Maximum-expectation matching under recourse

João Pedro Pedroso[a], Shiro Ikeda[b]

[a] *CMUP and Department of Computer Science, Faculty of Sciences, University of Porto*
R Campo Alegre, 4169– 007 Porto, Portugal
[b] *The Institute of Statistical Mathematics*
10-3 Midori-cho, Tachikawa, Tokyo 190-8562, Japan

Abstract

This paper addresses the problem of maximizing the expected size of a matching in the case of unreliable vertices and/or edges. The assumption is that the solution is built in several steps. In a given step, edges with successfully matched vertices are made permanent; but upon edge or vertex failures, the remaining vertices become eligible for reassignment. This process may be repeated a given number of times, and the objective is to end with the overall maximum number of matched vertices.

An application of this problem is found in kidney exchange programs, going on in several countries, where a vertex is an incompatible patient-donor pair and an edge indicates cross-compatibility between two pairs; the objective is to match these pairs so as to maximize the number of served patients. A new scheme is proposed for matching rearrangement in case of failure, along with a prototype algorithm for computing the optimal expectation for the number of matched edges (or vertices), considering a possibly limited number of rearrangements.

Computational experiments reveal the relevance and limitations of the algorithm, in general terms and for the kidney exchange application.

Keywords: Matching; Combinatorial optimization; Stochastic optimization

1. Introduction

Algorithms for matching have in the past years raised interest as a research topic on a particular application: maximizing the number of transplants in kidney exchange programs. These programs have been organized in several countries, in order to provide patients with kidney failure with an

alternative to traditional treatment, in cases where they have a donor willing to provide a kidney, but the pair is not physiologically compatible (de Klerk et al., 2005; Segev et al., 2005; Saidman et al., 2006; Biro et al., 2009). These programs are based on the concept of *exchange* between two patient-donor pairs: donors are allowed to provide a kidney to the other pair's patient, if compatibility exists, so that both patients benefit.

Figure 1 (left) illustrates the simplest case with only two pairs, (P_1, D_1) and (P_2, D_2). Donor D_1 of the first pair is allowed to give a kidney to patient P_2 of the second pair, and patient P_1 may get a kidney from donor D_2. These graphs concern only preliminary compatibilities, which must be reassessed prior to actual transplant, by confirming the availability of the intervening persons and through additional medical exams.

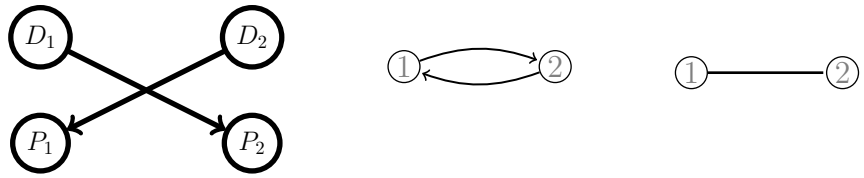


Figure 1: An exchange between two incompatible pairs; arcs represent a preliminary assessment compatibility, and arrows define a possible exchange (left). Representation of this situation as a directed graph (center), and as a graph where an edge stands for a pair of opposite arcs between two nodes (right).

The situation depicted in Figure 1 may be extended to cycles with more than two vertices; typical programs may allow two to five vertices in an exchange, though smaller cycles are preferable due to hospital logistics, and other reasons (Biró et al., 2019, 2021). The optimization problem underlying a standard kidney exchange usually considers maximizing the number of transplants (de Klerk et al., 2005; Segev et al., 2005), though other criteria have been proposed; *e.g.*, the overall weight assigned to each transplant (Li et al., 2014; Manlove & O'malley, 2015), and the expected number of transplants for selected cycles (Pedroso, 2014) or subsets of vertices (Klimentova et al., 2016). Here, we take a more general view: we consider that after a failure, any recourse solution may be chosen as long as it does not involve previously matched vertices or vertices/edges that failed. Recourse may be repeated an undetermined number of times. This is relevant in practice, as data available show a rate of positive crossmatch (*i.e.*, arc failure) of up to 44% (Glorie, 2012), and in many situations there are no natural obstacles

limiting recourse to a single trial.

Throughout this paper we will only consider the case of cycles of two vertices; in such a case, the directed graph representing an instance may be simplified, by replacing opposite arcs between two nodes by an edge, and ignoring all the other arcs (extensions are mentioned in Section 4). The relevant base problem is maximum-weighted matching in a graph, for which efficient algorithms are known. However, maximum-expectation matching under recourse seems to be a inherently intractable problem; we provide some examples, discuss how to specify and compute a solution, and analyze the behavior of the proposed algorithm with some instances of the real-world situation that motivated this work, the kidney exchange problem.

In the model proposed, graph's vertices and edges can exist in three states: undecided, successful, or failed. Initially, all vertices and edges are undecided. Queries alter their states: upon a query, a live edge $\{i, j\}$ may fail with probability p_{ij} and be successful with probability $1 - p_{ij}$; if successful, all adjacent edges are deleted. Vertices are treated in a similar way to edges. The remaining graph is called the *residual graph*.

The algorithm operates in rounds, which we also call *observations*. At the beginning of each observation, a non-empty matching M is selected from the undecided edges. While the order of queries within a round doesn't matter, the order of rounds themselves is crucial. Queries at a given round reveal a realization of probabilities, which will determine the residual graph and influence queries in the subsequent rounds.

The process ends when no edges remain undecided. The goal is to maximize the expected number of successful edges at the end of the process.

Our contributions are the following. This work is put in the context of the current state-of-the-art in Section 2. In Section 3, we model and analyze matching with repeated recourse, for the relevant cases where probabilities of failure of vertices and arcs are known. An algorithm for tackling this problem is provided in Section 4, and its behavior is experimentally assessed in Section 5. We conclude with Section 6, which presents considerations on the applicability and limitations of the approach proposed.

2. Literature review

Algorithms for finding a maximum matching in a graph have been proposed in the 1960's by Edmonds (1965b), based on some properties of maximum matchings by Berge (1957). Maximum-weighted matching in a graph

has been studied in Edmonds (1965a), who proposed an algorithm involving a formulation of the problem as a linear program, linear programming duality, and the previous algorithm for maximum matching. The complete algorithm is polynomial, solving the weighted matching problem in $O(n^4)$ time. An analysis of efficient algorithms and data structures for this problem is presented in Galil (1986).

A stochastic, two-stage version of the matching problem has been studied in Kong & Schaefer (2006), where the authors consider that each edge has two weights, corresponding to matching in the first-stage and in the second-stage. After the first-stage decision, different edge weights are realized in second-stage scenarios, where each edge weight is assigned a value among a set of possibilities; for each scenario, the second-stage decision chooses a matching over vertices unmatched in the first-stage. The decision version of this problem was shown to be $\mathsf{NP}-$complete. This work is related to ours, though it considers only two stages and the outcome of the first stage is deterministic.

The deterministic version of a kidney exchange problem, where all the elements of the graph are assumed reliable, is naturally modeled through integer optimization; a summary of models for the kidney exchange problem has been presented and analyzed by Constantino et al. (2013), and more recent formulations have been proposed in Anderson et al. (2015); Dickerson et al. (2016); Delorme et al. (2023). Here, we study this problem under uncertainty, in a situation where vertices and/or edges may fail. This problem has been raised in Li et al. (2011) and Chen et al. (2012), who propose a simulation system for maximizing the expected utility when arcs are subject to failure, in a dynamic version of the problem. Concerning the static version, Dickerson et al. (2013) propose a branch-and-price method for tackling the maximization of the expected number of transplants under uncertainty, with no recourse action.

A two-phase method for dealing with edges that failed in the first phase of a kidney exchange problem was presented in Anderson et al. (2015). The authors deal with the possibility of edges becoming ineligible for matching in the following manner. In the first phase, a subset of the edges in the graph are selected to be tested for failure, based on operational constraints. In the second phase, edges which failed in phase one are removed, and a probabilistic kidney exchange problem is resolved by means of random variables which indicate if an edge not previously checked can be used or not. The objective is to maximize the expected size of the phase-two matching.

The notion of recourse on solutions for kidney exchange was tackled in Pedroso (2014). There, a method for dealing with reconfiguration within a cycle's vertices was proposed, with the objective of maximizing the expectation of the overall number of transplants. Reconfiguration of the solution after a failure is observed has also been addressed in Manlove & O'malley (2015), where the number of effective 2-cycles is taken into account in the weights assigned to 3-cycles—hence prefering cycles which can be "reconfigured" upon observing certain failures. Reconfiguration involving vertices outside a cycle, in what has been called *subset recourse*, was considered in Klimentova et al. (2016).

A related problem is that of maximum matching in a graph whose edges are unknown but can be accessed via queries, a problem dealt with in (Blum et al., 2020). For each edge there is a known existence probability; if all edges were queried, then the maximum matching could be easily be determined. Here, the question is rather to determine the subset of edges that should be queried for achieving a given fraction of the omniscient solution. Both *adaptive* and *non-adaptive* algorithms are proposed for this, where in the adaptive version queries are conditioned on the answers to previous queries and in the non-adaptive there is only one round of queries.

The setting of our paper is close to a slightly different problem, the *query-commit problem*, motivated by applications in kidney exchanges and online dating (Chen et al., 2009). Here, given a graph where edges have distinct probabilities of existing, it is possible to query (probe) the edges; if a queried edge exists, then its endpoints are irrevocably matched [1] . The authors propose a greedy algorithm for this problem:

Algorithm 1: Greedy

1 Sort all edges in E by probabilities, say, $p(e_1) \geq p(e_2) \geq \ldots \geq p(e_m)$;
2 For $i = 1, \ldots, |E|$, if the two endpoints of e_i are available, probe e_i;

The problem has been studied with the goal of finding a querying strategy which maximizes the expected size of the matching obtained (Molinaro & Ravi, 2011; Goel & Tripathi, 2012), where several matching heuristics are analyzed experimentally. Additionally, both Chen et al. (2009) and Bansal et al. (2012) consider a limit on the number of times a vertex may be probed.

[1] Notice that in the context of kidney exchange both edges and vertices may probed; probing an edge relates to a final check that transplants between involved donors and patients are possible, while probing its vertices assesses if each pair remains available.

Approximation schemes for the bipartite case have been studied in (Gamlath et al., 2019; Derakhshan & Farhadi, 2023).

Chen et al. (2009) also mentions a generalization of the greedy algorithm, considering that the algorithm proceeds in rounds and is allowed to probe a set of edges (which have to be a matching) in each round, and that there is a maximum number of rounds k:

Algorithm 2: Greedy-k

```
for each round i = 1, ..., k: do
    compute a maximum weighted matching in the current graph;
    probe all edges in the matching;
end
```

This is the setting considered in this paper, for which we propose an exact approach.

A different approach to dealing with matching in unreliable graphs is searching for robust solutions, which provide a good outcome in case of failure of some components. This has been dealt with in Carvalho et al. (2021), where robust optimization models are proposed for protection against failure. The authors explicitly consider the possibility of flexible response upon failure by means of recourse, which includes considering back-arcs and full recourse. In this last approach, a second-stage solution may detach previously matched vertices, though the number of these occurrences is minimized. Another approach is proposed in Blom et al. (2024), which introduces a policy for recourse named *Fix Successful Exchanges*, where cycles and chains of the initial solution that do not fail (are not affected by a subsequent *attack*) are eligible in a recourse solution.

Our approach differs from these, in that we consider that besides edges vertices may also fail; it is assumed that the probability of failure is known. Chen et al. (2009) consider that each vertex has an associated parameter, the *patience*, which is the maximum number of times edges at a given vertex may be probed. We only consider the possibility of having a limit on the total number of rounds allowed in the solution, which we believe is more realistic.

3. Preliminaries and problem description

The input to our problem is a graph $G = (V, E)$. A matching M in G is a subset of E such that no two edges in M have a vertex in common. A maximal matching M has the property that no edge can be added to M (*i.e.*,

M is not a proper subset of any other matching in G). Vertices $i \in V$ may fail with probability p_i, and edges $\{i, j\}$ (also denoted by ij) may fail with probability p_{ij}.

In our setting, after a matching is proposed, the matched edges and vertices are *queried* or *probed*. An *observation* is the set of queries in a matching, where each matched edge *succeeds* if neither the edge nor the incident vertices fail. If a matched edge succeeds, the incident vertices are "served" and the edge will no longer be considered. If a matched edge fails, the incident vertices are not served; the failure is permanent, and hence that edge is no longer considered either. The search for *a posteriori* attempts to serve more vertices is called *recourse*; it is similar to the original problem, except that it may not involve vertices that have been served already.

After each observation/round, the graph is updated into a *residual graph*. Edges are eliminated at each observation as follows:

- successful edges, and all edges at their end vertices, are removed;
- edges that failed are removed;
- edges at vertices that failed are removed;
- vertices that failed or became isolated are removed.

If a proposed matching is maximal and in the observation there are no failures, the residual graph will contain no edges.

The goal is to find a decision tree of queries, guiding through the path to follow when a proposed matching succeeds or fails, until the residual graph is empty. The objective is to maximize the expected number of edges (or, possibly, the associated weight). Hence, for a given graph G, the problem is to progressively find a matching resulting in

$$\max_{M \in \mathcal{M}(G)} E(M),$$

where $\mathcal{M}(G)$ is the set of all matchings in G and $E(M)$ is the expected cardinality of matching M. Denoting by $V(M)$ the set of vertices incident to edges in M, this last value can be conveyed through a recursive mathematical

equation:

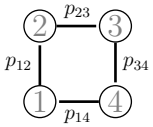
$$E(M) = \sum_{U \in 2^{M \cup V(M)}} \prod_{ij \in E(U)} (1 - p_{ij}) \cdot \prod_{i \in V(U)} (1 - p_i) \cdot \prod_{ij \in M \setminus E(U)} p_{ij} \cdot \prod_{i \in V(M) \setminus V(U)} p_i \left(|S(U)| + \max_{Q \in \mathcal{M}(R(U))} E(Q) \right),$$

where each $U \in 2^{M \cup V(M)}$ is a set of edges and vertices that do not fail (denoted by $E(U)$ and by $V(U)$, respectively, *i.e.*, $U = E(U) \cup V(U)$), the set of edges which succeed is denoted by $S(U)$ (neither incident vertices nor the edges themselves fail, *i.e.*, $S(U) = \{ij \in E(U) : i, j \in V(U)\}$), and $R(U)$ is the residual graph at instantiation U of edges and vertices of matching M.

The basis for the efficiency of algorithms for maximum-weighted matching is that a maximum matching is also *maximal*. Unfortunately, this property does not hold in our context.

Proposition 1. *For obtaining maximum expectation under recourse, the matching proposed at a given round may be non-maximal.*

Proof. By counterexample. Consider the graph C_4, where the labels on edges correspond to their probability of failure.



Assuming unitary weights on each edge, the expression of the expectation for the weight of the matching $\{\{1, 2\}, \{3, 4\}\}$ is the following:

$$2(1 - p_{12})(1 - p_{34}) + (1 - p_{12})p_{34} + (1 - p_{34})p_{12} + p_{12}p_{34}(2(1 - p_{23})(1 - p_{14}) + (1 - p_{14})p_{23} + (1 - p_{23})p_{14}).$$

The first term concerns the outcome when both of the edges in the matching succeed, the second and third terms are related to a failure of one of these edges, and the third terms, in the lower line, calculates the outcome when the two selected edges fail.

The expression of the expectation for the weight of the non-maximal matching $\{\{1, 2\}\}$ is the following:

$$(1 - p_{12})(1 - (1 - p_{34})) + p_{12}(2(1 - p_{23})(1 - p_{14}) + p_{23}(1 - p_{14}) + p_{14}(1 - p_{23}) + p_{23}p_{14}(1 - p_{34})).$$

Here, as detailed later, the second term (in the lower line) concerns the outcome when the edge in the matching fails, which includes the possibility of still having a matching with 4 vertices (when both $\{2,3\}$ and $\{1,4\}$ succeed).

The difference between the former and the latter expressions is:

$$-p_{12}(1-p_{14})(1-p_{23})(1-p_{34}),$$

which is non-positive. □

Actually, when observations are unlimited, focusing solely on single-edge matchings at each round doesn't compromise the ability to find the optimal solution.

Proposition 2. *With no limit on the number of observations allowed, there is a maximum-expectation solution with one edge chosen per each round.*

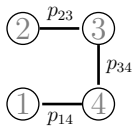
Proof. Any query involving a matching with multiple edges in one round can be broken down into a sequence of queries, each containing just one edge. This offers greater flexibility, as the selection in each round can be adapted based on the successes and failures observed in previous rounds. □

The previous example puts in evidence that the maximum-expectation matching for that graph is one of $\{\{1,2\}\}$, $\{\{2,3\}\}$, $\{\{3,4\}\}$, or $\{\{1,4\}\}$. It becomes also clear that the complete specification of a solution involves the statement of the choices in the initial decision, followed by the choices at the second level (after the first observation), and so on, until the set of edges in the residual graph becomes empty. In this example, it would be (assuming that the optimum is $\{\{1,2\}\}$):

At level 1: the matching $\{\{1,2\}\}$;

At level 2:

- If $\{\{1,2\}\}$ succeeds in the observation: the matching $\{\{3,4\}\}$
- If $\{\{1,2\}\}$ fails in the observation: one must find the maximum-expectation matching in the residual graph

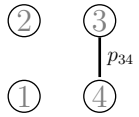


This graph has four possible matchings: $\{\{1,4\},\{2,3\}\}$ (maximal), or one of the non-maximal matchings $\{\{1,4\}\}$, $\{\{2,3\}\}$, or $\{\{3,4\}\}$. It turns out that all of them are equivalent, with expectation

$$p_{14}(1-p_{34}) + (1-p_{14})(1-(1-p_{23})(1-p_{34})).$$

For example, if at level 2 one chooses $\{\{1,4\},\{2,3\}\}$, then level 3 would be

- If both $\{1,4\}$ and $\{2,3\}$ succeed in the observation, then the residual graph will have no edges;
- If $\{1,4\}$ succeeds and $\{2,3\}$ fails in the observation, then the residual graph will have no edges;
- If $\{1,4\}$ fails and $\{2,3\}$ succeeds in the observation, then the residual graph will have no edges;
- If both $\{1,4\}$ and $\{2,3\}$ fail in the observation, then the residual graph will be

  and the obvious choice at level 4 would be the matching $\{\{3,4\}\}$.

If, without loss of generality, one considers proposing one edge per observation, a possible solution for the case of the graph C_4 is shown in Figure 2. A solution for the problem of maximum-expectation matching under recourse is a tree, where each path from the root to a leaf represents a succession of edges proposed and the corresponding observation of its success or failure.

The previous example shows that a complete specification of a solution involves predicting all the relevant scenarios, and recursively finding the optimum for each of them. Notice that if $p_{12} < p_{14} < p_{23} < p_{34}$ the previous tree would be the greedy solution obtained with Algorithm 1. For values of p_{12}, p_{14}, p_{23} near zero and p_{34} close to one, the expectation would approach one, whereas if we started by probing $\{3,4\}$ we would reach an expectation tending towards two.

Proposition 3. *Max-expectation matching is intractable.*

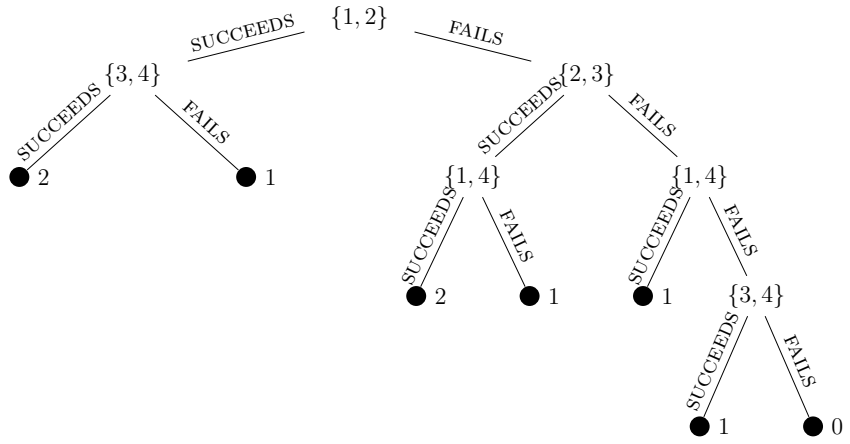


Figure 2: A possible solution for the graph C_4: tree with success/failure alternatives for the edges in the matching proposed.

Proof. Since there is no advantage in choosing multiple edges simultaneously, we assume they are chosen sequentially. The contribution of an edge to the expected value of a matching is not known in advance, as it depends on the outcome of its observation (success/failure) and on possible rearrangements of other edges.

For the sake of illustration, consider a graph with m edges forming a cycle where each edge connects two consecutive nodes, like $\{1, 2\}, \{2, 3\}, \ldots, \{m, 1\}$. In the first level, we need to select one edge among m; we need to define the solution paths for both success and failure of this edge. In the second level, there will be up to $m - 3$ edges remaining. Again, we select one and define solutions for success and failure, leaving at least $m - 6$ edges undecided. We continue this process until there are no undecided edges remaining. This procedure requires making on the order of $2^{m/3}$ decisions.

Therefore, there exist graphs for which we need to specify an exponential number of decisions. This exponential growth in decisions implies the intractability of the problem under this representation. □

Given this property, there is little hope of solving large instances. However, an algorithm for tackling this problem may still be useful for small real-world cases. Besides, limiting the number of observations allowed, as described in the next section, may allow improvements in the actual computational time required. The actual CPU time needed for solving a set of benchmark instances will be analyzed in Section 5.

3.1. Limited recourse

In most situations, there is a limit in the allowed number of observations and recourse reconfigurations; *e.g.*, probing may incur delays, and matchings often have to be completed within a limited timeframe. We call N*-recourse* to a matching problem where the solution must be reached within N observations. If the bound N is one, the problem falls back to standard matching, maximizing expectation based on failure probabilities, but without recourse; $N = \infty$ corresponds to the unlimited case previously described.

The difficulty of solving an N-recourse problem increases with N, being solvable in polynomial time for $N = 1$. Therefore, in terms of complexity, 1-recourse is in P (using, *e.g.*, the algorithm proposed by Edmonds (1965b)), with complexity increasing with N, and ∞-recourse being intractable, as shown before. Notice, however, that even for $N = 1$ evaluating a solution takes exponential time, as all success/failure patterns must be taken into account.

A practical approach to solving this problem consists of obtaining an initial solution for $N = 1$, and then incrementing the number of allowed observations N until the additional gain is considered acceptably low, or until the computational time becomes excessive.

A solution for the problem under limited recourse is no longer a binary tree: on a node one may optimally propose multiple edges, and children of the node must include all the patterns of success or failure for those edges.

4. Algorithm

The basis for the method that we propose is the enumeration of all the matchings in a graph. An interesting algorithm for enumerating all the minimum-cost perfect matchings (where all vertices in the graph are matched) has been proposed in Fukuda & Matsui (1992) for the case of bipartite graphs; to the best of our knowledge, there is no equivalent algorithm for more general cases. Algorithm 3 proposes a very simple recursive procedure for enumerating all the matchings in a graph.

Determining the maximum-expectation matching involves an indirect recursion between the main function `Solve`, presented in Algorithm 4, and function `EvaluateMatching`, presented in Algorithm 5. The former starts by finding the connected components present in the graph; as the expectation for each of them is independent of the others, it may be computed

Algorithm 3: Algorithm for enumerating all matchings.

Data: graph:
- graph $G = (V, E)$ with edges remaining for enumeration (initially, original graph);
- matching M currently under construction (initially empty);
- current set of matchings S (initially empty);

Result:
- list of all matchings in $G = (V, E)$.

```
1  procedure Matchings(V, E, M, S)
2      if E = ∅ then
3          return S
4      ij ← arbitrary edge from E
5      M' ← M ∪ {ij}
6      S ← S ∪ {M'}
7      E' ← {ab ∈ E : {a, b} ∩ {i, j} = ∅}
8      Matchings(V, E', M', S)          // case 1: add ij to current matching
9      Matchings(V, E \ {ij}, M, S)     // case 2: don't add ij to current matching
10     return S
```

separately. Then, each matching in a given component is evaluated with `EvaluateMatching` and the best of them is chosen.

The evaluation of a matching involves listing all the patterns of success or failure of its edges. For each pattern, some bookkeeping is necessary for determining its probability of occurrence; this value is then multiplied by the number of edges for computing the associated contribution to the expectation. However, edges which did not fail and which were not involved with success in the current pattern (stored in variable R') are free for a rearrangement; this is the reason why `Solve` is called inside `EvaluateMatching`, for determining their contribution to the expectation under the current pattern.

As an example, consider again graph C_4 as input and $N = 2$. Algorithm 3 will enumerate matchings $\{34\}, \{34, 12\}, \{23\}, \{23, 14\}, \{14\}, \{12\}$[2], in arbitrary (assume this) order. Each of them will be evaluated, and the best will be proposed as the solution at level 1. Procedure `Solve` in Algorithm 4 will call (line 9) `EvaluateMatching` (Algorithm 5), in the first iteration with matching $M = \{34\}$ and R containing the initial graph. Then, `EvaluateMatching` will assess all the failure patterns in M — in this case,

[2]Simplifying edge notation for clarity.

Algorithm 4: Algorithm for finding a maximum-expectation matching.

Data:

- instance:
  - set of vertices V;
  - set of edges E;
  - probabilities of failure p_{ij} for edges $ij \in E$;
- limit N of observations allowed.

Result:

- maximum-expectation value.

```
1  procedure Solve(V, E, p, N)
2      z* ← 0
3      foreach C ∈ ConnectedComponents(V, E) do
4          if |C| = 1 then continue                    // no matching with only one vertex
5          (V', E') ← subgraph induced on vertex set C
6          z ← 0
7          foreach M ∈ Matchings(V', E') do
8              R ← E'
9              z' ← EvaluateMatching(V', E', p, M, R, N)
10             if z' ≥ z then
11                 z ← z'
12         z* ← z* + z
13     return z*
```

Algorithm 5: Algorithm for evaluating a matching under recourse.

Data:

- instance:
  - set of vertices V;
  - set of edges E;
  - probabilities of failure p_{ij} for edges $ij \in E$;
- matching M;
- edges R in the residual graph;
- limit N of observations allowed.

Result:

- expectation considering all failure patterns.

```
1  procedure EvaluateMatching(V, E, p, M, R, N)
2      if M, N was previously memoized then return T_MN
3      z ← 0
4      foreach b ∈ {0,1}^|M| do                    // binary patterns of size |M|
5          q ← 1
6          n ← 0
7          R' ← R
8          z' ← 0
9          for k ← 1 to |M| do
10             ij ← k^th edge of matching M
11             if b_k = 0 then                     // edge ij fails in this pattern
12                 q ← q × p_ij
13                 R' ← R' \ {ij};
14             else                                // edge ij succeeds in this pattern
15                 q ← q × (1 − p_ij)
16                 n ← n + 1
17                 R' ← {ab ∈ R' : {a,b} ∩ {i,j} = ∅}
18         if R' ≠ ∅ and N > 1 then
19             z' ← Solve(V, R', p, N − 1)
20         z ← z + q × (n + z')
21     memoize T_MN ← z
22     return z
```

just edge $\{34\}$ failing or succeeding. If the edge fails, then procedure `Solve` will be called with edges $\{12, 14, 23\}$ in the residual graph, which has its own matchings $\{23\}, \{23, 14\}, \{14\}, \{12\}$; each of them will be evaluated by `EvaluateMatching`, where no further recursion to `Solve` in line 19 is allowed, because the limit of rounds was reached ($N = 1$). Going back to the assessment of pattern where edge $\{34\}$ succeeds: this will lead to a call of procedure `Solve` with edge $\{12\}$ in the residual graph (in line 19 of `EvaluateMatching`). This single-edge matching will be evaluated through a call to `EvaluateMatching` (with no further recursion to `Solve`, as we reach the limit of rounds).

The previous algorithms can be extended for considering also the case of failure in vertices. For the sake of readability, we have also omitted the steps necessary to build the solution (*i.e.*, a tree similar to the one presented in Figure 2).

5. Results

Let us start by analyzing what differences may be expected between situations without and with recourse. Figure 3 shows the expectation for the number of matched vertices for a graph with four edges, for varying probability of failure p on edges (considered identical for all edges). The graphs considered, with four vertices, are a cycle (C_4), and the complete graph (K_4). With no recourse, the expectation is $2(1 - p)$ for both graphs. With unlimited recourse, we can observe a considerable improvement on graph C_4, and further improvements on K_4, especially for moderate probability of failure. This first result motivates for the use of recourse.

5.1. Results on general graphs

This section reports results for the application of the algorithm in the worst scenario: unlimited number of observations, on possibly dense graphs. A more realistic experiment is presented in the next section. As expected, the exponential growth of the time required for solving in terms of the number of edges is clearly observed in Figure 4, which plots the CPU time used as a function of the number of edges in the graph, for all graphs with up to six vertices[3]. The points almost overlap, showing that there is virtually no

[3]The algorithms presented were implemented in the Python language, version 3.10. All the results were obtained with PyPy implementation version 7.3.15, on a computer with

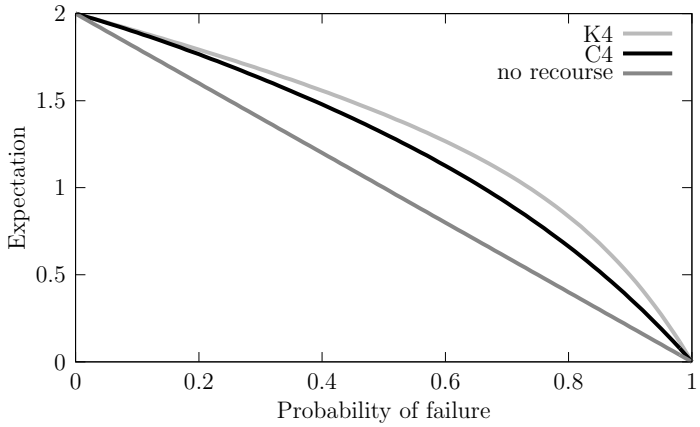


Figure 3: Expectation for the number of matched edges for varying probability of failure on edges.

influence of the number of vertices on the CPU time used; this behavior will no longer be observed in the more realistic cases of next section, where graphs are relatively sparse and have a special structure.

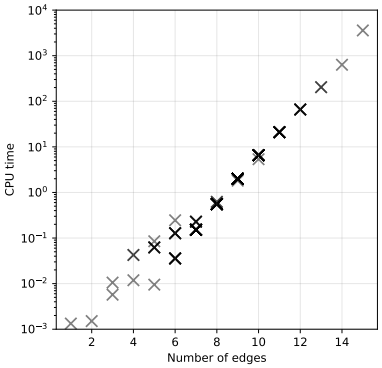


Figure 4: Average CPU time required for solving instances for all graphs up to six vertices, as a function of the number of edges in the graph. (Notice that many points almost overlap.)

an Intel I7 processor at 3.20 GHz, 64 GB of RAM and running macOS version 13.2.1. A prototype implementation is available at https://github.com/joaopedroso/matching-under-recourse.

5.2. Results on an application: kidney exchange

We now turn to the usage of the algorithm proposed on its main application, *i.e.*, in kidney exchange programs, firstly using data provided in Constantino et al. (2013). The graphs based on these data have been randomly generated based on characteristics of real-world kidney exchange pools; probabilities have been randomly drawn with uniform distribution (Pedroso, 2014)[4]. These data are for the more general case of directed graphs $G' = (V, A)$, where exchanges may involve more than two pairs. Undirected graphs $G = (V, E)$ are generated as mentioned previously, with an edge $\{i, j\} \in E$ for each 2-cycle $(i, j) - (j, i)$ in the arc set A. For each edge $\{i, j\} \in E$ we consider its probability of failure as $p_{ij} = 1-(1-p'_i)(1-p'_j)(1-p'_{ij})(1-p'_{ji})$, where p' are the original probabilities for the directed graph.

We analyzed the performance of obtaining the exact solution with Algorithm 4, compared to finding the greedy solution with Algorithm 2. Figure 5 shows the time required for solving each benchmark instance, for a number of pairs in the pool varying from 10 to 50; the actual number of vertices in the graph is typically smaller, after removing isolated vertices. Each point corresponds to a benchmark instance, representing the CPU time required for solving it in terms of the number of vertices and edges in the graph, for a maximum number N of observations ranging from $N = 1$ (no recourse, top) to $N = \infty$ (no limit, bottom). Even though finding a maximum matching in the greedy method, at each observation, is polynomial, evaluating the solution found by either method takes exponential time, even for $N = 1$. However, the time required for finding each maximum matching in Algorithm 2 is negligible. Both algorithms exhibit an exponential increase in runtime with the number of vertices/edges, with the exact solution (Algorithm 4) showing a steeper growth as expected. While Algorithm 4 could employ a polynomial method for optimal matching in the final observation, which would have resulted in completely overlapping points for the two methods in the topmost plots of Figure 5, we opted for a straightforward implementation based on the pseudocode.

Table 1 reports the number of successes, out of 50, for each instance size. As expected, increasing the number of allowed observations leads to a sharp raise on the CPU time required for solving an instance. Note that when only one observation is allowed, the problem can be easily solved for

[4]Instances' data are available at `http://www.dcc.fc.up.pt/~jpp/code/KEP`.

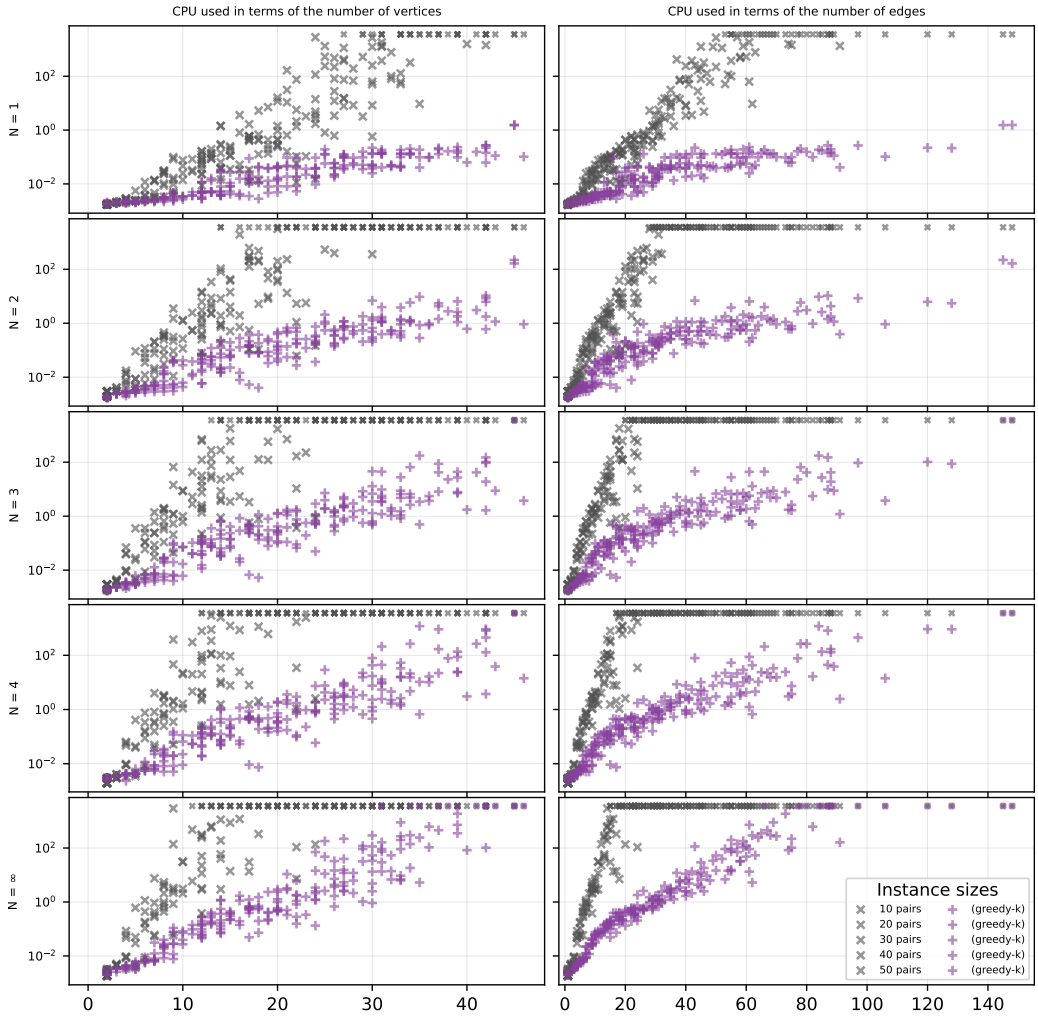


Figure 5: CPU time required for solving realistic KEP instances in terms of the number of vertices (left) and edges (right) in the graph, for a maximum number of observations ranging from 1 (top) to no limit (bottom). Markers $\times$ and $+$ represent exact and greedy solutions, respectively. Smaller dots represent cases with no solution within 3600s.

much larger instances using Edmonds' algorithm or a general-purpose mixed-integer optimization solver; as our implementation of Algorithm 4 relies on enumerating all the matchings, it is inappropriate in this case.

	$N=1$		$N=2$		$N=3$		$N=4$		$N=\infty$	
Instances	E	G	E	G	E	G	E	G	E	G
10 pairs	50	50	50	50	50	50	50	50	50	50
20 pairs	50	50	49	50	40	50	37	50	31	50
30 pairs	50	50	32	50	22	50	18	50	15	50
40 pairs	41	50	15	50	11	50	11	50	9	49
50 pairs	24	50	4	50	2	48	2	48	1	34

Table 1: Number of instances of each size (out of 50) successfully solved in 3600 seconds for exact (E) and greedy (G) methods.

Figure 6 shows the expected number of matched edges for both exact and greedy solutions. In cases where the exact method timed out, we observe isolated points representing the expected performance of the greedy method. For most instances that were solved exactly, the results of both methods were virtually identical, with overlapping points. To facilitate comparison, Table 2 presents the expectations for exact and greedy solution, for varying numbers of allowed rounds N (instances with 10 pairs have 14 trivial cases with no edges in the graph, which were not included). On average, the expected performance of the exact method is identical or only marginally better than that of the greedy method. This table also reports the average and maximum gap of these expectations across all instances; for a given instance, the gap is calculated as $100(1 - g/e)$, where e is the optimum and g is the greedy value. The number of instances with non-zero gaps for each N is shown in the rightmost column. While the gaps tend to decrease with N for $N > 1$, the proportion of instances with non-zero gaps appears to increase. To better

Observations	#	Exact	Greedy	Gap%	Max Gap%	Non-zero/Zero Gaps
N=1	201	0.569	0.569	0.000	0.000	0/201
N=2	136	0.506	0.504	0.257	4.270	40/96
N=3	111	0.476	0.474	0.211	3.748	48/63
N=4	104	0.471	0.470	0.151	1.987	47/57
N=∞	92	0.438	0.437	0.112	1.559	40/52

Table 2: Average expectations and gap statistics for KEP instances.

illustrate the effect of increasing the number of rounds, Table 3 shows the average expected number of matched edges for exact and greedy solutions across a subset of instances of each size. This subset includes only those

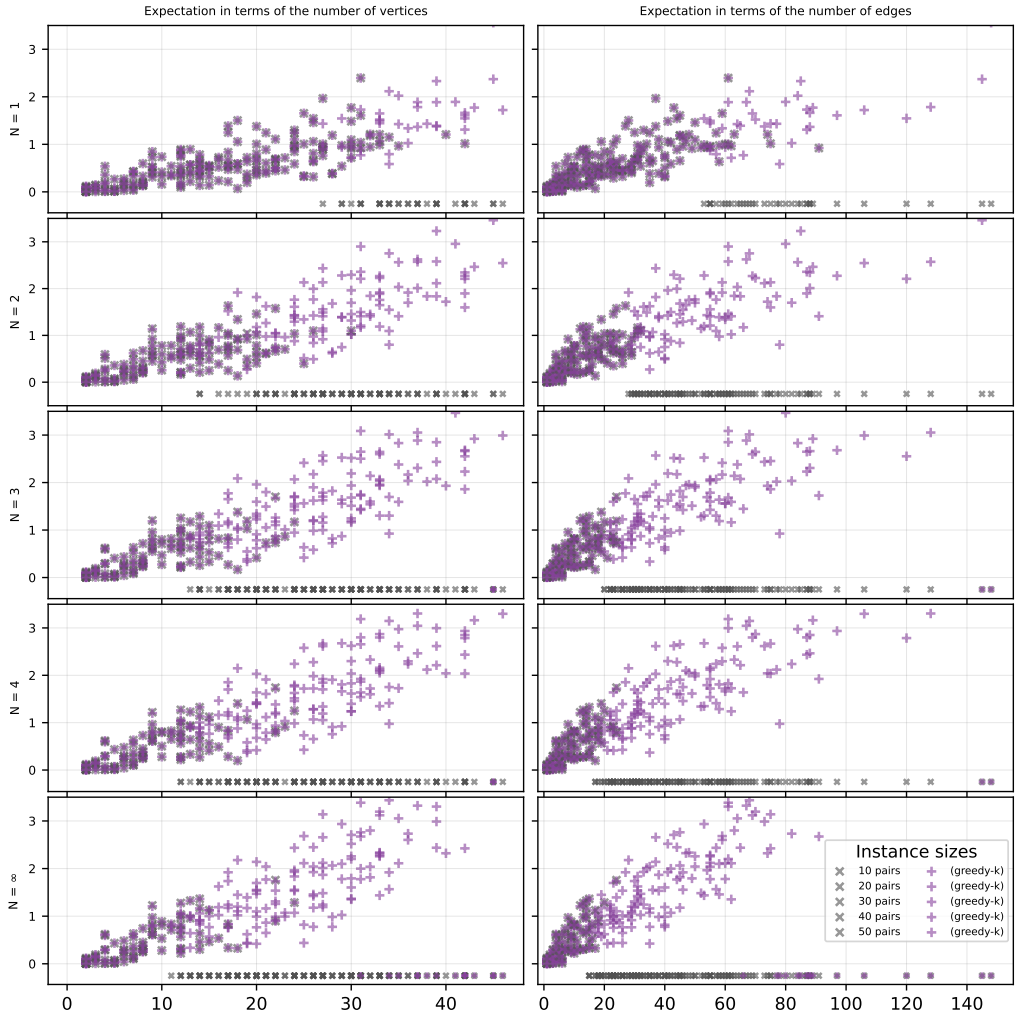


Figure 6: Expected number of edges in the matching for KEP instances in terms of the number of vertices (left) and edges (right) in the graph, for a maximum number of observations ranging from 1 (top) to no limit (bottom). Markers × and + represent exact and greedy solutions, respectively. Smaller dots (below zero) represent cases with no solution within 3600s.

instances solvable by both methods within the time limit for all values of N (the number of such instances is shown in the '#' column). The table presents these averages for various numbers of allowed rounds.

		$N=1$		$N=2$		$N=3$		$N=4$		$N=\infty$	
Instances	#	E	G	E	G	E	G	E	G	E	G
10 pairs	36	0.135	0.135	0.183	0.183	0.197	0.196	0.200	0.200	0.202	0.201
20 pairs	31	0.360	0.360	0.471	0.470	0.512	0.512	0.532	0.531	0.550	0.549
30 pairs	15	0.306	0.306	0.408	0.406	0.446	0.444	0.464	0.462	0.483	0.480
40 pairs	9	0.518	0.518	0.708	0.706	0.794	0.790	0.825	0.822	0.850	0.846
50 pairs	1	0.530	0.530	0.907	0.907	1.100	1.097	1.125	1.117	1.125	1.122

Table 3: Average expected number of edges in the matching for exact (E) and greedy (G) solution, for the subset of non-trivial instances (with at least one edge) solved in less than 3600 seconds for all values of N.

A more recent generator for KEP was proposed by Delorme et al. (2022), aiming to produce more realistic instances. We again analyzed the performance of exact solution of Algorithm 4, comparing it to greedy solution of Algorithm 2, on instances used by Blom et al. (2024) (as before, failure probabilities on vertices and arcs were drawn with uniform distribution, which leads to rather low values of the expected number of edges in the matchings). All 30 instances with 50 vertices were solved by both algorithms within 3600 seconds; for instances with 100 vertices, neither algorithm could solve most cases for $N > 1$. Table 4 shows the average expected number of matched edges across all instances for various numbers of allowed rounds, as well as statistics related to the performance gaps. As before, the gaps are very small and tend to decrease with N for $N > 1$. The proportion of these instances with non-zero gaps is also small and decreases with N.

Observations	Exact	Greedy	Gap% × 1000	Max Gap%	Non-zero/Zero Gaps
N=1	0.275	0.275	0.000	0.000	0/30
N=2	0.328	0.328	6.617	0.127	3/27
N=3	0.337	0.337	6.045	0.111	6/24
N=4	0.342	0.342	4.821	0.066	7/23
N=∞	0.349	0.349	5.030	0.050	8/22

Table 4: Average expectations and gap statistics for Delorme's 50-node instances.

6. Conclusions

Dealing with unreliability is an issue with great importance in many applications involving optimization in graphs. In this paper we study the problem

of maximizing the size of the expected matching under uncertainty, in a situation where vertices or edges may fail and recourse actions can be used to address these failures. The setting is similar to that proposed by Chen et al. (2012), where a greedy algorithm proceeds in rounds; in each round, a greedy matching is proposed on the set of edges available after commitments or failures in previous rounds, with a predetermined maximum number of rounds. Our model is also based on successive observations of failures on proposed matchings, and on recourse solutions being proposed on the remaining graph; it provides an optimal solution for this case. While we focus on the unweighted case here, the model can be easily extended to scenarios where weights are assigned to vertices and/or edges.

The problem has been shown to be intractable, but small instances with practical relevance could be solved with our prototype implementation. Medium to large instances could not be solved with the current implementation; however, there is room for improvement, and small enhancements will have direct impact on the applicability of the method.

As for the practical application of the method, one of the current limitations concerns the availability of data. Indeed, to our knowledge, currently there are no available data on probabilities, and their determination is not trivial; it will likely involve the usage of machine learning tools on related historical data, collected in running exchange programs. This implies that presently we cannot assess our model with real instances. However, the results suggest that the exact solution offers a greater (though small) advantage over the greedy solution with a small, positive number of rounds, which requires fewer resources than a larger number.

Notice that the memory requirements of the implementation provided would make the solution process unthinkable only a few years ago. Hopefully, within some years the evolution in hardware and software will allow tackling larger instances.

There are several interesting future directions for research in this field. The development of approximation algorithms is a requirement for tackling practical applications of moderate size. Since the size of the solution tree can grow exponentially as instance size increases, a key question is whether using heuristics to assess a rule, rather than evaluating the entire solution tree, could provide useful approximations of the true expectation. An interesting variant to investigate is the case where the number of edges that can be probed in each round is limited to a given number. Considering vertex failure, in addition to edge failure, while conceptually simple, poses a real challenge,

due to increasing the number of possibilities on components that may fail. Besides these extensions, there are many challenges that must be overcome for being able to solve larger instances, both on the improvement of the method and on its computational implementation.

Acknowledgements. We would like to thank Dr. Margarida Carvalho and Dr. Xenia Klimentova for their valuable comments on a previous version of this paper. We are indebted to three anonymous referees for their insightful suggestions, which significantly improved this paper. We are also grateful to The Institute of Statistical Mathematics for the support provided.

References

Anderson, R., Ashlagi, I., Gamarnik, D., & Roth, A. E. (2015). Finding long chains in kidney exchange using the traveling salesman problem. *Proceedings of the National Academy of Sciences*, 112(3), 663–668. https://doi.org/10.1073/pnas.1421853112

Bansal, N., Gupta, A., Li, J., Mestre, J., Nagarajan, V., & Rudra, A. (2012). When LP is the cure for your matching woes: Improved bounds for stochastic matchings. *Algorithmica*, 63(4), 733–762.

Berge, C. (1957). Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9), 842.

Biró, P., Haase-Kromwijk, B., Andersson, T., Ásgeirsson, E. I., Baltesová, T., Boletis, I., Bolotinha, C., Bond, G., Böhmig, G., Burnapp, L., et al. (2019). Building kidney exchange programmes in Europe——an overview of exchange practice and activities. *Transplantation*, 103(7), 1514.

Biro, P., Manlove, D., & Rizzi, R. (2009). Maximum weight cycle packing in directed graphs, wiht application to kidney exchange programs. *Discrete Mathematics, Algorithms and Applications*, 1(4), 499–517.

Biró, P., Van de Klundert, J., Manlove, D., Pettersson, W., Andersson, T., Burnapp, L., Chromy, P., Delgado, P., Dworczak, P., Haase, B., et al. (2021). Modelling and optimisation in European kidney exchange programmes. *European Journal of Operational Research*, 291(2), 447–456.

Blom, D., Hojny, C., & Smeulders, B. (2024). Cutting plane approaches for the robust kidney exchange problem. *Computers & Operations Research*, 162, 106470. https://doi.org/https://doi.org/10.1016/j.cor.2023.106470

Blum, A., Dickerson, J. P., Haghtalab, N., Procaccia, A. D., Sandholm, T., & Sharma, A. (2020). Ignorance is almost bliss: Near-optimal stochastic matching with few queries. *Operations research*, 68(1), 16–34.

Carvalho, M., Klimentova, X., Glorie, K., Viana, A., & Constantino, M. (2021). Robust models for the kidney exchange problem. *INFORMS Journal on Computing*, 33(3), 861–881.

Chen, N., Immorlica, N., Karlin, A. R., Mahdian, M., & Rudra, A. (2009). Approximating matches made in heaven. *International Colloquium on Automata, Languages, and Programming*, 266–278.

Chen, Y., Li, Y., Kalbfleisch, J. D., Zhou, Y., Leichtman, A., & Song, P. X.-K. (2012). Graph-based optimization algorithm and software on kidney exchanges. *IEEE Trans. Biomed. Engineering*, 59(7), 1985–1991.

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

de Klerk, M., Keizer, K., Claas, F., Haase-Kromwijk, B., & Weimar, W. (2005). The Dutch national living donor kidney exchange program. *American Journal of Transplantation*, 5, 2302–2305.

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

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

Derakhshan, M. & Farhadi, A. (2023). Beating (1-1/e)-approximation for weighted stochastic matching. *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1931–1961.

Dickerson, J. P., Manlove, D. F., Plaut, B., Sandholm, T., & Trimble, J. (2016). Position-indexed formulations for kidney exchange. *CoRR*, abs/1606.01623. http://arxiv.org/abs/1606.01623

Dickerson, J. P., Procaccia, A. D., & Sandholm, T. (2013). Failure-aware kidney exchange. *Proceedings of the fourteenth ACM conference on Electronic commerce*, 323–340.

Edmonds, J. (1965a). Maximum matching and a polyhedron with 0, 1-vertices. *J. Res. Nat. Bur. Standards B*, 69(1965), 125–130.

Edmonds, J. (1965b). Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3), 449–467.

Fukuda, K. & Matsui, T. (1992). Finding all minimum-cost perfect matchings in bipartite graphs. *Networks*, 22(5), 461–468.

Galil, Z. (1986). Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Surv.*, 18(1), 23–38.

Gamlath, B., Kale, S., & Svensson, O. (2019). Beating greedy for stochastic bipartite matching. *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2841–2854.

Glorie, K. (2012). *Estimating the probability of positive crossmatch after negative virtual crossmatch.* Econometric Institute report, (2012-25).

Goel, G. & Tripathi, P. (2012). Matching with our eyes closed. *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, 718–727.

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

Kong, N. & Schaefer, A. J. (2006). A factor 1/2 approximation algorithm for two-stage stochastic matching problems. *European Journal of Operational Research*, 172(3), 740 – 746.

Li, Y., Kalbfleisch, J., Song, P., Zhou, Y., Leichtman, A., & Rees, M. (2011). Optimization and simulation of an evolving kidney paired donation (KPD)

program. Working Paper Series 90, Department of Biostatistics, University of Michigan. http://www.bepress.com/umichbiostat/paper90.

Li, Y., Song, P. X.-K., Zhou, Y., Leichtman, A. B., Rees, M. A., & Kalbfleisch, J. D. (2014). Optimal decisions for organ exchanges in a kidney paired donation program. *Statistics in biosciences*, 6(1), 85–104.

Manlove, D. F. & O'malley, G. (2015). Paired and altruistic kidney donation in the UK: Algorithms and experimentation. *Journal of Experimental Algorithmics (JEA)*, 19, 2–6.

Molinaro, M. & Ravi, R. (2011). The query-commit problem. *arXiv preprint arXiv:1110.0990*.

Pedroso, J. P. (2014). Maximizing expectation on vertex-disjoint cycle packing. *Computational Science and Its Applications – ICCSA 2014*, volume 8580 of *Lecture Notes in Computer Science*, 32–46. Springer International Publishing.

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

Segev, D., Gentry, S., Warren, D., Reeb, B., & Montgomery, R. (2005). Kidney paired donation and optimizing the use of live donor organs. *The Journal of the American Medical Association*, 293(15), 1883–1890.