a proportion of non-directed donors $f \in \{0.01, 0.05, 0.10\}$. These values were chosen so that each algorithm could solve all the instances to optimality in a reasonable time. We then tested the hybrid algorithm on the remaining 900 instances to show its scalability. When not specified otherwise, we display the results by number of recipients (i.e., instances with different values of f are merged).

Table 6 contains the results of the five algorithms for instances with 50 and 100 recipients, which are comparable in size to medium-sized European KEPs such as the Spanish (around 110 recipients per run [8]) or the Dutch (around 40 recipients per run) KEPs. The "Algorithm" column specifies the algorithm used, the attribute "- DC" indicates that dominated chains were removed from the model (for all the objective functions for the diving algorithm and for z_4 and z_5 for the hybrid algorithm), columns "TT" give the average total time required by the algorithm to solve an instance, and columns "Tk" indicate the time required to optimise objective function z_k, where $k = 1, \ldots, 5$. The last line of the table indicates the optimal value of each objective function $z_1, \ldots, z_5$ averaged over the 90 instances. As no time limit was imposed, all the algorithms solved the instances to optimality. Note that (i) "TT" also incorporates the time to generate the variables of the model, and thus, might be few seconds away from the sum of the "Tx" for large size instances, (ii) the time spent by the hybrid algorithm to perform the transition between PICEF and the cycle formulation is included in T4, and (iii) for algorithms including a diving component, the time displayed in T2 only incorporates the time spent solving z_2 once the optimal value of z_1 was found. The time spent solving z_2 for wrong values of z_1 is incorporated in T1. The same comments apply for the time displayed in T3.

The table shows that instances with up to 100 recipients can be solved to optimality with the cycle formulation in less than one second on average. Even though our new approaches are faster (on average 0.06s for the hybrid algorithm vs 0.81s for the cycle formulation for instances with 100 recipients), the time saved is not enough to motivate a switch from the cycle formulation for such instances. Table 7 contains the same information for instances with 200 and 400 recipients, which is of comparable size to a large European KEP such as the UKLKSS

Table 6: Average time (in seconds) required by each of the proposed algorithms to solve instances with 50 and 100 recipients

Algorithm	50 recipients, $f = \{0.01, 0.05, 0.10\}$						100 recipients, $f = \{0.01, 0.05, 0.10\}$					
	TT	T1	T2	T3	T4	T5	**TT**	T1	T2	T3	T4	T5
Cycle	**0.07**	0.01	0.02	0.02	0.01	0.02	**0.81**	0.09	0.15	0.15	0.17	0.25
Cycle/chain deactivation	**0.04**	0.01	0.01	0.01	0.01	0.01	**0.11**	0.06	0.02	0.01	0.01	0.01
Diving algorithm	**0.03**	0.00	0.00	0.01	0.01	0.01	**0.08**	0.04	0.01	0.01	0.01	0.01
Diving algorithm - DC	**0.03**	0.00	0.00	0.01	0.01	0.01	**0.06**	0.02	0.01	0.01	0.01	0.01
Hybrid algorithm - DC	**0.03**	0.00	0.00	0.01	0.01	0.00	**0.06**	0.01	0.01	0.01	0.02	0.01
Optimal values	-	18.63	0.94	2.94	3.77	1407	-	47.19	2.39	8.73	9.6	3852

(around 300 recipients per matching run).

Table 7: Average time (in seconds) required by each of the proposed algorithms to solve instances with 200 and 400 recipients

Algorithm	200 recipients (90 instances)						400 recipients (90 instances)					
	TT	T1	T2	T3	T4	T5	**TT**	T1	T2	T3	T4	T5
Cycle	**25.58**	2.54	4.82	5.34	4.35	8.49	**866.88**	60.39	191.05	227.79	125.64	261.44
Cycle/chain deactivation	**1.29**	1.08	0.11	0.03	0.02	0.01	**34.83**	31.37	2.40	0.24	0.16	0.07
Diving algorithm	**1.20**	1.00	0.09	0.03	0.02	0.02	**32.95**	28.42	2.64	0.29	0.14	0.08
Diving algorithm - DC	**0.44**	0.31	0.04	0.03	0.02	0.01	**9.86**	7.79	1.05	0.25	0.12	0.07
Hybrid algorithm - DC	**0.20**	0.06	0.02	0.03	0.06	0.02	**1.18**	0.45	0.14	0.17	0.28	0.09
Optimal values	-	122.94	5.48	26.06	25.06	10 235	-	310.49	12.82	72.46	58.88	25 686

The table shows a number of interesting facts:

- While the cycle formulation can solve to optimality instances with 200 recipients in 26 seconds on average, it takes almost 15 minutes on average to solve instances with 400 recipients.
- The cycle formulation spends a significant amount of time solving each objective function, with z_3 and z_5 being the longest to solve.
- All our new approaches spend the majority of their time solving z_1, the other objective functions are solved almost instantly.
- Removing the dominated chains has a significant impact on the model performances: it takes 32.95s on average to solve an instance with 400 recipients by the diving algorithm if we allow dominated chains while it only takes 9.86s if we remove them. As additional information, we also mention that on average, for instances with 400 recipients, there are 12 469 dominated chains of length 3 (out of 46 021) and 970 050 dominated chains of length 4 (out of 1 468 336).
- The hybrid algorithm is by far the fastest among all the tested algorithms and obtain outstanding results. It solves instances with 400 recipients three orders of magnitude faster, on average, than the cycle formulation. Its PICEF component saves a significant

amount of time while solving z_1, z_2, and z_3 with respect to the other methods. The time it spends during the transition between PICEF and the cycle formulation is noticeable (+0.16s on average to solve z_4 for instances with 400 recipients with respect to the diving algorithm without dominated chains), but irrelevant when compared to the time earned solving the first three objective functions.

Table 8 gives detailed information about the models for instances with 200 and 400 recipients. In particular, it gives the number of variables, constraints, and non-zero elements in the last ILP model (i.e., when solving z_5) in columns "nb. var.", "nb. cons.", and "nb. nzs", respectively. It also indicates in columns "nb Fk" the average number of times the bound L_k ($k = 1, \ldots, 4$) had to be updated, meaning that a backtracking step was necessary. We remind the reader that there is no backtracking in the cycle formulation.

Table 8: Detailed information on the algorithms for instances with 200 and 400 recipients

Algorithm	200 recipients (90 instances)							400 recipients (90 instances)						
	nb. var.	nb. cons.	nb. nzs.	nb. F1	nb. F2	nb. F3	nb. F4	nb. var.	nb. cons.	nb. nzs.	nb. F1	nb. F2	nb. F3	nb. F4
Cycle	90 861	174	623 669	-	-	-	-	1 526 232	400	10 571 785	-	-	-	-
Cycle/chain deactivation	319	143	1892	0	0.01	0.08	0	958	338	5636	0.02	0.03	0.26	0.24
Diving algorithm	319	143	1892	0	0.01	0.10	0	958	338	5636	0.02	0.03	0.26	0.24
Diving algorithm - DC	319	143	1892	0	0.01	0.10	0	955	338	5615	0.02	0.03	0.26	0.24
Hybrid algorithm - DC	318	143	1887	0	0.01	0.10	0	939	338	5523	0.02	0.03	0.26	0.24

We observe that:

- Our new approaches have significantly fewer variables and non-zero elements when optimising the last objective function z_5, indicating that the cycle/chain deactivation is extremely effective.
- The number of backtracking steps is relatively small for objective functions z_1 and z_2, supporting the hypothesis that the diving algorithm can reasonably assume that the bounds obtained after solving the linear relaxation of the first two objective functions are accurate.
- The number of variables when solving the last objective function z_5 is similar when comparing the hybrid algorithm and the diving algorithm without dominated chains. Out of 180 instances with 200 and 400 recipients, we counted:
  - 164 instances in which the hybrid algorithm had the same number of variables as the diving algorithm without dominating chains;
  - 8 instances in which the hybrid algorithm had more variables than the diving algorithm without dominating chains;
  - 8 instances in which the hybrid algorithm had fewer variables than the diving algorithm without dominating chains;

Table 9 contains the results of the five algorithms for instances with 600 recipients, which could be realistic size instances if half a dozen European countries were merging their pools.

Table 9: Average time (in seconds) required by each of the proposed algorithms to solve instances with 600 recipients for each value of f

Algorithm	$f = 0.01$ (30 instances)						$f = 0.05$ (30 instances)						$f = 0.10$ (30 instances)					
	TT	T1	T2	T3	T4	T5	**TT**	T1	T2	T3	T4	T5	**TT**	T1	T2	T3	T4	T5
Cycle	**576**	44	109	150	88	184	**7328**	343	1494	1927	1611	1951	**14805**	907	4550	2951	2375	4016
Cycle/chain deactivation	**24.1**	22.8	0.4	0.2	0.1	0.1	**196.8**	157.5	24.6	4.5	5.7	2.1	**486.2**	424.7	54.4	1.7	0.5	0.2
Diving algorithm	**21.7**	20.4	0.1	0.2	0.1	0.1	**163.3**	107.9	42.7	5.7	2.4	2.0	**280.9**	223.2	50.3	1.6	0.5	0.2
Diving algorithm - DC	**8.4**	7.4	0.1	0.2	0.1	0.1	**68.7**	40.2	17.3	4.5	2.6	2.3	**103.3**	73.7	23.7	1.4	0.3	0.2
Hybrid algorithm - DC	**2.0**	1.0	0.1	0.2	0.5	0.1	**11.0**	2.1	2.7	1.8	2.7	1.7	**4.9**	1.9	1.2	0.6	0.8	0.2

Not surprisingly, the table shows that the cycle formulation is much slower than our other methods, since it required more than 4 hours on average to solve an instance with 600 recipients and 60 non-directed donors while it only took our hybrid algorithm 4.9 seconds on average to solve the same instances. It is also interesting to observe that the proportion of non-directed donors f has a dramatic impact on the average time required to solve an instance for most approaches (the average times for $f = 0.10$ are between 10 and 30 times larger than the average times for $f = 0.01$), with the exception of the hybrid algorithm, which seems to scale well as f increases. This can be explained by the fact that $O(|\mathcal{P}|^3)$ new chains (and thus, new variables) are created in the cycle formulation for each new altruistic donor, while only $O(|\mathcal{P}|)$ new arcs are created in PICEF for each new altruistic donor (in most cases).

Table 10 displays the time spent on average by the hybrid algorithm to solve instances with up to 1400 recipients and with a proportion of non-directed donors up to $f = 0.20$. Results from previous tables are added for the sake of comparison.

Table 10: Average time (in seconds) required by the hybrid algorithm to solve each set of instances

$\lvert\mathcal{P}\rvert$	$f = 0.01$						$f = 0.05$						$f = 0.10$						$f = 0.15$						$f = 0.20$					
	TT	T1	T2	T3	T4	T5	**TT**	T1	T2	T3	T4	T5	**TT**	T1	T2	T3	T4	T5	**TT**	T1	T2	T3	T4	T5	**TT**	T1	T2	T3	T4	T5
50	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0
100	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0
200	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0	**0**	0	0	0	0	0
400	**1**	0	0	0	0	0	**1**	0	0	0	0	0	**1**	1	0	0	0	0	**2**	1	0	0	0	0	**2**	1	0	0	0	0
600	**2**	1	0	0	1	0	**11**	2	3	2	3	2	**5**	2	1	1	1	0	**6**	2	1	1	1	1	**7**	2	2	1	1	0
800	**7**	3	1	1	2	1	**15**	4	2	3	3	3	**13**	4	2	3	2	1	**16**	4	3	3	3	2	**13**	4	3	2	2	1
1000	**18**	7	3	4	3	2	**32**	7	4	10	6	4	**29**	7	5	6	7	4	**23**	7	5	5	4	1	**26**	8	6	6	4	1
1200	**26**	9	2	8	4	3	**49**	10	7	20	6	4	**53**	12	9	16	11	5	**42**	13	9	9	7	3	**46**	13	11	10	8	4
1400	**77**	16	4	28	12	17	**163**	17	13	77	24	32	**275**	18	14	49	69	125	**138**	16	14	28	28	51	**78**	17	16	26	13	6

We note that the hybrid algorithm can solve instances with up to 1400 recipients and up to 280 non-directed donors in slightly more than a minute on average, indicating that our algorithm scales well with both the number of recipients and the proportion of non-directed donors. As expected, there is a strong correlation between the number of recipients and the time required to solve an instance. For example, instances with 800 recipients and 80 non-directed donors

are solved in 13 seconds on average while instances with 1400 recipients and 140 non-directed donors are solved in 275 seconds on average.

The correlation between the proportion of non-directed donors and the time required to solve an instance is not as straightforward. Instances with $f = 0.01$ are always the fastest to solve, but we observe that instances with $f = 0.20$ do not necessarily take the longest to solve. For example, instances with 1400 recipients and 280 non-directed donors are solved in 78 seconds on average, while instances with the same number of recipients and half the number of non-directed donors are solved in 275 seconds on average. A possible explanation for this phenomenon is that instances with $f = 0.20$ do not necessarily produce larger mathematical models compared to instances with $f = 0.10$ because of the cycle/arc deactivation. Table 11 gives information about the size of the model solved when optimising the last objective function z_5, and indeed, we observe that the last model has 72 167 variables on average for instances with 1400 recipients and 280 non-directed donors, while it has 139 962 variables on average for instances with the same number of recipients and 140 non-directed donors.

Table 11: Model size of the hybrid algorithm when optimising z_5 on each set of instances

$\|\mathcal{P}\|$	$f = 0.01$			$f = 0.05$			$f = 0.10$			$f = 0.15$			$f = 0.20$		
	nb. var.	nb. cons.	nb. nzs.	nb. var.	nb. cons.	nb. nzs.	nb. var.	nb. cons.	nb. nzs.	nb. var.	nb. cons.	nb. nzs.	nb. var.	nb. cons.	nb. nzs.
50	11	20	58	19	26	103	12	21	65	52	37	223	79	44	341
100	40	46	220	87	60	523	105	73	545	179	85	846	268	98	1197
200	156	122	898	252	139	1542	546	167	3223	730	194	3907	956	218	4653
400	527	289	3137	838	340	5100	1453	385	8333	3054	429	16 100	3109	477	5233
600	802	473	4629	5155	547	32 985	3098	608	16 780	8027	673	42 098	9225	734	45 017
800	2516	678	15 095	4861	759	28 395	13 908	835	77 037	13 955	916	69 877	17 213	990	80 736
1000	5281	892	30 787	14 174	971	79 483	30 670	1064	168 546	20 200	1152	96 636	24 496	1248	109 174
1200	8148	1081	45 779	21 596	1182	119 370	38 887	1294	206 681	37 641	1394	182 593	42 065	1501	187 638
1400	16 595	1283	92 957	64 875	1401	363 897	139 962	1535	770 552	116 531	1638	622 379	72 167	1760	332 253

4.3 Randomly generated instances with the Spanish KEP objectives

We tested our best approach, the hybrid algorithm, on the Spanish KEP objectives. As we did not have any information regarding the score distribution, we only optimised the four first objective functions, which are: (i) maximise the number of transplants, (ii) maximise the number of distinct exchanges selected, (iii) maximise the number of cross arcs, and (iv) maximise the number of highly sensitised recipients selected. We used the same instances as the ones created for the UKLKSS objectives, and we used a binary indicator defining a recipient as highly sensitised if their cPRA value was at 0.85 or above.

According to [8], only cycles of size two and three are allowed in the Spanish KEP. There is no theoretical restriction on the maximum allowed length of chains, but given that the longest chain that has proceeded in Spain to date had length six, we impose this as an upper bound on the chain length for our experiments. All the techniques presented in Section 3 can be trivially extended to the Spanish KEP with the exception of the dominated chains, for which we detail in Figures 5 and 6 the most common generic cases of dominated chains of length five and six and their respective dominating set. Note that there exist more complex dominated chains of

length five and six (e.g., chain $A \to W \to X \to Y \to Z$ with cross arcs (A, X) (Z, W), and (W, Z) is dominated by chain $A \to X \to Y$ and cycle $[W, Z]$).

Figure 5: Dominated chains of length 5 for the Spanish KEP and their dominating sets

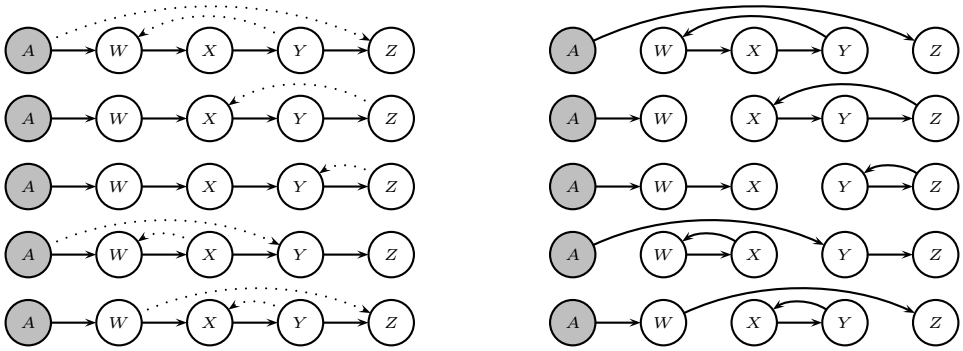
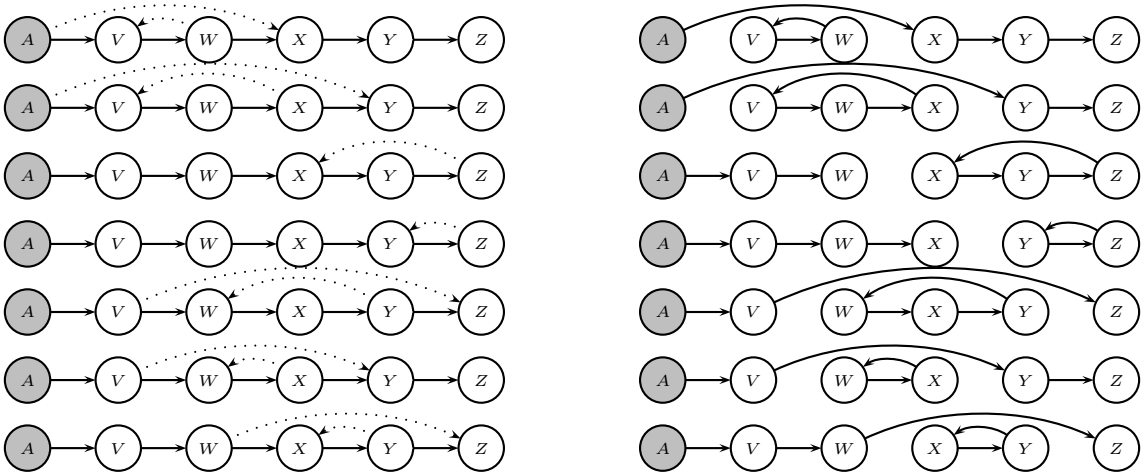


Figure 6: Dominated chains of length 6 for the Spanish KEP and their dominating sets



We first tested the adapted hybrid algorithm on a subset of the randomly generated instances and we compared its results with those obtained by the cycle formulation. The subset contains 270 instances with the number of recipients $|P| \in \{50, 100, 200\}$ and the proportion of non-directed donors $f \in \{0.01, 0.05, 0.10\}$. These values were chosen so that each algorithm could solve all the instances to optimality in a reasonable time. We then tested the hybrid algorithm on the remaining 1080 instances to show its scalability. We added a time limit to this set of experiments because some instances could not be solved even after ten hours of computation time. As we observed some memory issues occurring when optimising z_3 due to the high number of chains of size five and six that were generated during the transition between PICEF and the cycle formulation (above two billion for some instances with more than 1200 recipients), we also imposed a limit for the model size: if at any stage the solver was dealing with an integer model containing more than 75 million variables, then the algorithm stopped and the instance was considered as unsolved in 3600 seconds. Table 12 displays the time spent on average by the cycle and the hybrid algorithms optimising each objective function and the model size when optimising z_4 for instances with 50, 100, and 200 recipients.

The table shows that also for the Spanish KEP, the hybrid algorithm is significantly faster than the cycle formulation. This is even true for small instances as it takes more than one

Table 12: Detailed information on the hybrid and cycle algorithms for instances with 50, 100, and 200 recipients

|P|	Algorithm	**TT**	T1	T2	T3	T4	nb. var.	nb. cons.	nb. nzs.
50	Cycle	**0.55**	0.08	0.13	0.16	0.18	5425	32	46 399
	Hybrid algorithm - DC	**0.04**	0.00	0.02	0.01	0.01	17	25	101
100	Cycle	**81.28**	8.91	20.31	22.83	29.11	299 108	75	2 642 251
	Hybrid algorithm - DC	**0.10**	0.02	0.03	0.03	0.01	68	58	495
200	Cycle	**13 305.51**	998.1	3090.59	3510.77	5698.48	19 735 661	181	176 175 415
	Hybrid algorithm - DC	**0.61**	0.18	0.16	0.21	0.05	407	145	3320

minute on average for the cycle formulation to solve an instance with 100 recipients while it only takes 0.1 second on average to solve the same instance for the hybrid algorithm. This is due to the very large number of variables involved in the cycle formulation, which is even higher in the Spanish KEP than it is in the UKLKSS because the chain length is capped at six instead of four.

For each set of instances with up to 1400 recipients and with a proportion of non-directed donors up to $f = 0.20$, Table 13 displays the number of instances solved by the hybrid algorithm in less than one hour, the average time spent solving each instance, and the average number of times the bound $L_k (k = 1, \ldots, 4)$ had to be updated.

Table 13: Detailed information of the hybrid algorithm obtained when solving instances with up to 1400 recipients for each value of f

$\|\mathcal{P}\|$	$f = 0.01$						$f = 0.05$						$f = 0.10$						$f = 0.15$						$f = 0.20$					
	opt	**TT**	nb. F1	nb. F2	nb. F3	nb. F4	opt	**TT**	nb. F1	nb. F2	nb. F3	nb. F4	opt	**TT**	nb. F1	nb. F2	nb. F3	nb. F4	opt	**TT**	nb. F1	nb. F2	nb. F3	nb. F4	opt	**TT**	nb. F1	nb. F2	nb. F3	nb. F4
50	30	**0**	0	0	0	0	30	**0**	0	0	0	0	30	**0**	0	0	0	0	30	**0**	0	0	0	0	30	**0**	0	0	0	0
100	30	**0**	0	0	0	0	30	**0**	0	0	0.1	0.1	30	**0**	0	0	0.1	0	30	**0**	0	0	0	0	30	**0**	0	0	0	0
200	30	**0**	0	0	0	0	30	**1**	0.1	0.2	0.2	0.2	30	**1**	0	0	0	0.1	30	**1**	0	0	0.1	0.1	30	**1**	0	0	0.2	0.1
400	30	**1**	0	0.1	0.1	0.1	30	**12**	0.2	0.5	1.8	1	30	**37**	0	0	0.5	0.3	30	**49**	0	0	0.4	0.5	30	**31**	0	0	0.1	0.1
600	30	**7**	0	0.1	0.9	0.9	27	**586**	0	0.2	2.2	2.5	29	**339**	0	0	0.7	1	26	**697**	0	0.1	0.6	0.7	29	**407**	0	0	0.4	0.4
800	30	**32**	0.1	0.4	1.8	2.1	26	**992**	0	0	1.7	1.9	28	**596**	0	0	0.6	1.1	26	**1123**	0	0	0.5	0.6	25	**1395**	0	0	0.3	0.4
1000	30	**90**	0	0.2	2.3	2.6	19	**1711**	0	0.1	1.4	1.7	23	**1795**	0	0	0.6	0.9	16	**2525**	0	0	0.3	0.8	14	**2481**	0	0	0.4	0.3
1200	30	**112**	0	0.1	0.9	1.1	16	**2322**	0	0.1	0.4	0.7	7	**3189**	0	0	0.2	0.5	8	**2878**	0	0	0.2	0.4	2	**3504**	0	0	0	0.1
1400	24	**950**	0	0.2	2	2.4	7	**3022**	0	0.1	0.3	0.4	7	**3110**	0	0	0.1	0.5	5	**3368**	0	0	0	0.1	1	**3504**	0	0	0.2	0

We observe that the hybrid algorithm does not scale as well for the Spanish KEP as it did for the UKLKSS since only one instance with 1400 recipients and 280 non-directed donors could be solved in less than an hour. Solving that instance is still a significant achievement since there are theoretically up to $O(10^{18})$ possible chains of length six (280×1400^5), which means up to $O(10^{18})$ variables if the cycle formulation was used, which is not solvable by any state-of-the-art ILP solvers. It seems that for the Spanish KEP, both the number of recipients and the proportion of non-directed donors have a strong impact on the time required to solve an instance. It is not surprising considering that each non-directed donor can potentially initiate $O(|\mathcal{P}|^5)$ chains, which may result in an excessive number of variables when the hybrid algorithm switches from

PICEF to the cycle formulation (between z_2 and z_3). Interestingly, we also notice that there are more backtracking steps in the Spanish KEP than in the UKLKSS. This is probably due to a weaker continuous relaxation caused by longer chains. We tried several extensions of the hybrid algorithms, either without any cycle/chain deactivation for z_3 and z_4, or with at most one iteration before reactivating every cycle/chain and removing bounds L_3 and L_4, but we obtained results that were significantly worse than those presented in the table.

4.4 Randomly generated instances with the Dutch KEP objectives

We also tested our best algorithm on the Dutch KEP objectives. As we did not have any information regarding the transplant centres nor on the waiting time experienced by the recipients, we only optimised the four first objective functions, which are: (i) maximise the number of transplants, (ii) maximise the number of transplants between a donor and recipient with the same blood group, (iii) prioritise the transplants to hard-to-match recipients, and (iv) minimise the length of the largest selected cycle.

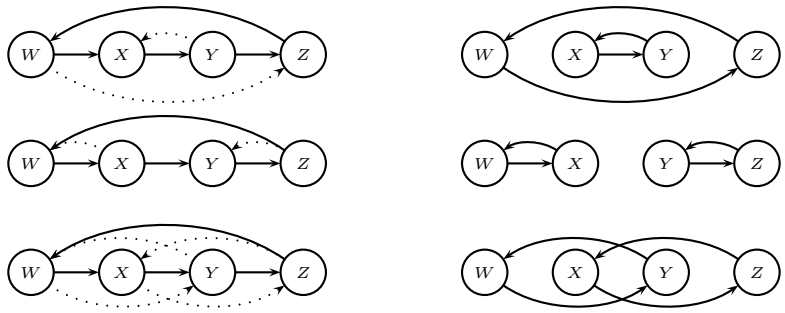
A more precise interpretation of objective function (iii) was obtained through personal correspondence [38]. For each recipient, their Match Probability (MP) is calculated based on their cPRA, their unacceptable HLA matches, and the distribution of HLA types in the current donor pool [25]. The Dutch system then attempts to find a solution that achieves optimal values on objectives (i) and (ii), and maximises the number of transplants to the recipients with the lowest MP, and subject to this, then maximises the number of transplants to recipients with the second-lowest MP, and so forth. As distributions of specific HLA types and unacceptable HLA matches are unknown, we approximated a recipient's MP as one minus their cPRA value. We ordered the recipients by non-decreasing MP, and assigned to each recipient r a value v_r that indicates how many distinct MP values are less than or equal to the MP value of r. As an example, if the MP values of 5 patients are $(0, 0, 0.15, 0.5, 0.5)$, then the v_r values are $(1, 1, 2, 3, 3)$. In our experiments we approximated objective (iii) by using a single objective function that minimises the sum of the v_r values of the recipients selected for transplant. This was done to avoid additional complexity, and in the knowledge that we would have been unable to generate realistic MP values in any case.

According to [8], the Dutch KEP allows cycles and chains up to size four. Most of the techniques presented in Section 3 can be trivially extended to the Dutch KEP. As a noticeable difference, we observe that since none of the objective functions involves maximising the number of cross arcs, PICEF can be used instead of the cycle formulation as a basic algorithm to compare with our best approach, which is now the diving algorithm without dominated cycles (but using PICEF instead of the cycle formulation for the models). Indeed, as there is no interest in switching from PICEF to the cycle formulation during the optimisation, there is no reason to try a hybrid approach for the Dutch KEP. Also, as mentioned in Section 3, it is not computationally interesting to forbid dominated chains in PICEF because it increases the model size. However, since we now have cycles of size up to four, and since forbidding a dominated cycle simply sets its corresponding value to 0 (and thus, reduces the model size), we detail in Figure 7 three generic cases of dominated cycles of length four. Note that in addition to the existence of specific cross arcs, we also have to make sure that any dominated cycle does not

involve more transplants between a donor and a recipient with the same blood group than its dominating set (because of the second objective function).

During our experiments on the Dutch KEP, we discovered that for both algorithms, it took the solver a significant amount of time to find a feasible solution when solving a model with integrality constraints. As a result, we used so-called *warm starts* as an additional feature: in the diving algorithm (respectively, in PICEF), the best integer solution obtained when optimising z_3 (respectively, z_1 to z_3) was saved and given to the solver as a warm start when optimising z_4 (respectively, z_2 to z_4). We also tested this feature for the UK and Spanish KEPs, but we did not notice any improvement in terms of computing time. Based on the solver outputs, our hypothesis is that it is significantly harder for the solver to find a feasible integer solution in PICEF than it is in the cycle formulation. Considering that in the case of the UKLKSS objectives, the hybrid algorithm never obtains a feasible integer solution of PICEF for z_1 and very rarely obtains one for z_2, then in most instances there is no integer solution that could be used as warm start when optimising z_3 (before the switch to the cycle formulation for z_4 and z_5).

Figure 7: Dominated cycles of length 4 for the Dutch KEP and their dominating sets (dominated cycles must not involve more transplants with the same blood group than their dominating set)



We tested the adapted diving algorithm without dominated cycles on a subset of the randomly generated instances created for the UKLKSS objectives and we compared its results with those obtained by PICEF. The subset contained 750 instances with the number of recipients $|P| \in \{50, 100, 200, 400, 600\}$ and the proportion of non-directed donors $f \in \{0.01, 0.05, 0.10, 0.15, 0.20\}$. These values were chosen so that each algorithm could solve all the instances to optimality in a reasonable time. We did not test the adapted diving algorithm without dominated cycles on larger instances because there were too many cycles of length four that were generated in the early stages of the approach. Similar to what was presented in Algorithm 2, our new approach dived until z_3, and checked at the same time if the bounds L_1 and L_2 were correct.

Table 14 displays the time spent on average by PICEF and the diving algorithm without dominated cycles (attribute "-DCl") optimising each objective function, the model size when optimising z_4, and the average number of times the bound $L_k (k = 1, \ldots, 4)$ had to be updated for instances with up to 600 recipients.

We observe that even for the Dutch KEP, our approach is faster than a plain ILP formulation such as PICEF. This is particularly true for instances with 400 recipients or more. For example, it takes 100 seconds on average for PICEF to solve an instance with 400 recipients while it only

Table 14: Detailed information on the diving algorithm without dominated cycles and PICEF for instances with up to 600 recipients

|P|	Algorithm	**TT**	T1	T2	T3	T4	nb. var.	nb. cons.	nb. nzs.	nb. F1	nb. F2	nb. F3	nb. F4
50	PICEF	**0.07**	0.01	0.02	0.02	0.02	340	196	1917	-	-	-	-
	Diving algoithm - DCl	**0.03**	0.00	0.01	0.01	0.01	114	132	597	0.01	0.01	0.06	0
100	PICEF	**0.37**	0.08	0.08	0.09	0.11	2466	1487	16 971	-	-	-	-
	Diving algoithm - DCl	**0.09**	0.02	0.01	0.03	0.02	479	410	2842	0	0	0.05	0
200	PICEF	**4.11**	0.78	0.94	1.21	1.16	22 671	16 879	178 082	-	-	-	-
	Diving algoithm - DCl	**0.67**	0.24	0.06	0.25	0.09	2666	1840	17 950	0	0.01	0.18	0
400	PICEF	**100.13**	17.35	23.42	39.60	19.49	299 249	269 390	2 564 640	-	-	-	-
	Diving algoithm - DCl	**14.40**	4.82	1.18	6.56	1.55	31 769	25 029	253 554	0	0.01	0.39	0
600	PICEF	**894.61**	105.6	135.29	564.42	88.07	1 310 222	1 235 630	11 465 402	-	-	-	-
	Diving algoithm - DCl	**389.22**	24.10	7.48	344.89	11.49	207 614	181 770	1 757 257	0	0	0.44	0

takes a bit less than 15 seconds on average to solve the same instance for the diving algorithm. We observe a significant reduction in the number of variables when optimising z_4 which can mainly be attributed to the cycle/arc deactivation algorithms. Indeed, for instances with 600 recipients, out of 1 200 292 cycles of size four that were created on average, only 10 913 were dominated, meaning that the 1.1 million difference in the average number of variables between the two models is due to cycle/arc deactivation. We also observe that the number of variables grows very fast as the number of recipients increases. This can be explained by the fact that the Dutch KEP allows cycles of size four, that PICEF requires one variable for each feasible cycle, and that not so many cycles of size four are necessarily removed by the cycle/arc deactivation algorithm at an early stage of the algorithm since none of the first objective functions requires to limit the use of these large cycles. In order to reduce even further the number of variables, an interesting approach would be to extend the column generation procedure developed in [20] for PICEF to tackle Dutch KEP objectives.

Finally, we observe that the number of backtracking steps is barely noticeable for z_1 and z_2 and more significant for z_3, indicating that the diving paradigm used (diving until z_3 and set $\bar{\Lambda}_2$ and $\bar{\Lambda}_3$ to 1) seems appropriate. We also notice that it can be time-consuming for the diving algorithm to solve z_3, which is mainly explained by the fact that, in most cases, the algorithm does not have an integer solution to use as a warm start when optimising z_3. Note that a heuristic is not an easy way to alleviate this problem since the solution given as a warm start needs to satisfy the bounds obtained when optimising z_1 and z_2.

5 Conclusion

Hierarchical optimisation is an important feature of many kidney exchange programmes globally. KEPs have established the set of objective functions that they wish to optimise and many use the cycle formulation to model their problem. Such models are manageable when the number of recipients is limited, but quickly become impractical when programmes reach 400–600 recipients. These numbers may seem high at the current time, but some European countries with large

and successful KEPs such as the United Kingdom already have around 300 recipients in their pool at each run. It is also known that cooperation among European countries for transnational KEPs could increase the number of transplants [21], so it is not impossible to envisage merged pools of more than 1000 recipients in the foreseeable future. It is important for algorithmic techniques to handle such pool sizes to be developed well ahead of the time that they may be required. Most countries do not completely agree on the set of objective functions that should be optimised, but all agree that the number of transplants should be maximised at some level [8, 9]. As a result, they generally prefer to use exact approaches since an additional unit in the objective function may be associated with an additional transplant, and thus, an additional life saved.

We described three tools to improve the performances of both the cycle formulation and PICEF: a cycle/chain deactivation algorithm and a dominated chain detector to remove unnecessary variables, and a diving algorithm to remove the necessity to solve complex integer linear programming models. We also showed that it was possible to transition from PICEF to the cycle formulation between the optimisation of two objective functions. As a result, PICEF can be used until a critical objective function (such as cross arcs maximisation) needs to be optimised.

We tuned our approaches for the UKLKSS and showed that they could be up to 1000 times faster than the cycle formulation. We could even solve instances with up to 1400 recipients and 280 non-directed donors in less than two minutes on average. Problems of such scale have never been solved before by the algorithms used for the UKLKSS. We also demonstrated that our approaches could be adapted for KEPs from other countries such as Spain and the Netherlands. For the Spanish KEP, our best approach was also several orders of magnitude faster than the cycle formulation, and we could solve an instance with 1400 recipients and 280 non-directed donors in less than an hour, which is significant considering that chains of length six are allowed. For the Dutch KEP, our best approach was almost up to 10 times faster than PICEF, and we could solve instances with up to 600 recipients and 120 non-directed donors, which is also significant since cycles of size four are allowed.

Naturally, we do not claim that the algorithms proposed in this work are ready-to-use tools for all the European KEPs. Among the possible limitations of our work, we remark that the instances we solved were generated with UK-specific parameters, that some objective functions in the Spanish and Dutch KEP were not optimised because we did not have any information regarding the data distribution, and that the tuning of our algorithms was focused on the UKLKSS and could be improved for the other countries. In addition, we used a commercial solver, which is sometimes too expensive (or simply unnecessary) for some KEPs. Lastly, our proposed methods are specifically designed for hierarchical optimisation, as is prevalent in Europe, thus only the dominated chains may be applicable to KEPs that optimise a single objective such as is common in KEPs based in the USA.

Still, we successfully showed the effectiveness of our tools for the UKLKSS and their adaptability to other countries' requirements. Some of the tricks we propose could even be used for hierarchical optimisation in other areas, provided that the bounds given by the continuous relaxations are tight. We leave as future work the adaptation of our techniques to multi-country kidney exchange programmes, a problem in which (i) pools of several countries are merged, (ii)

a consensus on the objective function has to be found, and (iii) a minimum number of transplants per country is imposed, so that no country loses any transplants by participating in the programme.

Acknowledgements

This research was supported by the Engineering and Physical Science Research Council through grant EP/P029825/1 (first four authors) and grant EP/P028306/1 (fifth and sixth authors). The authors would like to thank Joris van de Klundert for helpful insights regarding objective (iii) used in the Dutch KEP.

References

[1] `https://www.kidney.org/atoz/content/dialysisinfo` (Dialysis | National Kidney Foundation). Accessed 11 June 2020.

[2] `https://optn.transplant.hrsa.gov/media/1200/optn_policies.pdf` (Organ Procurement and Transplantation Network (OPTN) Policies). Accessed 4 June 2020.

[3] D.J. Abraham, A. Blum, and T. Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In *Proceedings of EC '07: the 8th ACM Conference on Electronic Commerce*, pages 295–304. ACM, 2007.

[4] F. Alvelos, X. Klimentova, and A. Viana. Maximizing the expected number of transplants in kidney exchange programs with branch-and-price. *Annals of Operations Research*, 272:429–444, 2019.

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

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

[7] B. Bikbov, N. Perico, and G. Remuzzi. Disparities in chronic kidney disease prevalence among males and females in 195 countries: Analysis of the global burden of disease 2016 study. *Nephron*, 139:313–318, 2018.

[8] P. Biró, B. Haase-Kromwijk, T. Andersson, E. I. Ásgeirsson, T. Baltesová, I. Boletis, C. Bolotinha, G. Bond, G. Böhmig, L. Burnapp, K. Cechlárová, P. Di Ciaccio, J. Fronek, K. Hadaya, A. Hemke, C. Jacquelinet, R. Johnson, R. Kieszek, D. R. Kuypers, R. Leishman, M.-A. Macher, D. Manlove, G. Menoudakou, M. Salonen, B. Smeulders, V. Sparacino, F. C. R. Spieksma, M. O. Valentín, N. Wilson, and J. van der Klundert. Building kidney exchange programmes in Europe–an overview of exchange practice and activities. *Transplantation*, 103:1514–1522, 2019.

[9] P. Biró, J. van de Klundert, D. Manlove, W. Pettersson, T. Andersson, L. Burnapp, P. Chromy, P. Delgado, P. Dworczak, B. Haase, A. Hemke, R. Johnson, X. Klimentova, D. Kuypers, A. N. Costa, B. Smeulders, F. Spieksma, M. O. Valentín, and A. Viana. Modelling and optimisation in European kidney exchange programmes. *European Journal of Operational Research*, 2020, to appear.

[10] A. Blum, J.P. Dickerson, N. Haghtalab, A.D. Procaccia, T. Sandholm, and A. Sharma. Ignorance is almost bliss: Near-optimal stochastic matching with few queries. *Operations Research*, 68:16–34, 2020.

[11] I. Caragiannis, A. Filos-Ratsikas, and A. D. Procaccia. An improved 2-agent kidney exchange mechanism. *Theoretical Computer Science*, 589:53–60, 2015.

[12] D. S. Chisca, M. Lombardi, M. Milano, and B. O'Sullivan. A sampling-free anticipatory algorithm for the kidney exchange problem. In L.-M. Rousseau and K. Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 146–162, Cham, 2019. Springer International Publishing.

[13] D.S. Chisca, M. Lombardi, M. Milano, and B. O'Sullivan. Logic-based benders decomposition for super solutions: An application to the kidney exchange problem. In T. Schiex and S. de Givry, editors, *Principles and Practice of Constraint Programming*, pages 108–125, Cham, 2019. Springer International Publishing.

[14] M. Constantino, X. Klimentova, A. Viana, and A. Rais. New insights on integer-programming models for the kidney exchange problem. *European Journal of Operational Research*, 231(1):57 – 68, 2013.

[15] M. Constantino, X. Klimentova, A. Viana, and A. Rais. New insights on integer-programming models for the kidney exchange problem. *European Journal of Operational Research*, 231:57–68, 2013.

[16] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, 1963.

[17] S. Das, J. P. Dickerson, Z. Li, and T. Sandholm. Competing dynamic matching markets. In *Proceedings of the Conference on Auctions, Market Mechanisms and Their Applications (AMMA)*, volume 112, page 245, 2015.

[18] M. Delorme and M. Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32:101–119, 2020.

[19] J. P. Dickerson, A. D. Procaccia, and T. Sandholm. Dynamic matching via weighted myopia with application to kidney exchange. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, page 1340–1346. AAAI Press, 2012.

[20] J.P. Dickerson, D.F. Manlove, B. Plaut, T. Sandholm, and J. Trimble. Position-indexed formulations for kidney exchange. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, pages 25–42, New York, NY, USA, 2016. ACM.

[21] http://www.enckep-cost.eu (ENCKEP website). Accessed 20 May 2020.

[22] I. Gao. Fair matching in dynamic kidney exchange. Technical Report 1912.10563, Computing Research Repository, Cornell University Library, 2019. Available from http://arxiv.org/abs/1912.10563.

[23] https://jamestrimble.github.io/kidney-webapp/#/generator (Dataset generator written by James Trimble). Accessed 30 April 2020.

[24] A. Hart, J. M. Smith, M. A. Skeans, S. K. Gustafson, D. E. Stewart, W. S. Cherikh, J. L. Wainright, A. Kucheryavaya, M. Woodbury, J. J. Snyder, B. L. Kasiske, and A. K. Israni. OPTN/SRTR 2015 annual data report: Kidney. *American Journal of Transplantation*, 17:21–116, 2017.

[25] K.M. Keizer, M. de Klerk, B.J.J.M. Haase-Kromwijk, and W. Weimar. The Dutch algorithm for allocation in living donor kidney exchange. *Transplantation Proceedings*, 37:589–591, 2005.

[26] X. Klimentova, J. P. Pedroso, and A. Viana. Maximising expectation of the number of transplants in kidney exchange programmes. *Computers & Operations Research*, 73:1 – 11, 2016.

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

[28] M. Lin, J. Wang, Q. Feng, and B. Fu. Randomized parameterized algorithms for the kidney exchange problem. *Algorithms*, 12(2):50, 2019.

[29] V. H. Mak-Hau. 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, 2017.

[30] D.F. Manlove and G. O'Malley. Paired and altruistic kidney donation in the UK: Algorithms and experimentation. *ACM Journal of Experimental Algorithmics*, 19:1–21, 2015.

[31] D. C. McElfresh, H. Bidkhori, and J. P. Dickerson. Scalable robust kidney exchange. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, AAAI'19, pages 1077–1084. AAAI Press, 2019.

[32] Global Burden of Disease Collaborative Network and Institute for Health Metrics and Evaluation. Global Burden of Disease Study, 2018. http://ghdx.healthdata.org/gbd-results-tool?params=gbd-api-2017-permalink/22c285161ee9f4b752353c203aadcc0c.

[33] F.T. Rapaport. The case for a living emotionally related international kidney donor exchange registry. In *Transplantation proceedings*, volume 18(3) Suppl. 2, pages 5–9, 1986.

[34] A.E. Roth, T. Sönmez, and M.U. Ünver. Kidney exchange. *Quarterly Journal of Economics*, 119(2):457–488, 2004.

[35] A.E. Roth, T. Sönmez, and M.U. Ünver. Pairwise kidney exchange. *Journal of Economic Theory*, 125(2):151–188, 2005.

[36] A.E. Roth, T. Sönmez, and M.U. Ünver. Efficient kidney exchange: Coincidence of wants in a market with compatibility-based preferences. *American Economic Review*, 97(3):828–851, 2007.

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

[38] J. van de Klundert. Personal Communication, May 2020.

[39] W. Wang, M. Bray, P. X.K. Song, and J. D. Kalbfleisch. An efficient algorithm to enumerate sets with fallbacks in a kidney paired donation program. *Operations Research for Health Care*, 20:45 – 55, 2019.

[40] R. A. Wolfe, E. C. Roys, and R. M. Merion. Trends in organ donation and transplantation in the United States, 1999–2008. *American Journal of Transplantation*, 10:961–972, 2010.