Algorithms for Social Good:

Kidney Exchange

Benjamin Plaut

School of Computer Science Honors Undergraduate Research Thesis
Carnegie Mellon University
Advisor: Tuomas Sandholm

Abstract

Kidney exchange is a type of barter market where patients exchange willing but incompatible donors. These exchanges are conducted via cycles—where each incompatible patient-donor pair in the cycle both gives and receives a kidney—and chains, which are started by an altruist donor who does not need a kidney in return. Finding the best combination of cycles and chains is hard. The leading algorithms for this optimization problem use either *branch and price*—a combination of branch and bound and column generation—or *constraint generation*. Although these approaches handle cycles efficiently, they are limited by how they handle chains. Branch-and-price based solvers must perform a process called *pricing*. We begin with a proof of correctness for a polynomial time pricing algorithm for cycles, and a complementary hardness result for pricing chains. Next, we introduce a new integer programming formulation which bypasses this hardness result. On real data from the UNOS nationwide exchange in the United States and the NLDKSS nationwide exchange in the United Kingdom, as well as on generated realistic large-scale data, we show that our new formulation scales significantly better than other solvers—in many cases by orders of magnitude. Finally, we show how our new formulation can be modified in a straightforward way to take post-match edge failure into account, under the restriction that edges have equal probabilities of failure. Post-match edge failure is a primary source of inefficiency in presently-fielded kidney exchanges. We end with a polynomial time algorithm for cycle pricing in the failure-aware context.

Keywords: Kidney exchange, matching markets, integer programming, branch and price

ACKNOWLEDGMENTS

This work was supported in part by NSF grants IIS-1320620 and IIS-1546752, NDSEG, Facebook, and Siebel fellowships, and by XSEDE computing resources provided by the Pittsburgh Supercomputing Center, and by EPSRC grants EP/K010042/1, EP/K503903/1 and EP/N508792/1.

Parts of this work also appeared in AAAI 2016 and EC 2016.

I would like to thank Ross Anderson and Kristiaan Glorie for valuable discussions regarding this work and for making available their kidney exchange software for the purposes of conducting our experimental evaluation. I would also like to acknowledge our collaborators from the University of Glasglow, James Trimble and David Manlove.

I am very grateful to my advisor, Tuomas Sandholm, for the guidance he has given throughout this project. Finally, I would like to especially thank John Dickerson, without whose mentorship and support I would not have been able to complete this thesis.

My contribution

This thesis presents several results regarding kidney exchange, leading to the state-of-the-art kidney exchange solver for finite chain caps. I proved correctness for a polynomial time algorithm for pricing cycles, and a complementary hardness result for chains. I then collaborated with several others in the development of a new integer programming formulation for the kidney exchange problem, called PICEF. Next, I adapted PICEF to the branch and price context, and showed through extensive experimentation that this new formulation scales significantly betters than previous solvers. Finally, I considered the failure-aware context, where the possibility of edges failing after algorithmic match but before transplant is taken into account. I developed a provably correct polynomial time algorithm for pricing cycles in this context, allowing PICEF to be fully adapted to the failure-aware setting.

Chronic kidney disease is a worldwide problem affecting, at various levels of severity, tens of millions of people at great societal burden [Neuen et al. 2013] and monetary cost [Saran et al. 2015]. For those with end-stage kidney failure—of which there are over $100{,}000$ in the US alone[1]—the procurement of a new healthy kidney is a life-saving necessity.

Deceased donors fulfill only a fraction of the demand for kidneys; indeed, the imbalance in supply and demand is growing. Living donation, where a willing donor with two healthy kidneys gives one organ to a patient with kidney failure, is also more desirable than deceased donation; grafts sourced in this manner generally last twice as long as cadaveric grafts in the recipient's body [HHS/HRSA/HSB/DOT 2011]. Finding a feasible living donor is difficult due to medical compatibility and other logistical issues. Toward this end, *kidney exchange* [Rapaport 1986; Roth et al. 2004] is a market where patients with willing but incompatible donors swap their paired donors, thus allowing participants to circumvent these compatibility issues.

Traditionally, exchanges took place in cycles, with each participating patient-donor pair both giving and receiving a kidney. Due to motivation reasons, transplants in a cycle must be carried out simultaneously: otherwise a donor could withdraw after their patient received a kidney. This logistical constraint makes only short cycles feasible, and in general only 2-cycles and 3-cyles are allowed.

Chains are a recent innovation, where a donor without a paired patient—known as an *altruist* or *non-directed donor (NDD)*—triggers a sequence of donations without requiring a kidney in return. Although the introduction of chains increased the efficacy of fielded kidney exchanges, it also dramatically raised the empirical computational difficulty of clearing the market in practice. The requirement of simultaneous operations can be relaxed for chains, but unbounded-length chains are not desirable: planned donations can fail before transplant for a variety of reasons, and the failure of single donation causes the rest of that chain to fail, so parallel shorter chains are better in practice.

In this thesis, we address kidney exchange from a computational point of view. Specifically, given a set of incompatible pairs of patients and donors, we are interested in computing the "best" set of feasible kidney trades. Even computing the maximum-cardinality set of cycles and chains is both theoretically and empirically hard to solve [Abraham et al. 2007]. Over the last decade, integer programming-based methods for solving different interpretations of the kidney exchange problem have been devel-

[1]`http://optn.transplant.hrsa.gov`

oped and then used in fielded exchanges. As kidney exchange matures, holes in the expressiveness and scaling capabilities of the current solvers are found, and improvements are made. We are actively involved in this feedback loop with the United Network for Organ Sharing (UNOS) US nationwide kidney exchange, where our algorithms are used to autonomously match real patients and donors.

In many fielded kidney exchanges, an optimal solution is found by using an integer programming (IP) solver to find a set of disjoint cyclic exchanges and chains that maximizes some scoring function. This approach has been tractable so far; Manlove and O'Malley [2014] report that each instance up to October 2014 in the United Kingdom's National Living Donor Kidney Sharing Schemes (NLDKSS)—one of the largest kidney exchange schemes—could be solved in under a second, with similar results using state-of-the-art solvers at other large exchanges in the US [Anderson et al. 2015] and the Netherlands [Glorie et al. 2014]. However, there is an urgent need for faster kidney exchange algorithms, for three reasons:

— Schemes have recently increased chain-length caps, and we expect further increases as more schemes evolve towards using *nonsimultaneous extended altruistic donor (NEAD) chains* [Rees et al. 2009], which can extend across dozens of transplants.
— Opportunities exist for cross-border schemes, which will greatly increase the size of the problem to be solved; indeed, collaborations have already occurred between, for example, the USA and Greece, and between Portugal and Spain.
— The run time for kidney exchange algorithms depends on the problem instance, and is difficult to predict. It is desirable to have improved algorithms to insure against the possibility that future instances will be intractable for current solvers.

With that motivation, this thesis presents a new scalable integer-programming-based approach to optimally clearing large kidney exchange schemes which can comfortably handle chain caps greater than 10.

2. PRELIMINARIES

Given a pool consisting of patient-donor pairs and altruists, we model the kidney exchange problem using a directed graph $G = (V, E)$. The set of vertices V is partitioned into A and P, which represent the altruists and patient-donor pairs, respectively.

For each $u \in A$ and each $v \in P$, E contains the edge (u, v) if and only if altruist u is compatible with patient v. For each $u \in P$ and each $v \in P \setminus \{u\}$, A contains the edge (u, v) if and only if the paired donor[2] of patient u is compatible with patient v. Each edge $e \in A$ has a weight $w_e \in \mathbb{R}^+$ representing the priority that the scheme administrator gives to that transplant. If the objective is to maximize the number of transplants, each edge has unit weight; most fielded exchanges use weights to encode various prioritization schemes and other value judgments about potential transplants.

Since altruists do not have paired patients, each vertex representing an altruist has no incoming edges. Moreover, we assume that no patient is compatible with her own paired donor, and therefore that the graph has no loops. (The IP model introduced in this thesis can trivially be adapted to the case where loops exist by adding a binary variable for each loop and modifying the objective and capacity constraints accordingly.)

We use the term *chain* to refer to a path in the graph initiated by an altruist vertex, and *cycle* to refer to a cycle in the directed graph (which must involve only vertices in P, since vertices in A have no incoming edges). The weight w_c of each cycle or chain c is defined as the sum of its edge weights. The length of a cycle or chain is defined as the number of edges.

Given a maximum cycle size L and a maximum chain length K, the *kidney exchange problem* is an optimization problem in which the objective is to select a vertex-disjoint packing in G of cycles of size up to L and chains of length up to K that has maximum weight. The problem is NP-hard for realistic parameterizations of L and K [Abraham et al. 2007; Biró et al. 2009]. In practice, L is kept low due to the logistical constraint of scheduling all transplants simultaneously. At both the United Network for Organ Sharing (UNOS) US-wide exchange and the UK's NLDKSS, $L = 3$. The chain cap K is typically longer due to chains being executed non-simultaneously; yet, typically $K \neq \infty$ due to potential matches failing before transplantation. This thesis addresses the realistic setting of small cycle cap L and large—but finite—chain cap K.

Figure 1 shows a problem instance with $|A| = 2$ and $|P| = 5$. If each edge has unit weight and $K = L = 3$, then the bold edges show an optimal solution: cycle $((3, 4), (4, 5), (5, 3))$ and chain $((1, 7), (7, 6))$, with a total weight of 5.

[2]In this thesis we assume that each patient has a single paired donor. In practice however a patient may have multiple paired donors; this can be modeled by regarding such a patient u's vertex in G as representing u and all of her paired donors; an edge (u, v) in G will then represent compatibility between at least one of u's donors and patient v.

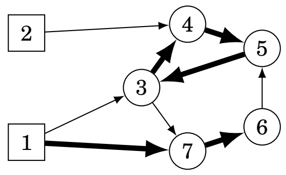

Fig. 1. The directed graph for a kidney-exchange instance with $|A| = 2$ and $|P| = 5$

3. PRIOR RESEARCH

In this section, we overview the two leading approaches to solving integer program (IP) models of the kidney exchange clearing problem.[3] Models solved by *branch and price* use one binary decision variable for each legal cycle and chain, while those solved by *constraint generation* use a combination of binary decision variables representing edges and cycles—but not chains.

3.1. Branch and price

Given a set of vertices $V = P \cup A$, the number of cycles of length at most L is $O(|P|^L)$, the number of uncapped chains is exponential in $|P|$ if $A \neq \emptyset$, and the number of capped chains of length at most K is $O(|A||P|^{K-1})$. Let $\mathcal{C}(L, K)$ represent the set of cycles of length at most L and chains of length at most K. With one decision variable per cycle and chain, an integer program *model* cannot even be *written* to main memory—much less solved—for large enough graphs. Thus, any solver must maintain at most a reduced model (i.e., subset of columns and rows in the constraint matrix) in memory.

Branch and price is a combination of standard branch and bound with column generation that searches for and proves the optimality of a solution to an integer program while maintaining only a reduced model in memory [Barnhart et al. 1998]. For kidney exchange, the idea is as follows [Abraham et al. 2007]. First, start with some relatively small number of, or no, "seed" cycle and chain variables in the model, and solve the linear program (LP) relaxation of this reduced model. Next, generate *positive price* cycles and chains—variables that might improve the solution when brought into the model. For the maximum-weight clearing problem, the price of a cycle or chain c is given by $\sum_{(u,v)\in c} w_{(u,v)} - \sum_{v\in c} \delta_v$, where δ_v is the dual value of vertex v in the LP.

The pricing problem is to generate at least one positive price cycle or chain to bring into the model, or prove that none exist. While any positive price cycles or chains exist at the current node in the branch and bound search tree, optimality has not been proven for the LP. Solving the pricing problem

[3]For an in-depth survey of integer programming approaches to the kidney exchange problem, see Mak-Hau [2015].

can be expensive in its own right, as we discuss in Section 4. Once there are no more positive price cycles or chains, if the LP solution is integral, optimality is proved at that node in the search tree. However, if the LP is fractional, branching occurs. Abraham et al. [2007] branched on individual cycles c, creating one subtree that includes c in the final solution and a second subtree that explicitly does not, and recursing in this way. These branches are then explored in depth-first order until a provably optimal solution is found.

3.2. Constraint generation

Constraint-generation-based approaches to kidney exchange have all variables of the appropriate model in memory from the start, but bring in the constraints of the model incrementally. A basic constraint generation form of the kidney exchange problem uses a decision variable for each edge (i.e., only $O(|V|^2)$ variables) in the compatibility graph and solves a flow problem such that unit flow into a vertex exists if and only if unit flow out of that vertex also exists [Abraham et al. 2007]. This relaxed form of the full problem with only a polynomial number of constraints will not obey cycle or chain caps, so constraints of that form are added until an optimal solution to the relaxed problem is also feasible with respect to cycle and chain caps.

Anderson et al. [2015] built the leading constraint-generation-based IP solver for the kidney exchange problem. Their solver builds on the prize-collecting traveling salesperson problem [Balas 1989], where the problem is to visit each city (patient-donor pair) exactly once, but with the additional option to pay some penalty to skip a city. The solver maintains decision variables for all cycles of length at most L, but build chains in the final solution from decision variables associated with individual edges. Then, an exponential number of constraints is required to prevent the solver from including chains of length greater than K; these are generated incrementally until optimality is proved.

In this thesis, we focus on three prior instantiations of kidney exchange clearing engines: BNP-DFS, the initial branch-and-price-based solver due to Abraham et al. [2007]; CG-TSP, the leading constraint-generation-based approach due to Anderson et al. [2015]; and BNP-POLY, a solver we built based on the methodology of Glorie et al. [2014]. This methodology is described in Section 4.2.

4. THE PRICING PROBLEM

In the branch-and-price approach, solving the pricing problem—that is, finding a positive price cycle or chain or set of cycles/chains, or proving that none exist—is performed at every node in the branch-and-bound search tree. Thus speedups in pricing can result in dramatic overall runtime gains. In this section, we discuss pricing methods for the kidney exchange problem.

4.1. Exponential-time pricing

The first branch-and-price-based IP solver for the kidney exchange problem solved the pricing problem by exhaustively considering all feasible cycles and chains, relative to the current partial solution represented by the search tree [Abraham et al. 2007]. At each node, an exhaustive depth-first-search (DFS) in the compatibility graph computes the price for all cycles until up to a user-specified maximum number of positive cycles are found, or until the search proves that no positive price cycles exist. That proof of nonexistence necessarily sometimes explores all cycles (of capped length) in G which is untenably slow. Indeed, for long chains in pools with many altruistic donors, the pricing problem cripples the BNP-DFS performance, as we show in Section 6.

4.2. Polynomial-time pricing for cycles

We discuss a variant of a polynomial-time pricing algorithm due to Glorie et al. [2014], and provide a proof of correctness. This algorithm is only correct for generating cycles: chains must be handled separately. That is, the algorithm can prove that no positive price cycles exist in a graph, but cannot prove that no positive price chains exist in graph. Indeed, we show in Section 4.3 that (the decision version of) the pricing problem for chains is NP-complete[4].

Glorie et al. [2014] reduce the problem of generating positive price cycles to finding negative weight cycles in a directed graph. They construct a "reduced" graph with the same vertices and edges, but with different weights on the edges. If $e = (u, v)$ is an edge in the original graph with weight w_e, and δ_v is the dual value of vertex v, its weight r_e in the reduced graph is given by $r_e = \delta_v - w_e$. Thus, a cycle is positive price in the original graph if and only if it is a negative cycle in the reduced graph[5].

[4]In [Plaut et al. 2016], we stated that our algorithm is also correct for pricing chains: however, we implicitly assumed that the cycle length cap and chain length cap were equal. Pricing for chains in general is NP-complete.
[5]It is worth noting that the reduction works equivalently if we set $r_e = w_e - \delta_v$, and look for positive cycles in the reduced graph. However, we found it more intuitive to look for lower-weight paths and cycles that higher-weight paths and cycles.

The next step is to efficiently find negative cycles of length at most L. We will use parentheses to denote a path, and angular brackets to denote a cycle. For example, $(v_1, v_2...v_n)$ is a path from vertex v_1 to v_n, while $\langle v_1, v_2...v_n\rangle$ is a cycle containing the above path, plus the edge (v_n, v_1). Glorie et al. [2014] note the following: suppose there is a path $(v_1, v_2, \dots, v_n)$ of reduced weight r_1, and an edge $e = (v_n, v_1)$ with reduced weight r_2. Then if $r_1 + r_2 < 0$, $\langle v_1, v_2, \dots, v_n\rangle$ is a negative cycle. Thus, efficiently finding short paths of length at most L in the reduced graph also finds positive price cycles in the compatibility graph. In this section, we use "short" and "long" to refer to the weight of path, not its edge count.

In general, the shortest path in a graph with negative edge weights is undefined due to the ability to repeat a negative weight cycle multiple times in a single path. Since a path in our context is not valid if it reuses vertices, the problem is well-defined. Yet, finding the shortest path is NP-hard via reduction from the Hamiltonian cycle problem: set all edge weights to -1 and ask if the shortest path from a source u to any neighbor v such that $(v, u) \in E$ is of weight $1 - |V|$. However, the pricing procedure need only find *some*—not necessarily the shortest—negative weight cycle or prove nonexistence.

The Bellman-Ford algorithm[6] is ideally suited to finding cycles of length at most L. As Glorie et al. [2014] note, the ith step of Bellman-Ford computes shortest paths using at most i edges; however, some edges in those paths may be reused by way of reusing negative sub-cycles in the path. To prevent confusion between the kidney exchange cycles and these sub-cycles in the reduced graph, we refer to sub-cycles as "loops."

To prevent looping, before updating the distance to some vertex v via the edge (u, v), we perform an additional check through the predecessors of u. If v already occurs in the path to u, this would create a loop; if this occurs, we do not update the distance to v.

The complexity of this algorithm is $O(|V||E|L^2)$: Bellman-Ford runs from each vertex for $L - 1$ steps and examines $O(|E|)$ edges at each step. Our modification adds an extra factor of L, since on each update, we now have to examine up to $O(L)$ predecessors. This yields an overall complexity of $O(|V||E|L^2)$.

Algorithm 1 provides pseudocode for this method. For a fixed source, let $d_i(v)$ represent the computed distance from that source to v after the ith step of the algorithm, where $d_0(v)$ represents the distances before any steps are performed. Distance is defined as the sum of the edge weights in the computed path. Let L be the maximum allowable cycle length. The function GETNEGATIVECYCLES is called with the reduced graph $G = (V, E)$ the cycle cap L, and the (reduced) edge weight vector w.

[6]Cormen et al. [2009] provide an overview of the Bellman-Ford algorithm.

Algorithm 1 Modified polynomial-time Bellman-Ford search for negative weight cycles.

```
function GetNegativeCycles($G = (V, E), L, w$)
    $\mathcal{C} \leftarrow \emptyset$                                          ▷ Accumulator set for negative weight cycles
    for each $s \in V$ do
        $pred_0(v) = \emptyset \;\; \forall v \in V$
        $d_0(s) = 0$                                              ▷ Distance from source to source is zero
        $d_0(v) = \infty \;\; \forall v \neq s \in V$                         ▷ Distance at step 0 to other vertices is infinite
        for $i \in \{1, \ldots, L-1\}$ do
            $d_i(v) = d_{i-1}(v) \;\; \forall v \neq s \in V$
            $pred_i(v) = pred_{i-1}(v) \;\; \forall v \neq s \in V$
            for each $(u, v) \in E$ do
                if $v \notin$ TraversePreds($u, pred, i-1$) then                    ▷ Avoid loops in path
                    if $d_{i-1}(u) + w(u, v) < d_i(v)$ then         ▷ If this step decreases the distance to node
                        $d_i(v) \leftarrow d_{i-1}(u) + w(u, v)$                    ▷ Update to shorter distance
                        $pred_i(v) \leftarrow (u, i-1)$                           ▷ Store correct predecessor
        for each $v \neq s \in V$ do                   ▷ Find negative weight cycles with $s$ as the source
            if $d_{L-1}(v) + w(v, s) < 0$ then
                $\mathcal{C} \leftarrow \mathcal{C} \cup$ TraversePreds($v, pred, L-1$)
    return $\mathcal{C}$
function TraversePreds($v, pred, n$)
    $c \leftarrow []$                                   ▷ Start with an empty list (representing a cycle)
    $curr \leftarrow v$
    while $curr \neq \emptyset$ do                                      ▷ Until we reach the source node ...
        $c \leftarrow curr + c$                                                  ▷ Add predecessor to path
        $(u, i) \leftarrow pred_n(curr)$                                    ▷ Get predecessor of predecessor
        $curr \leftarrow u; \;\; n \leftarrow i$
    return $c$
```

Although our algorithm is a variant of that of Glorie et al. [2014], they did not prove correctness, which we do now.

THEOREM 4.1. *If there is a negative cycle in the graph, Algorithm 1 will return at least one negative cycle.*

PROOF. We will show that if there is a negative cycle c that we do not find, there must exist a negative cycle with strictly fewer vertices. Thus, for any negative cycle c that we do not return, there must exist a negative cycle p^*q^* with fewer vertices. So, there exists a negative cycle with no negative cycles smaller than it, which our algorithm finds and returns.

Say $c = \langle v_1, v_2, \ldots, v_n \rangle$ is that negative cycle that we do not return. Without loss of generality, assume that c contains the shortest path from v_1 to v_n; if it does not, then that cycle containing the shortest path is also a negative cycle.

Consider running the modified Bellman-Ford method with v_1 as the source. Since by assumption the algorithm does not find c, it must compute a different path from v_1 to v_n than the one in c. We know that the computed path is not shorter, since c contains the shortest path to v_n. Without loss of generality, assume it is strictly longer; were it equal in weight, we would be done (as this is a negative cycle that is found by the algorithm as well).

The only way our modified Bellman-Ford method does not compute the shortest path to v_n is if there exists some vertex v_{split}, where $v_{split} \in c$, but the shortest path to v_{split} is not in c. This can occur due to the modification that prevents loops in shortest paths. Let p be the shorter path from v_1 to v_{split}, and let p_c be the path from v_1 to v_{split} in c. Let q be the path from v_{split} to v_n in c, plus the edge (v_n, v_1). This is shown in Figure 2.

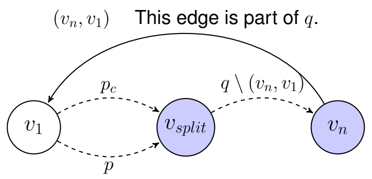

Fig. 2. Widget with a negative cycle and existence of a shorter negative cycle. Dotted arrows are paths that contain zero or more vertices (and thus one or more edges).

Then $c = p_c v_{split} q$. Also, since the weights on the paths are $w(p) < w(p_c)$, we have $w(pq) < w(p_c q) = w(c) < 0$.

We know that $c = p_c q$ satisfies the cycle size cap, since it is valid by assumption. For any path ρ, let $|\rho|$ represent the number of vertices in that path.

CLAIM 4.1.1. $|p| \leq |p_c|$.

Proof: By way of contradiction, assume $|p_c| < |p|$. Then, the sequence of updates along p_c will reach v_{split} before p does—which means we will have computed p_c. Even though we may compute p later, we will still be able to build off of path p_c: this is because we maintain the full 2D predecessor array, which is necessary for other reasons. Therefore we will go on to compute the full $p_c q$, which is a contradiction.

This issue may arise again when computing a path to v_n with p_c as the base; in the process of computing q with p_c as the base, there may exist some vertex v'_{split} that causes the same issue as v_{split}. In that case, our logic can be applied recursively until no such vertex like v'_{split} exists. ■

Using Claim 4.1.1, we can ignore the cycle cap for the rest of the proof, since all cycles discussed will have size $|pq| \leq |p_c q| = |c|$, which is legal by assumption.

At this point, we have p and q such that pq is a *circuit* (i.e., a path that starts and ends at the same vertex but which might not be a cycle because it might visit some vertices more than once), and

$w(pq) < 0$. Claim 4.1.2 gives a tool that we will use to finish the proof of the theorem through repeated use.

CLAIM 4.1.2. *In a directed graph, if there exists a circuit π that is not a cycle and $w(\pi) < 0$, then there exists a circuit π' where $w(\pi') < 0$ and $|\pi'| < |\pi|$.*

Proof: Because π is a circuit but not a cycle, it must have an intersection. Therefore one can split π into two non-empty paths, α and β, where α and β intersect. Thus there exists $v_\cap \in \alpha$ where $v_\cap \in \beta$. Then $\alpha = \alpha_1 v_\cap \alpha_2$ and $\beta = \beta_1 v_\cap \beta_2$, where α_1, α_2, β_1, and β_2 are nonempty.

We know that α_1 is a path from some vertex u to $v_\cap$ and that β_2 is a path from $v_\cap$ back to u. Similarly, β_1 is a path from some vertex u' to $v_\cap$ and α_2 is a path from $v_\cap$ back to u'.

This implies that both $\alpha_1\beta_2$ and $\alpha_2\beta_1$ are circuits. Because $\alpha_1, \alpha_2, \beta_1$, and β_2 are nonempty, $\alpha_1\beta_2$ does not contain u', and $\alpha_2\beta_1$ does not contain u. Therefore $|\alpha_1\beta_2| < |\pi|$ and $|\alpha_2\beta_1| < |\pi|$.

Furthermore, since $w(\pi) = w(\alpha_1\beta_2) + w(\alpha_2\beta_1) < 0$, we know that at least one of $\alpha_1\beta_2$ and $\alpha_2\beta_1$ must be negative. This shows the existence of a negative circuit with strictly fewer vertices. ■

We now return to the proof of the theorem. Recall that we have p and q such that pq is a circuit, and $w(pq) < 0$.

By the claim above, the presence of a negative circuit pq implies that p and q do not intersect, or that there exists a negative circuit $p'q'$ that has fewer vertices. If p and q were not intersecting, pq would be a shorter path than $p_c q$, which violates the assumption that c contains the shortest path. Thus, p and q do intersect. Therefore, there exists a negative circuit $p'q'$ that has fewer vertices.

Since we can only shrink pq, $p'q'$, and so on in this fashion a finite number of times, there must exist some negative circuit p^*q^* where p^*q^* does not self-intersect; so, the negative circuit is a cycle. □

4.3. Hardness of pricing for chains

The pricing problem for chains is to find at least one positive price chain of length at most K, or show that none exist. Recall from Section 3.1 that the price of a chain is $\sum_{(u,v)\in c} w_{(u,v)} - \sum_{v\in c} \delta_v$. Section 4.2 shows how finding a positive price cycle can be reduced to finding a negative weight cycle, by setting $r_{(u,v)} = \delta_v - w_{(u,v)}$.

A similar reduction can be used for chains. We must be careful, however, since the number of vertices in a chain exceeds the number of edges by 1. We now define $r_{(u,v)}$ as follows:

$$r_{(u,v)} = \begin{cases} \delta_v - w_{(u,v)} & u \in P \\ \delta_u + \delta_v - w_{(u,v)} & u \in A \end{cases}$$

Since an outgoing edge from an altruist will only ever be used in a chain, this ensures that a chain is positive price if and only if it is negative weight in the reduced graph.

We define the *negative chain problem* as follows: given a directed graph $G = (V, E)$, where $V = P \cup A$, is there a path (using each vertex at most once) of negative weight, using at most K edges, and starting at some vertex $a \in A$? We call such a path a negative chain.

THEOREM 4.2. *The negative chain problem is NP-complete.*

PROOF. The negative chain problem is trivially in NP: simply sum the edge weights in a proposed path and check its sign. To show NP-hardness, we reduce from the directed Hamiltonian path problem. Given some graph $H = (V, E)$, the directed Hamiltonian path problem asks if there is a directed path visiting each vertex exactly once. Let $n = |V|$. Construct the graph G as follows: set $w_e = -1$ for each $e \in E$, and add a vertex a with an edge (a, v) with $w_{(a,v)} = n - 2$ for each $v \in V$. Figure 3 gives an example graph G. Let $P = V$, $A = \{a\}$, and $K = n$.

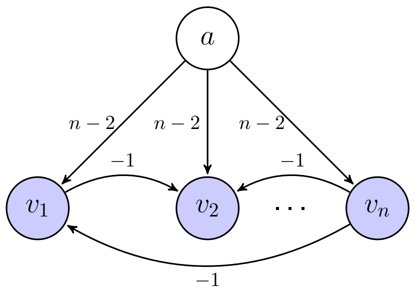

Fig. 3. Example construction for the proof of Theorem 4.2.

Suppose h is a Hamiltonian path in H starting at v_i. Let $c = (a, v_i) \cup h$. Since h has exactly $n-1$ edges, c contains n edges, satisfying the length constraint. Since h visits each v_i exactly once and never visits a, c visits each vertex in G at most once. Finally, since h has weight $1-n$, c has weight $n-2+1-n = -1$. Therefore c is a negative chain is G.

Suppose c is a negative chain in G. Then c must begin at a, so we can write $c = (a, v_i) \cup h$, for some $v_i \in V$ and path h. Let m be the number of edges in h. Then $w_c = n - 2 - m$. Since $w_c < 0$, we have $m > n - 2$. Since c can use each vertex at most once, we have $m \leq n - 1$. Therefore $m = n - 1$. Since h has $n - 1$ edges, h visits every vertex in V exactly once, making it a valid Hamiltonian path in H. $\square$

The general pricing problem (where both cycles and chains are included) is to find a positive price (negative weight) cycle of length at most L or chain of length at most K, or show that none exist. One may note that solving the general pricing problem does not necessarily solve the negative chain problem. If X is the set of negative chains and Y is the set of negative cycles, the general pricing problem is to determine whether $X \cup Y = \emptyset$. The negative chain problem is to determine whether $X = \emptyset$: however, determining whether $X \cup Y = \emptyset$ does not necessarily determine whether $X = \emptyset$.

To show that the general pricing problem is NP-hard, we modify the above construction by expanding each edge in H to a series of $max(L, 1)$ edges whose weights sum to -1. Then any cycle in G has length at least $2L$, which violates the length constraint for $L \geq 2$. For $L < 2$, there are no valid negative cycles regardless. Since there are no valid negative cycles in G, the general pricing problem becomes equivalent to the negative chain problem. Finally, we set $K = n \cdot max(L, 1)$ to ensure that any chain satisfying the length cap in the original construction remains valid. Therefore, the general pricing problem is also NP-hard. Since the general pricing problem is also trivially in NP, it is NP-complete.

5. PICEF: POSITION-INDEXED CHAIN-EDGE FORMULATION

5.1. Description of the model

We now present a new IP formulation: the position-indexed chain-edge formulation, or PICEF. PICEF uses a binary decision variable for each cycle, but edge variables for chains. The idea of using variables for edges in chains and a variable for each cycle was introduced in the PC-TSP-based algorithm of Anderson et al. [2015]. The innovation in our IP model is the use of position indices on edge variables, which results in polynomial numbers of constraints and edge-variables; this is in contrast to the exponential number of constraints in the PC-TSP-based model.

We define $\mathcal{K}(u, v)$, the set of possible positions at which edge (u, v) may occur in a chain in the graph G. For $u, v \in V$ such that $(u, v) \in E$,

$$\mathcal{K}(u,v) = \begin{cases} \{1\} & u \in A \\ \{2,\dots,K\} & v \in P \end{cases}.$$

Thus, any edge leaving an altruist can only be in position 1 of a chain, and any edge leaving a patient vertex may be in any position up to the cycle-length cap K, except 1.

For each $(u,v) \in E$ and each $k \in \mathcal{K}(u,v)$, create variable y_{uvk}, which takes value 1 if and only if edge (u,v) is selected at position k of some chain. For each cycle c in G of length up to L, define a binary variable z_c to indicate whether c is used in a packing.

For example, consider the instance in Figure 4, in which $|A| = 2$ and $|P| = 4$. Suppose that $L = 3$ and $K = 4$, and suppose further that each edge has unit weight. The IP model includes variables y_{131}, y_{141}, and y_{241}, corresponding to edges leaving altruists. For each $k \in 2,3,4$, the model includes variables y_{34k}, y_{45k}, y_{56k}, y_{64k}, and y_{65k}, corresponding to edges between donor-patient pairs at position k of a chain. Finally, the model includes z_c variables for the cycles $((4,5),(5,6),(6,4))$ and $((5,6),(6,5))$.

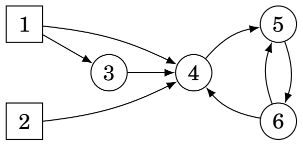

Fig. 4. An instance with $|A| = 2$ and $|P| = 4$

The following integer program is solved to find a maximum-weight packing of cycles and chains.

$$\max \quad \sum_{(u,v)\in E} \sum_{k\in\mathcal{K}(u,v)} w_{(u,v)} y_{uvk} + \sum_{c\in\mathcal{C}} w_c z_c \tag{1a}$$

$$\text{s.t.} \quad \sum_{v:(v,u)\in A} \sum_{k\in\mathcal{K}(v,u)} y_{vuk} + \sum_{c\in\mathcal{C}:u \text{ appears in } c} z_c \le 1 \qquad u \in P \tag{1b}$$

$$\sum_{v:(u,v)\in A} y_{uv1} \le 1 \qquad u \in A \tag{1c}$$

$$\sum_{\substack{v:(v,u)\in E\wedge\\ k\in\mathcal{K}(v,u)}} y_{vuk} \ge \sum_{v:(u,v)\in E} y_{u,v,k+1} \qquad u \in P, \quad k \in \{1,\dots,K-1\} \tag{1d}$$

$$y_{uvk} \in \{0,1\} \qquad (u,v) \in E, k \in \mathcal{K}(u,v) \tag{1e}$$

$$z_c \in \{0,1\} \qquad c \in \mathcal{C} \tag{1f}$$

Inequality (1b) is the capacity constraint for patients: each patient vertex is involved in at most one chosen cycle or incoming edge of a chain. Inequality (1c) is the capacity constraint for altruists: each altruist vertex is involved in at most one chosen outgoing edge. The flow inequality (1d) ensures that patient-donor pair vertex u has an outgoing edge at position $k+1$ of a selected chain only if u has an incoming edge at position k; we use an inequality rather than an equality since the final vertex of a chain will have an incoming edge but no outgoing edge.

We now give an example of each of the inequalities (1b–1d) for the instance in Figure 4. For $u = 4$, the capacity constraint (1b) ensures that $y_{141} + y_{241} + y_{342} + y_{343} + y_{344} + z_{((4,5),(5,6),(6,4))} \leq 1$. For $u = 1$, the altruist capacity constraint (1c) ensures that $y_{131} + y_{141} \leq 1$. For $u = 5$ and $k = 2$, the chain flow constraint (1d) ensures that $z_{452} + z_{652} \geq z_{563}$; that is, the outgoing edge $(5, 6)$ can only be selected at position 3 of a chain if an incoming edge to vertex 5 is selected at position 2 of a chain.

In our example in Figure 4, the optimal objective value is 4. One satisfying assignment that gives this objective value is $y_{131} = y_{342} = z_{((5,6),(6,5))} = 1$, with all other variables equal to zero.

5.2. A branch and price adaptation of the PICEF model

For high enough cycle caps or large enough graphs, it is infeasible to enumerate all cycles, as discussed in Section 3.1. To solve this, we propose a branch and price adaptation of the PICEF model, where only a subset of cycle variables are kept in the model. Promising cycle variables are incrementally added until correctness can be proven at each node in a branch-and-bound search tree. Our method uses the polynomial pricing algorithm from Section 4.2 for pricing cycles.

A great advantage of PICEF is that pricing is not an issue for chains: since the number of edge variables is polynomial in the graph size and K, all edge variables can be enumerated, even for large graphs. In this way, PICEF bypasses the hardness result of Section 4.3.

6. EXPERIMENTAL COMPARISON OF STATE-OF-THE-ART KIDNEY EXCHANGE SOLVERS

In this section, we compare implementations of our new models against existing state of the art kidney exchange solvers. To ensure a fair comparison, we received code from the author of each solver that is not introduced in this thesis.

We compare run times of the following state-of-the-art solvers:

— BnP-DFS, the original branch-and-price-based cycle formulation solver due to Abraham et al. [2007];

— BnP-Poly, a branch-and-price-based cycle formulation solver with pricing due to Glorie et al. [2014];

— CG-TSP, a recent IP formulation based on a model for the prize-collecting traveling salesman problem, with constraint generation [Anderson et al. 2015];

— PICEF, the model from Section 5 of this thesis;

— BnP-PICEF, a branch and price version of the PICEF model, as presented in Section 5.2 of this thesis; and

— HPIEF, the Hybrid PIEF model: a model similar to PICEF, developed by our collaborators.

A cycle length cap of 3 was used for all runs, and a time limit of 3600 seconds was imposed on each run. When a timeout occurred, we counted the run time as 3600 seconds.

We tested on two types of data: real and generated. Section 6.1 shows run time results on *real* match runs, including 286 runs from the UNOS US-wide exchange, which now contains 143 transplant centers, as well as 17 runs from the NLDKSS UK-wide exchange, which uses 24 transplant centers. Following this, Section 6.2 increases the size and varies other traits of the compatibility graphs via a realistic generator seeded by the real UNOS data. We find that PICEF and HPIEF substantially outperform all other models.

6.1. Real match runs from the UK- and US-wide exchanges

We now present results on real match run data from two fielded nationwide kidney exchanges: The United Network for Organ Sharing (UNOS) US-wide kidney exchange where the decisions are made by algorithms and software from Prof. Sandholm's group, and the UK kidney exchange (NLDKSS) where the decisions are made by algorithms and software from Dr. Manlove's group.[7] The UNOS instances used include all the match runs starting from the beginning of the exchange in October 2010 to January 2016. The exchange has grown significantly during that time and chains have been incorporated. The match cadence has increased from once a month to twice a week; that keeps the number of altruists relatively small. On average, these instances have $|A| = 2$, $|P| = 231$, and $|E| = 5021$. The

[7]Due to privacy constraints on sharing real healthcare data, the UNOS and NLDKSS experimental runs were necessarily performed on different computers—one in the US and one in the UK. All runs *within* a figure were performed on the same machine, so relative comparisons of solvers within a figure are accurate.

NLDKSS instances cover the 17 quarterly match runs during the period January 2012-January 2016. On average, these instances have $|A| = 7$, $|P| = 201$, and $|E| = 3272$.

Figure 5 shows mean run times across all match runs for both exchanges. Immediately obvious is that BNP-DFS and CG-TSP tend to scale poorly compared to the newer formulations. Interestingly, BNP-PICEF tends to perform worse than the base PICEF and HPIEF; we hypothesize that this is because branch-and-price-based methods are necessarily more "heavyweight" than standard IP techniques, and the small size of presently-fielded kidney exchange pools may not yet warrant this more advanced technique. Perhaps most critically, both PICEF and HPIEF clear real match runs in both exchanges within seconds.

In the NLDKSS results, the wide fluctuation in mean run time as the chain cap is varied can be explained by the small sample size of available NLDKSS instances, and the fact that the algorithms other than HPIEF and PICEF occasionally timed out at one hour. By contrast, each of the HPIEF and PICEF runs on NLDKSS instances took less than five seconds to complete. We also note that the LP relaxation of PICEF and HPIEF are very tight in practice; the LPR bound equaled the IP optimum for 614 of the 663 runs carried out on NLDKSS data.

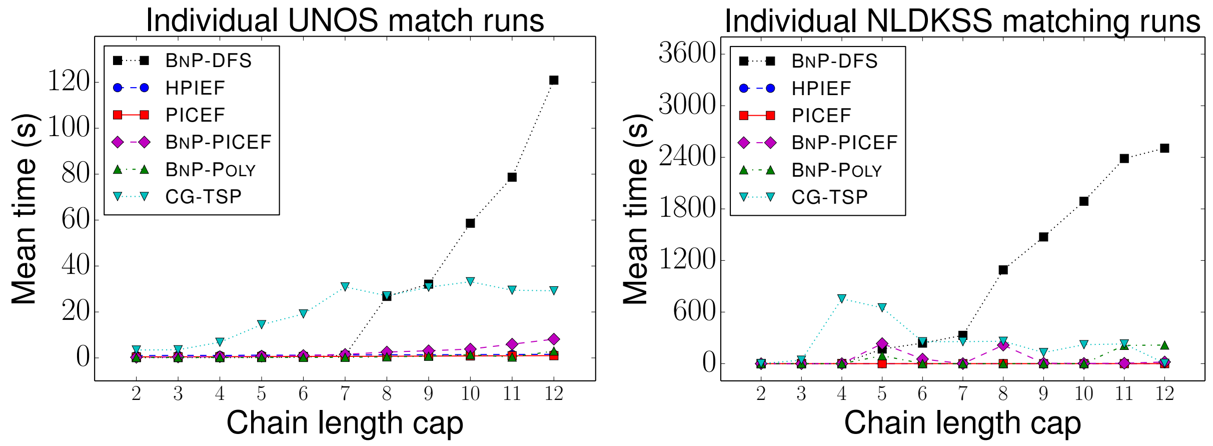

Fig. 5. Mean run times for various solvers on 286 real match runs from the UNOS exchange (left), and 17 real match runs from the UK NLDKSS exchange (right).

6.2. Scaling experiments on realistic generated UNOS kidney exchange graphs

As motivated earlier in the thesis, it is expected that kidney exchange pools will grow in size as (a) the idea of kidney exchange becomes more commonplace, and barriers to entry continue to drop, as well as (b) organized large-scale international exchanges manifest. Toward that end, in this section, we test solvers on generated compatibility graphs from a realistic simulator seeded by all historical UNOS data; the generator samples patient-donor pairs and altruists with replacement, and draws edges in the compatibility graph in accordance with UNOS' internal edge creation rules.

Figure 6 gives results for increasing numbers of patient-donor pairs (each column), as well as increasing numbers of altruists as a percentage of the number of patient-donor pairs (each row). As expected, as the number of patient-donor pairs increases, so too do run times for all solvers. Still, in each of the experiments, for each chain cap, both PICEF and HPIEF are on par or (typically) much faster—sometimes by orders of magnitude compared to other solvers.

In addition to its increased scalability, we note an additional benefit of the PICEF and HPIEF models: reduced variance in run time. In both the real and simulated experimental results, we found that the run time of both the PICEF and HPIEF formulations was substantially less variable than other solvers. While the underlying problem being solved is NP-hard, and thus will always present worst-case instances that take substantially longer than is typical to solve, the increased predictability of the run time of these models relative to other state-of-the-art solutions—including those that are presently fielded—is attractive.

7. FAILURE-AWARE KIDNEY EXCHANGE

Real-world exchanges all suffer to varying degrees from "last-minute" failures, where an algorithmic match or set of matches fails to move to transplantation. This can occur for a variety of reasons, including more extensive medical testing performed before a surgery, a patient or donor becoming too sick to participate, or a patient receiving an organ from another exchange or from the deceased donor waiting list; Leishman et al. [2013] gives a more complete overview of reasons for these failures.

To address these post-match edge failures, Dickerson et al. [2013] augmented the standard model of kidney exchange to include a success probability p for each edge in the graph. They showed how to solve this model using branch and price, where the pricing problem is solved in time exponential in the chain and cycle cap. Prior formulations like those due to Abraham et al. [2007], Constantino et al. [2013], and Anderson et al. [2015] are not expressive enough to allow for generalization to this model. Intuitively, while a single edge failure prevents an entire cycle from executing, chains are capable of incremental execution, yielding utility from the altruist to the first edge failure. Thus, the expected

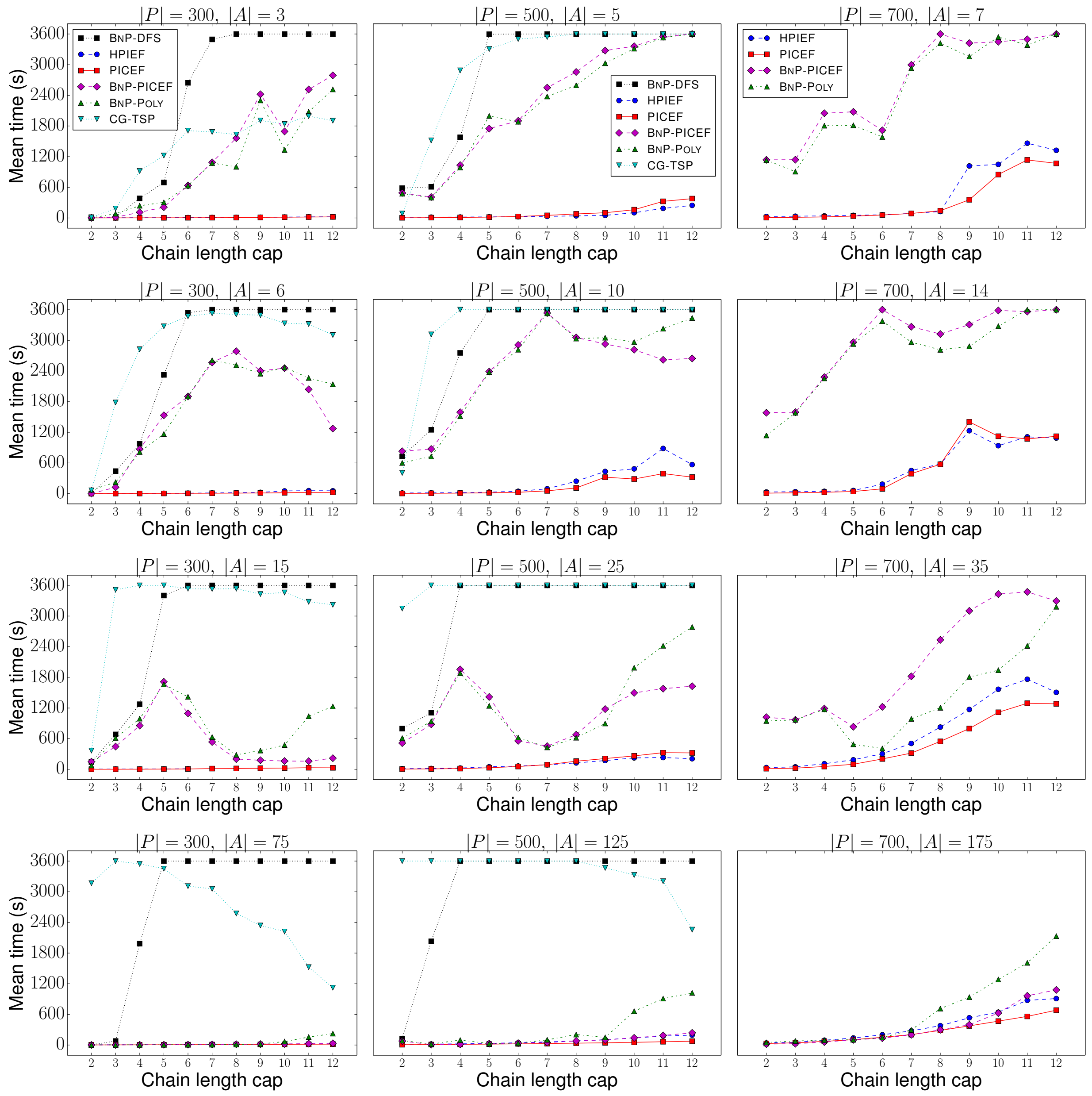

Fig. 6. Mean run time as the number of patient-donor pairs $|P| \in \{300, 500, 700\}$ increases (left to right), as the percentage of altruists in the pool increases $|A| = \{1\%, 2\%, 5\%, 25\%\}$ of $|P|$ (top to bottom), for varying finite chain caps.

utility gained from an edge in a chain is dependent on where in the chain that edge is located, which is not expressed in those models.

PICEF for failure-aware matching. With only minor modification, PICEF allows for implementation of failure-aware kidney exchange, under the restriction that each edge is assumed to succeed with equal probability p. While this assumption of equal probabilities is likely not true in practice, Dickerson et al. [2013] motivated why a fielded implementation of this model would potentially choose to equalize all failure probabilities; namely, so that already-sick patients—who will likely have higher failure rates—are not further marginalized by this model. Thus, given a single success probability p, we can adjust the PICEF objective function to return the maximum expected weight matching as follows:

$$\max \qquad \sum_{(u,v)\in E} \sum_{k\in\mathcal{K}(u,v)} p^k w_{(u,v)} y_{uvk} + \sum_{c\in\mathcal{C}} p^{|c|} w_c z_c \tag{2a}$$

Objective (2a) is split into two parts: the utility received from edges in chains, and the utility received from cycles. For the latter, a cycle c of size $|c|$ has probability $p^{|c|}$ of executing; otherwise, it yields zero utility. For the former, if an edge is used at position k in a chain, then it yields a p^k fraction of its original weight—that is, the probability that the underlying chain will execute at least through its first k edges.

7.1. Failure-aware polynomial pricing for cycles

The initial failure-aware branch-and-price work by Dickerson et al. [2013] generalized the pricing strategy of Abraham et al. [2007], and thus suffered from a pricing problem that ran in time exponential in cycle and chain cap. Section 4.2 discussed a polynomial pricing algorithm for cycles in the *deterministic* case. We present an augmented version of that algorithm which solves the failure-aware, or *discounted*, pricing problem for cycles in polynomial time, under the restriction that all edges have equal success probability p.

The discounted price of a cycle is $p^{|c|}\sum_{(u,v)\in c} w_{(u,v)} - \sum_{v\in c}\delta_v$. We cannot simply use the same reduction from Section 4.2, because the utility of an edge in a cycle now depends on the length of the cycle. Therefore the relative importance of dual values and edge weights depends on the final path length, making it impossible to compare paths in the current Bellman-Ford scheme.

Figure 7 gives an example of this. For consistency with Section 4.2, edge weights in Figure 7 are negative, and the goal is to find a (discounted) negative cycle. Consider running Bellman-Ford with s as the source and $L = 3$. The path (s, v_2, v_3) is preferable to (s, v_1, v_3), since we will end with the 3-cycle $\langle s, v_2, v_3\rangle$ which has weight $p^3(\frac{-\eta}{p^3}) + \eta - 1 = -1$. However, suppose $L = 4$, and we removed the edge (v_3, s). Then $\langle s, v_2, v_3\rangle$ is no longer a cycle, and the path (s, v_1, v_3, v_4) will have weight $p^4(\frac{\eta}{p^3} - 1) + \eta - 1 = \eta - p\eta - p^4 - 1 > 0$, assuming η is large and p is not close to 1. However, the path (s, v_1, v_3, v_4) would

Algorithm 2 Polynomial-time Bellman-Ford failure-aware pricing for cycles.

```
1: function GetDiscountedPositivePriceCycles($G = (V, E), L, p, w, \delta$)
2:     $\mathcal{C} \leftarrow \emptyset$
3:     for each $\ell = 2...L$ do                                   ▷ Consider all possible cycle lengths
4:         $w_\ell(u,v) \leftarrow \delta_v - p^\ell w_{(u,v)}$  $\forall(u,v) \in E$         ▷ For each $\ell$, use reduction from Section 4.2
5:         $\mathcal{C} \leftarrow \mathcal{C} \cup$ GetNegativeCycles($G, \ell, w_\ell$)   ▷ Then the discounted pricing problem is reduced
   to the deterministic pricing problem
6:     return $\mathcal{C}$
```

lead to a discounted negative cycle with weight $-p^4$. The current algorithm cannot compare two paths without knowing the final cycle length.

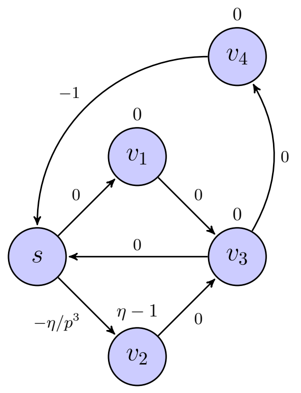

Fig. 7. Example demonstrating that multiple possible final path lengths must be considered.

With this motivation, we augment the algorithm to run $O(L)$ iterations for each source vertex: one for each possible final cycle length. On each iteration, we know exactly how much edge weights will be worth in the final cycle. Therefore we can directly use the reduction from Section 4.2, and reduce the discounted pricing problem to the deterministic pricing problem.

Pseudocode for the failure-aware cycle pricing algorithm is given by Algorithm 2. Let w and δ be the edge weights and dual values respectively in the original graph. The function GetNegativeCycles is defined in the pseudocode for Algorithm 1. Since the reduction is used as a subroutine and not just performed a single time at the beginning of the algorithm, our algorithm is defined as searching for (discounted) positive price cycles, instead of negative cycles.

This augmentation adds a factor of L to the runtime, bringing the overall complexity of the algorithm to $O(|V||E|L^3)$.

Theorem 7.1. *If there is a discounted positive price cycle in the graph, Algorithm 2 will return at least one discounted positive price cycle.*

PROOF. Let $c = \langle v_1, v_2, \ldots, v_n \rangle$ be a discounted positive price cycle. Then $p^n \sum_{(u,v)\in c} w_{(u,v)} - \sum_{v\in c} \delta_v > 0$. Therefore $\sum_{v\in c} \delta_v - p^n \sum_{(u,v)\in c} w_{(u,v)} < 0$. Then by definition of w_ℓ, we have $\sum_{(u,v)\in c}(\delta_v - p^n w_{(u,v)}) = \sum_{(u,v)\in c} w_n(u,v) < 0$.

This implies the existence of a negative cycle in G on the $\ell = n$ iteration of Algorithm 2. By the correctness of GETNEGATIVECYCLES, if there is a negative cycle in the graph, GETNEGATIVECYCLES(G, n, w_n) will return at least one negative cycle of length at most n.

Let c' be a returned cycle. Since c' is negative in G on the $\ell = n$ iteration, we have $\sum_{v\in c'} \delta_v - p^n \sum_{(u,v)\in c'} w_{(u,v)} < 0$. Therefore $p^n \sum_{(u,v)\in c'} w_{(u,v)} - \sum_{v\in c'} \delta_v > 0$.

Since $|c'| \leq n$ by the correctness of GETNEGATIVECYCLES, we have $p^{|c'|} \geq p^n$. Because all edge weights in the original graph are nonnegative, $\sum_{(u,v)\in c'} w_{(u,v)} \geq 0$. Therefore $p^{|c'|} \sum_{(u,v)\in c'} w_{(u,v)} \geq p^n \sum_{(u,v)\in c'} w_{(u,v)}$. Then $p^{|c'|} \sum_{(u,v)\in c'} w_{(u,v)} - \sum_{v\in c'} \delta_v \geq p^n \sum_{(u,v)\in c'} w_{(u,v)} - \sum_{v\in c'} \delta_v > 0$, so c' is indeed discounted positive price.

Therefore Algorithm 2 returns at least one discounted positive price cycle. □

8. CONCLUSIONS & FUTURE RESEARCH

In this thesis, we addressed the optimal clearing of kidney exchanges with short cycles and long, but bounded, chains. This is motivated by kidney exchange practice, where chains are often long but bounded in length due to post-match edge failure. We gave a proof of correctness for a polynomial time pricing algorithm for cycles, and showed that the corresponding pricing problem for chains is NP-complete. We introduced a new IP formulation which bypasses this hardness result. Then, on real data from the UNOS US nationwide exchange and the NLDKSS United Kingdom nationwide exchange, as well as on generated data, we showed that our new model outperforms all other solvers on realistically-parameterized kidney exchange problems–often dramatically. Finally, we showed how our model could be adapted to take post-match edge-failure into account, and provided a polynomial time algorithm for cycle pricing in the failure-aware setting, under the restriction that all edges have equal success probability.

Beyond the immediate importance of more scalable static kidney exchange solvers for use in fielded exchanges, solvers like the ones presented in this thesis are of practical importance in more advanced—and as yet unfielded—approaches to clearing kidney exchange. In reality, patients and donors arrive to and depart from the exchange dynamically over time [Ünver 2010]. Approaches to clearing *dynamic* *kidney exchange* often rely on solving the static problem many times [Awasthi and Sandholm 2009; Dickerson et al. 2012; Anderson 2014; Dickerson and Sandholm 2015; Glorie et al. 2015]; thus, faster static solvers result in better dynamic exchange solutions. Techniques in this thesis—or adaptations thereof—are therefore of interest to dynamic kidney exchange, as well as general barter exchanges.

REFERENCES

David Abraham, Avrim Blum, and Tuomas Sandholm. 2007. Clearing Algorithms for Barter Exchange Markets: Enabling Nationwide Kidney Exchanges. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*. 295–304.

Ross Anderson. 2014. *Stochastic models and data driven simulations for healthcare operations*. Ph.D. Dissertation. Massachusetts Institute of Technology.

Ross Anderson, Itai Ashlagi, David Gamarnik, and Alvin E Roth. 2015. Finding long chains in kidney exchange using the traveling salesman problem. *Proceedings of the National Academy of Sciences* 112, 3 (2015), 663–668.

Pranjal Awasthi and Tuomas Sandholm. 2009. Online Stochastic Optimization in the Large: Application to Kidney Exchange. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*. 405–411.

Egon Balas. 1989. The prize collecting traveling salesman problem. *Networks* 19, 6 (1989), 621–636.

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations Research* 46, 3 (1998), 316–329.

Péter Biró, David F Manlove, and Romeo Rizzi. 2009. Maximum weight cycle packing in directed graphs, with application to kidney exchange programs. *Discrete Mathematics, Algorithms and Applications* 1, 04 (2009), 499–517.

Miguel Constantino, Xenia Klimentova, Ana Viana, and Abdur Rais. 2013. New insights on integer-programming models for the kidney exchange problem. *European Journal of Operational Research* 231, 1 (2013), 57–68.

Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (third ed.). MIT Press, Cambridge, MA.

John P. Dickerson, Ariel D. Procaccia, and Tuomas Sandholm. 2012. Dynamic Matching via Weighted Myopia with Application to Kidney Exchange. In *AAAI Conference on Artificial Intelligence (AAAI)*. 1340–1346.

John P. Dickerson, Ariel D. Procaccia, and Tuomas Sandholm. 2013. Failure-Aware Kidney Exchange. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*. 323–340.

John P. Dickerson and Tuomas Sandholm. 2015. FutureMatch: Combining Human Value Judgments and Machine Learning to Match in Dynamic Environments. In *AAAI Conference on Artificial Intelligence (AAAI)*. 622–628.

Kristiaan Glorie, Margarida Carvalho, Miguel Constantino, Paul Bouman, and Ana Viana. 2015. Robust Models for the Kidney Exchange Problem. (2015). Working paper.

Kristiaan M. Glorie, J. Joris van de Klundert, and Albert P. M. Wagelmans. 2014. Kidney Exchange with Long Chains: An Efficient Pricing Algorithm for Clearing Barter Exchanges with Branch-and-Price. *Manufacturing & Service Operations Management (MSOM)* 16, 4 (2014), 498–512.

HHS/HRSA/HSB/DOT. 2011. OPTN/SRTR Annual Data Report. (2011).

Xenia Klimentova, Filipe Alvelos, and Ana Viana. 2014. A New Branch-and-Price Approach for the Kidney Exchange Problem. In *Computational Science and Its Applications (ICCSA-2014)*. Springer, 237–252.

Ruthanne Leishman, Richard Formica, Kenneth Andreoni, John Friedewald, Elizabeth Sleeman, Catherine Monstello, Darren Stewart, and Tuomas Sandholm. 2013. The Organ Procurement and Transplantation Network (OPTN) Kidney Paired Donation Pilot Program (KPDPP): Review of Current Results. In *American Transplant Congress (ATC)*. Talk abstract.

Vicky Mak-Hau. 2015. On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. *Journal of Combinatorial Optimization* (2015), 1–25.

David Manlove and Gregg O'Malley. 2014. Paired and Altruistic Kidney Donation in the UK: Algorithms and Experimentation. *ACM Journal of Experimental Algorithmics* 19, 1 (2014).

Brendon L Neuen, Georgina E Taylor, Alessandro R Demaio, and Vlado Perkovic. 2013. Global kidney disease. *The Lancet* 382, 9900 (2013), 1243.

Benjamin Plaut, John P. Dickerson, and Tuomas Sandholm. 2016. Fast Optimal Clearing of Capped-Chain Barter Exchanges. In *AAAI Conference on Artificial Intelligence (AAAI)*.

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

Michael Rees, Jonathan Kopke, Ronald Pelletier, Dorry Segev, Matthew Rutter, Alfredo Fabrega, Jeffrey Rogers, Oleh Pankewycz, Janet Hiller, Alvin Roth, Tuomas Sandholm, Utku Ünver, and Robert Montgomery. 2009. A Nonsimultaneous, Extended, Altruistic-Donor Chain. *New England Journal of Medicine* 360, 11 (2009), 1096–1101.

Alvin Roth, Tayfun Sönmez, and Utku Ünver. 2004. Kidney exchange. *Quarterly Journal of Economics* 119, 2 (2004), 457–488.

Rajiv Saran, Yi Li, Bruce Robinson, John Ayanian, Rajesh Balkrishnan, Jennifer Bragg-Gresham, JT Chen, Elizabeth Cope, Debbie Gipson, Kevin He, and others. 2015. US Renal Data System 2014 Annual Data Report: Epidemiology of Kidney Disease in the United States. *American Journal of Kidney Diseases* 65, 6 Suppl 1 (2015), A7.

Utku Ünver. 2010. Dynamic kidney exchange. *Review of Economic Studies* 77, 1 (2010), 372–414.

Appendix to:

Algorithms for Social Good: Kidney Exchange

Benjamin Plaut

A. IMPLEMENTATION DETAILS

A.1. Bellman-Ford pricing is more complicated than normal Bellman-Ford

We now describe how it is necessary in the implementation of the adapted Bellman-Ford method of Algorithm 1 to maintain the entire 2-dimensional predecessor array for vertices in the pricing graph, whereas a 1-dimensional array suffices in typical Bellman-Ford (see, e.g., Cormen et al. [2009]). This difference arises from the fact that we need to limit the number of edges in a path, or else the cycles we generate may exceed the permissible length. If we only use a 1-dimensional predecessor array, running Bellman-Ford for k steps does not guarantee paths of length at most k.

The intuition for this requirement is as follows: say we would like to compute paths of length at most k, so we run k steps of Bellman-Ford. Suppose that after $k-1$ steps, the path to vertex u has $k-1$ edges, and that on the last step, the distance to a neighboring vertex v is updated via vertex u. If nothing else is updated, the path to vertex v would have k edges, which is valid. However, suppose the path to vertex u also gets updated, and that this updated path also contains k edges. Since at this final step vertex v has u as its predecessor (denoted $\texttt{pred}(v) = u$), the path to vertex v is the path to u plus the edge (u, v), so the path to v is now $k+1$ edges long, which is invalid.

It is also not viable to simply exclude those paths that end up with more than k edges, since the algorithm may have forgotten a different path to v that was less promising at the time, but—under the additional constraint that paths of length greater than k are invalid—would have ended up only using k edges. If that other path were to represent the only positive price cycle, Algorithm 1 would mistakenly return that there are no positive price cycles, breaking the correctness of the branch-and-price solver. Figure 8 illustrates such a situation.

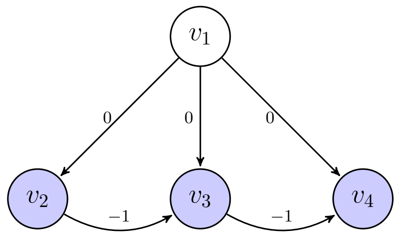

Fig. 8. Example pricing graph where a 2-dimensional predecessor array is used for correctness (when $L = 3$).

In the pricing graph of Figure 8, suppose we used the standard 1-dimensional predecessor array and ran Bellman-Ford for two steps, using vertex v_1 as the source. Then, Figure 9 shows the computed predecessors for each of the three non-source vertices.

Step #	pred(v_2)	pred(v_3)	pred(v_4)
0	–	–	–
1	v_1	v_1	v_1
2	v_1	v_2	v_3

Fig. 9. Predecessor table computed for the graph of Figure 8 with vertex source v, for $k = 2$ steps.

If only the last predecessor array row is examined, the path to vertex v_4 that is extracted by following the pred mapping will be (v_1, v_2, v_3, v_4)—which contains three edges, even though we only ran Bellman-Ford for $k = 2$ steps.

It is even possible to form paths of arbitrary length after two steps. Suppose there also existed $v_4 \dots v_n$. Add an edge (v_1, v_i) with weight 0, and an edge (v_i, v_{i+1}) with weight -1 for all i. Then after two steps, a 1D predecessor array would implicitly hold a path of length n.

We solve this issue as follows. When we update the distance to vertex v by way of neighboring vertex u, we cannot simply replace the new path to vertex v by the path to vertex u plus the edge (u, v), as would be done in typical Bellman-Ford. Instead, the new path to vertex v should be the path to vertex u *at the time of the update* plus the edge (u, v). In the above example, when we update the distance to vertex v_4 on the second step, the path to vertex v_3 is (v_1, v_3), so the overall path to vertex v_4 should be (v_1, v_3, v_4).

To handle this, whenever we make an update we must not only store the predecessor, but also the time of the update (the step number). Then when extracting a path to vertex, we can jump to the predecessor of that vertex at the time of the update. However, this process requires storing the entire 2-dimensional array of all (updated) predecessors on each step.

Note that every time the algorithm jumps to a vertex's predecessor, it moves at least one time step backwards in the 2-dimensional predecessor array. Since the path creation process ends when the time step reaches 0, and there are a total of k steps (rows in the array), any extracted paths are guaranteed to have at most k edges. Also note that this process does not change the sequence of updates in the algorithm; instead, it ensures that the paths extracted in the end accurately reflect the sequence of updates.

Finally, note that when we store paths at the time of update, the final path to a vertex v may contain a vertex u but not the final path to vertex u. This is explains in the proof of Theorem 4.1 how we are able to continue to make updates to p_c, even after p is computed later.

A.2. Edge branching scheme

During the branch-and-bound (and thus also branch-and-price) search, when the LP relaxation at a given node is non-integral, and the upper and lower bounds do not fathom that subtree, a decision variable (or set of decision variables) must be chosen on which to branch. For example, the original branch-and-price-based solver BNP-DFS chooses a single variable x_c corresponding to a cycle or chain c whose value is non-integral (i.e., for a binary variable x_c in the IP, the relaxed value in the LP $x_c \in (0,1)$) to branch on [Abraham et al. 2007]. If there is more than one such non-integral variable, the one with value closest to 0.5 is chosen, and the subtree with $x_c = 1$ is explored first in depth-first order.

As discussed by Glorie et al. [2014], this polynomial pricing algorithm is incompatible with branching on cycles. Consequently, our branch and price implementations branch on edges: when the LP solution at a node is fractional, a non-integral *edge* is chosen to branch on. Glorie et al. [2014] showed how to branch on edge variables, while still in a cycle formulation model. That is the scheme we use in this thesis, as described below.

Let x_c be the relaxed decision variable for cycle c in the LP. Then, the value for any edge e in the compatibility graph is $e = \sum_{c:e\in c} x_c$, the sum of relaxed values for $x_c \in [0,1]$ over all cycles c that contain edge e. It is not immediately clear that the presence of a fractional cycle in the LP implies the presence of a fractional edge, but Glorie et al. [2014] show that this is in fact the case.