Finding long chains in kidney exchange using the traveling salesman problem

Ross Anderson[a], Itai Ashlagi[b], David Gamarnik[b], and Alvin E. Roth[c,1]

[a]Operation Research Center and [b]Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02142; and [c]Department of Economics, Stanford University, Stanford, CA 94305

Contributed by Alvin E. Roth, November 18, 2014 (sent for review July 15, 2014; reviewed by Peter Biro)

As of May 2014 there were more than 100,000 patients on the waiting list for a kidney transplant from a deceased donor. Although the preferred treatment is a kidney transplant, every year there are fewer donors than new patients, so the wait for a transplant continues to grow. To address this shortage, kidney paired donation (KPD) programs allow patients with living but biologically incompatible donors to exchange donors through cycles or chains initiated by altruistic (nondirected) donors, thereby increasing the supply of kidneys in the system. In many KPD programs a centralized algorithm determines which exchanges will take place to maximize the total number of transplants performed. This optimization problem has proven challenging both in theory, because it is NP-hard, and in practice, because the algorithms previously used were unable to optimally search over all long chains. We give two new algorithms that use integer programming to optimally solve this problem, one of which is inspired by the techniques used to solve the traveling salesman problem. These algorithms provide the tools needed to find optimal solutions in practice.

kidney exchange | kidney paired donation | transplantation | algorithms | computation

As of May 2014 there were over 100,000 patients on the waiting list for a kidney transplant from a deceased donor. Many of these patients have a friend or family member willing to be a living kidney donor, but the donor is biologically incompatible. Kidney exchange, also called kidney paired donation (KPD), arose to allow these patients with willing donors (referred to as patient–donor pairs) to exchange kidneys, thus increasing the number of living donor transplants and reducing the size of the waiting list.

In KPD, incompatible patient–donor pairs can exchange kidneys in cycles with other such pairs so that every patient receives a kidney from a compatible donor (Fig. S1). Additionally, there are a small number of altruistic donors who are willing to donate their kidney to any patient without asking for anything in return. In KPD, these donors initiate a chain of transplants with incompatible pairs, ending with a patient on the waiting list that has no associated donor (Fig. S2).

Integrating both cycles and chains in KPD was proposed in ref. 1, allowing both chains and cycles to be of unlimited size. To ensure that every patient receives a kidney before her associated donor donates her kidney, cycles are conducted simultaneously (otherwise, if the intended donor is unable to donate to the patient whose associated donor already gave her kidney the pair not only did not receive a kidney, but also could not participate in future exchanges). Because organizing many surgeries simultaneously is logistically very complex, the first implementations of KPD by the New England Program for Kidney Exchange and other clearinghouses used only two-way cyclic exchanges. After a short period, clearinghouses have moved to allow three-way exchanges as well.

In ref. 2 it was proposed to relax the requirement of simultaneity to the weaker requirement that every patient–donor pair receive a kidney before they give a kidney. Although for cycles this restriction still required all surgeries be performed simultaneously, it did allow for nonsimultaneous chains. Note that these nonsimultaneous chains still protected patient–donor pairs from irreparable harm but allowed for the possibility of donors' backing out after their patient had received a transplant. Since the first nonsimultaneous chain was arranged (3) chain-type exchanges have accounted for a majority of the transplants in kidney exchange clearinghouses. [Approximately 75% of the transplants in the National Kidney Registry (NKR) and the Alliance for Paired Donation (APD) are done through chains.] Chains involving as many as 30 pairs have been performed in practice, capturing significant public interest (4). Very long chains are often planned in segments, in which the donor from the final pair in a segment is used to begin another segment after more patients arrive. Such donors are called "bridge donors." Once the segment is executed, the bridge donors and altruistic donors are essentially identical for the purpose of planning future transplants and are collectively referred to here as "nondirected donors" (NDDs). The problem of determining how to optimally select a set of cycles and chains to maximize the number of transplants performed is the focus of this work.

We refer to the problem of finding the maximum (possibly weighted) number of transplants for a pool of incompatible patient–donor pairs and NDDs as the kidney exchange problem (KEP). Optionally included in the problem are a maximum chain length and a maximum cycle length. Weights can be used to prioritize difficult-to-match patients. An example of a KEP instance is shown in Fig. S3. When there is no bound on the chain or cycle length, the problem can solved efficiently by reducing it to a maximum weighted perfect matching problem on a bipartite graph, as described in ref. 5. The special case in which only cycles

Significance

There are currently more than 100,000 patients on the waiting list in the United States for a kidney transplant from a deceased donor. To address this shortage, kidney exchange programs allow patients with living incompatible donors to exchange donors through cycles and chains initiated by altruistic nondirected donors. To determine which exchanges will take place, kidney exchange programs use algorithms for maximizing the number of transplants under constraints about the size of feasible exchanges. This problem is NP-hard, and algorithms previously used were unable to optimize when chains could be long. We developed two algorithms that use integer programming to solve this problem, one of which is inspired by the traveling salesman, that together can find optimal solutions in practice.

Author contributions: R.A., I.A., D.G., and A.E.R. designed research; R.A. and I.A. performed research; A.E.R. contributed new reagents/analytic tools; R.A., I.A., and D.G. analyzed data; and R.A., I.A., D.G., and A.E.R. wrote the paper.

Reviewers included: P.B., Institute of Economics, Hungarian Academy of Sciences.

The authors declare no conflict of interest.

[1]To whom correspondence should be addressed. Email: alroth@stanford.edu.

This article contains supporting information online at www.pnas.org/lookup/suppl/doi:10.1073/pnas.1421853112/-/DCSupplemental.

of length two are used can be exactly solved very efficiently, because it is trivially equivalent to the maximum matching problem. In ref. 5, the case of this problem where only cycles of length two and three are used was shown to be NP-hard, meaning that it is unlikely there will be an algorithm that always finds the optimal solution quickly in every instance of the problem; see ref. 6 for a stronger negative result in this special case. However, integer programming (IP) techniques have been used by a variety of authors to solve special cases of the KEP without chains or with chains of bounded length, as first proposed in ref. 7. In ref. 5, by improving the IP formulation of ref. 7 and devising good heuristics the authors were able to solve KEP instances with thousands of donor pairs, but without chains. Alternate IP formulations were further explored for this special case in ref. 8. In ref. 1, a heuristic to produce feasible solutions when using chains and cycles of bounded length was suggested. However, no optimization algorithm was given, opening a major algorithmic challenge to reliably solve large-scale instances of the general KEP. The technique of ref. 5 was extended in refs. 9–12 to solve large instances with bounded cycles and bounded chains. However, the algorithm became impractical when the maximum chain length was beyond four or five, because the formulation required a decision variable for each chain up to the maximum chain length. As seen in Table S1, the number of chains of length up to k grows very quickly with k, at least when creating instances using patient–donor pairs drawn at random from the historical dataset (a more thorough explanation is given in *Supporting Information*). When both chains and cycles are bounded by length k, the relaxed version of the IP formulation (as a linear program) provides a good approximation to the original formulation (13, 14), and a large body of literature suggests that such problems can be effectively solvable in practice. However, this approximation property does not seem to hold if chains are unbounded (15), and hence this problem is likely to be much more challenging than the bounded case, as confirmed by our computational experiments.

The primary challenge addressed in this paper is to find an algorithm to solve real instances of the KEP without bounding the maximum chain length. Solving this optimization problem is critical to the operations of KPD programs, which form long chains in practice but previously relied on heuristics that could lead to suboptimal solutions. We emphasize that we focus on real instances drawn from historical data, as opposed to synthetic instances, because the statistics of instances encountered in practice are quite complicated, making it difficult to generate representative random instances (a patient–donor pair waiting to be matched is on average more difficult to match than a patient–donor pair randomly selected from historical data, because the easy-to-match patients are on average matched more quickly). The second challenge we address in this paper is finding an algorithm that can solve instances using chains and cycles with an arbitrary bound on the maximum cycle length (with no maximum cycle length, the problem can be solved trivially). Although in practice cycles longer than three are rarely formed owing to logistical issues, there is some evidence that there may be benefits to considering longer cycles (16). Finally, in *Supporting Information* we demonstrate how our approach can be extended to efficiently deal with the case of bounded chain lengths as well as a stochastic version with "edge failures" (see refs. 17 and 18).

We propose two new algorithms for the KEP that address these challenges. The first algorithm is a recursion based on a direct IP formulation for the KEP. Despite the NP hardness of the underlying problem, the algorithm is surprisingly effective in practice in most instances, as we will show in a later section devoted to the computational results. Additionally, the algorithm does not break down as the maximum cycle length is increased. On the contrary, as the bound on the maximum cycle length increases, the performance improves. At the same time there are several KEP instances of moderate size that have been encountered in practice on real data drawn from a particular KPD program that our algorithm was unable to solve. Thus, we have devised a second algorithm to reliably solve nearly all KEP instances of practical interest. The algorithm is motivated by an IP formulation of the so-called prize-collecting traveling salesman problem (PC-TSP), another NP-hard problem. The PC-TSP is a variant of the classical traveling salesman problem (TSP), one of the most widely studied NP-hard problems in combinatorial optimization. Although our PC-TSP–based algorithm was able to solve every instance we have encountered, somewhat surprisingly we have discovered that it is at times orders of magnitude slower than the direct recursive algorithm described above.

The two algorithms described above allow us to optimally allocate kidneys transplanted in KPD with arbitrary chain lengths. These algorithms were capable of solving to optimality all real data instances we have encountered, using only a single desktop computer. Furthermore, several of the most active KPD programs are using our implementations of these algorithms in conducting their kidney exchanges, including Methodist Specialty and Transplant Hospital in San Antonio, Texas and the Northwestern University Medical Center in Chicago.

The remainder of the paper is organized as follows. First, we formally define the KEP as an optimization problem on a graph. Then, we describe the recursive algorithm and PC-TSP–based algorithm for the KEP. Both algorithms are based on formulating associated IP problems. Next, we demonstrate the value of being able to find long chains by measuring the number of additional transplants they enable using simulations based on historical clinical data. We then compare the performance of our two algorithms in KEP instances found in actual data from the KPD programs, demonstrating the relative strength of the PC-TSP–based algorithm compared with the recursion-based algorithm. To provide some insight into the power of the PC-TSP–based algorithm we state a result showing that the formulation of the IP problem associated with the PC-TSP is stronger than the IP problem corresponding to the recursive algorithm. The proof of this result can be found in *Supporting Information*. Furthermore, we provide an example of a pathological instance of the KEP for which the recursive algorithm takes a very long time, whereas the PC-TSP–based algorithm solves this instance very quickly. Finally, we conclude with a summary of our results. Some technical results can be found in *Supporting Information*.

KEP

An instance of the KEP is described as follows:

- a list of NDDs,
- a list of patient–donor pairs (in which the donor wants to donate to the paired patient but is not compatible with this patient),
- the compatibility information between all donors and patients,
- the "weight," or priority, of each potential transplant, and
- a bound on the maximum cycle length.

The goal is then to find a set of transplants, organized into cycles and chains initiated by NDDs that uses each donor and patient at most once and maximizes the sum of the weights of all transplants performed. If all transplants have weight one, then we are simply trying to find the arrangement that maximizes the total number of transplants. In *Supporting Information* we will show how this definition can be supplemented to include an optional bound on the maximum chain length.

We now formalize this definition of the KEP in graph theoretic terms. We are given a directed graph $G=(V,E)$, a weight vector $\mathbf{w}=(w_1,\ldots,w_{|E|})$, and a nonnegative integer parameter k. The set V is partitioned into sets N (the NDDs) and P (the pairs of incompatible donors and patients). For $u,v\in V$; a directed edge from u to v in E indicates that the donor in node u is compatible with the patient in node v. Because the nodes of N have no patient, they all must have in-degree zero (although there can be nodes in P with in-degree zero as well). The values w_e for each edge $e \in E$ are weights for the edges, indicating the importance of this transplant, and our goal is to find a maximum weight node disjoint cycle and chain packing, where the cycles can use at most k nodes and the chains must originate from nodes in N.

We introduce some notation. For each $v \in V$, let $\delta^-(v)$ be the edges pointing to v and $\delta^+(v)$ be the edges outgoing from v. Likewise, for a set of nodes $S \subset V$, let $\delta^-(S)$ be the set of edges into S and $\delta^+(S)$ be the set of edges out of S. For every $S \subset V$, let $E(S)$ be the set of all edges with both endpoints in S. For a set of edges $D \subset E$, let $V(D)$ be the set of vertices that are of the endpoints of edges in D. Let $\mathcal{C}$ be the set of all simple cycles in G, where each cycle $C \in \mathcal{C}$ is represented by a collection of edges, that is, $C \subset E$. Let $\mathcal{C}_k$ be the subset of $\mathcal{C}$ consisting of cycles that use k or fewer edges. For each $v \in V$, let $\mathcal{C}_k(v)$ be the cycles from $\mathcal{C}_k$ containing an edge incident to v. Given a cycle C, let $w_C = \sum_{e \in C} w_e$ be the total weight of the cycle according to our weight vector $\mathbf{w}$.

Recursive Algorithm for the KEP

We now describe the first of our two algorithms. The algorithm uses a straightforward IP formulation of the KEP, with a binary variable for every edge, and constraints so that each node is used at most once and no long cycles occur.

We first formulate the problem so that the objective is to maximize a weighted number of matches subject to the constraint that no cycle in the graph of length more than k is chosen. Because the number of such constraints is too large, the algorithm will only add these constraints when necessary. In particular, the algorithm first solves the relaxed problem where the bound on the cycle length is dropped, and then if it finds a solution with cycles longer than permitted it adds the corresponding constraints to eliminate these cycles and resolves the corresponding optimization problem.

Formally, we use decision variables y_e for $e \in E$ and f_v^i (flow in) and f_v^o (flow out) for $v \in V$, and solve

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e y_e \\ \text{s.t.} \quad & \sum_{e \in \delta^-(v)} y_e = f_v^i \quad v \in V \\ & \sum_{e \in \delta^+(v)} y_e = f_v^o \quad v \in V \end{aligned} \qquad [1]$$

$$f_v^o \le f_v^i \le 1 \quad v \in P, \qquad [2]$$

$$f_v^o \le 1 \quad v \in N, \qquad [3]$$

$$\sum_{e \in C} y_e \le |C| - 1 \quad C \in \mathcal{C} \backslash \mathcal{C}_k, \qquad [4]$$

$$y_e \in \{0,1\} \quad e \in E. \qquad [5]$$

Note that we introduce some auxiliary variables f_v^i and f_v^o for all $v \in V$ to simplify the formulation, although because they are defined by the equality constraints they can be eliminated. In words, **2** says that for the patient–donor pair nodes the flow out is at most the flow in, and the flow in is at most one; **3** says that for the NDD nodes the flow out is at most one; and in **4** the "cycle inequalities" say that for any cycle C of length greater than k the number of edges we can use is at most $|C| - 1$, thus prohibiting long cycles when $\mathbf{y}$ is integral.

The number of constraints in the IP above is exponential in $|E|$, owing to **4**. As a result, for large instances we cannot simply enumerate all of these constraints and give them directly to the IP solver. Instead, we use a simple recursive algorithm to solve the problem. First, we eliminate all of the constraints in **4** and solve the integer program to optimality. Then, we check whether the proposed solution contains any cycles of length greater than k. If so, we add the violated constraint from **4** and resolve. We repeat this procedure until our solution contains no cycles longer than k. This methodology is generally referred to using "lazy constraints." The effectiveness of this technique depends on our not having to generate too many of the constraints from **4** on a typical input. In the worst case, we might have to solve exponentially many IPs, and to solve each IP may take an exponential amount of time. However, as we will show, this technique is often quite effective in practice. Finally, note that the efficiency of this procedure relies on the fact that we can very quickly detect whether any of the constraints from **4** are violated for an integer solution, because we can (trivially) find the largest cycle in a degree-two graph in linear time.

PC-TSP–Based Algorithm for the KEP

We now describe our second algorithm for solving the KEP, inspired by a solution to the PC-TSP. Recall that in the TSP one is given a list of cities and the cost of going between pairs of cities, and the goal is to find a cycle visiting each city exactly once at the minimum cost (19). In the PC-TSP, again one must find a cycle visiting each city at most once, but now one has the option of skipping some cities entirely and paying a penalty (see ref. 20 or, more recently, ref. 21). Qualitatively, the PC-TSP problem is similar to the KEP in that one wants to find long paths in a graph (which the PC-TSP then closes off as a cycle) without the need to visit every node. As we will see below, our solution is similar to the solution for the PC-TSP.

For each cycle C of length at most k, we introduce a new variable z_C that indicates whether we are using the cycle C. We make the natural updates to **1** so the objective value does not change when the same edges are used and to **2** so that edges cannot be used both in a z_C variable and a y_e variable. Finally, we add **7** to block cycles longer than length k, similarly to **4**.

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e y_e + \sum_{C \in \mathcal{C}_k} w_C z_C \\ \text{s.t.} \quad & \sum_{e \in \delta^-(v)} y_e = f_v^i \quad v \in V \\ & \sum_{e \in \delta^+(v)} y_e = f_v^o \quad v \in V \\ f_v^o + & \sum_{C \in \mathcal{C}_k(v)} z_C \le f_v^i + \sum_{C \in \mathcal{C}_k(v)} z_C \le 1 \quad v \in P, \end{aligned} \qquad [6]$$

$$\begin{aligned} & f_v^o \le 1 \quad v \in N, \\ & \sum_{e \in \delta^-(S)} y_e \ge f_v^i \quad S \subseteq P, \ v \in S \\ & y_e \in \{0,1\} \quad e \in E, \\ & z_C \in \{0,1\} \quad C \in \mathcal{C}_k. \end{aligned} \qquad [7]$$

The constraint **7** is very similar to the cut set inequalities for the TSP (19) as adapted to the PC-TSP by several authors (see ref. 21 and references therein). Fig. S4 provides a clarifying example explaining these constraints. Essentially, they work as follows. Suppose that a chain is reaching some node v, and as a result f_v^i equals one. Now suppose that we cut the graph into two pieces such that the half containing v does not contain any of the NDD nodes from N. Because every chain begins at some node in N (and thus does not begin in S), in order for our chain to reach

$v \in S$ it must use an edge that begins not in S and ends in S, that is, an edge $e \in \delta^{-}(S)$. Thus, our constraint requires that whenever there is flow into v, for every way that v can be cut off from the NDDs N there is at least this much flow over the cut.

Observe that the use of cycles is important for this formulation to be feasible, which makes it quite novel. For example, consider a graph that contains just a set of short disjoint cycles, which is therefore also the optimal solution. Using only constraint **7** with the set $S = V$ and any $v \in V$ implies that the flow going into each of the nodes has to be zero.

Again, the IP formulation has exponentially many constraints from **7**, so we cannot enumerate them and feed them directly to the IP solver. We could simply use the same recursive heuristic (lazy constraints) from the previous section to obtain a correct algorithm. Our solution still relaxes the constraints **7**, but instead we more aggressively attempt to find violated constraints and add them sooner, using a technique called "cutting planes" (or "user cuts"). To apply this method, we needed an efficient algorithm that given a potentially fractional solution can either find a violated constraint from **7** or determine that none exists. This problem is known in the field of optimization as the separation problem.

Theorem 1. *The separation problem for* **7** *can be solved by solving* $O(|P|)$ *network flow problems.*

A proof is given in *Supporting Information*. See ref. 19 for more on the separation problem and on the network flow problem. The solution is again very similar to the solution used to solve separation problems for the TSP and PC-TSP.

Measuring the Value of Long Chains with Clinical Data

To demonstrate the need for long chains in practice, we give an analysis of the number of additional transplants long chains enable, and the types of patients that receive these additional transplants. A patient's so-called percentage reactive antibody (PRA) measures the percentage of the donor pool that a patient has a tissue type incompatibility with (thus patients with a PRA near 100 are very difficult to match, whereas patients with a PRA near zero are easier to match). Patients with a PRA above 80 are commonly referred to as highly sensitized patients. In ref. 16 it was shown for a random model of kidney exchange that when there are many highly sensitized patients allowing for long chains will significantly increase the total number of transplants performed, and that the extra transplants will largely benefit the population of highly sensitized patients. In the data obtained from the APD and NKR exchange programs, we found that in the current exchange pools more than 35% of the patients are highly sensitized, so ref. 16 suggests that we should get some benefit out of allowing long chains.

To assess the added benefit of long chains, we used the following two-stage experiment. First, we repeatedly simulated the dynamic environment of a kidney exchange pool using clinical data (we use 650 patient–donor pairs and 33 altruistic donors from a period of 2.5 y and to create a large set of realistic snapshots of the exchange pool. We measured the effect of changing the maximum chain length on (i) the total number of transplants performed and (ii) the number of transplants performed on highly sensitized patients.

Note that this experiment is designed to demonstrate that long chains will often provide immediate opportunities to perform more transplants. This experiment, however, does not assess the long-run implications of using long chains or the effect of using long chains on the dynamics of the exchange pool (because in practice matched pairs leave the pool, whereas matched pairs may still be present in subsequent matchings during the simulation). Although demonstrating the latter would make a stronger case for the value of long chains, doing so requires dealing with several subtleties that are beyond the scope of this work. In particular, to understand long-run pool performance it is insufficient to simply look at the total number of transplants performed over the entire time horizon, because given enough time in a model with no exit of patients and donors eventually everyone will be matched. To get the full picture, the time patients spend waiting to be matched should also be considered. However, to accurately model this requires more complicated simulations of the pool dynamics than are used here that account for real-world features such as exit, edge failures, and the time to schedule transplants. Thus, we choose not to demonstrate the value of long chains by computing the long-term gains from their use, because these calculations are sufficiently complicated to be a separate problem. We instead attempt the more modest goal of demonstrating that in the types of pools that would be generated by the natural kidney exchange dynamics using longer chains enables more immediate transplants.

Finally, note that to create snapshots that are representative of actual instances encountered by a KPD program it is insufficient to simply take altruistic donors and patient–donor pairs at random from a historical dataset, because the patients that are left over after each match run statistically tend to be harder to match than a randomly selected patient (the easy-to-match patients are more likely to be matched immediately).

We now describe our two-stage experiment design in full detail:

i) Generate a set of realistic snapshots of the exchange pool by performing 50 trials of the procedure below and saving all snapshots generated (every trial involves more than 100 snaphots):

- Assign each patient–donor pair and altruistic donor an arrival date by shuffling their historical arrival dates (i.e., select a permutation uniformly at random).
- Wait for the arrival of eight new participants to the exchange pool (either patient–donor pairs or altruistic donors) and record the current pool of those not yet matched as a snapshot.
- Solve the KEP for the current snapshot with the objective of maximizing the total number of transplants, a maximum cycle length of three, and a maximum chain length of five.
- Remove all matched patient–donor pairs from the current pool. The unmatched pairs stay in the pool. Now, if there are more arrivals remaining, we return to the second step (where we wait for new patient donor pairs), and we repeat in this manner until all pairs have arrived.

ii) Produce the following measures of improvement from allowing unbounded maximum chain length in comparison with a bounded chain length. In particular, for every bounded chain length j for $j \in \{3, 4, 5, 6\}$ we measure the following.

- Chance of more highly sensitized patients receiving a transplant: For each snapshot we count the increase percentage of highly sensitized patients (with PRA above 95) matched by allowing unbounded chains vs. bounded chain length of j.
- For each snapshot we measure the increase percentage of all matched patients by allowing unbounded chains vs. bounded chain length of j.
- We measure the number of instances in which there was at least one highly sensitized patient matched and measure the percentage of instances in which more highly sensitized patients were matched using unbounded chains vs. bounded chains of length j.

Note that in our method of generating snapshots bounding the maximum chain of length of five is somewhat arbitrary. Although this method is not ideal, because the method used to generate snapshots affects the statistics of subsequent snapshots, studying the dynamic optimization problem is beyond the scope of this paper.

The results of our experiment are summarized in Table S2. Allowing longer chains allows matching more of the most highly sensitized patients (with PRA at least 95) and also increases the total number of matched patients. Note also that the percentage of instances in which more highly sensitized patients are matched using unbounded chains in comparison with bounded chains of length $j=3,4,5,6$ is nonnegligible (between 35% and 23%).

Algorithm Performance in Practice

We briefly summarize computational results for both the recursive formulation and the PC-TSP formulation described above. Table S3 provides the running time of both formulations on a series of "difficult" but realistic KEP instances encountered in practice.

To create these difficult but realistic instances we adapted the method from the previous section for generating realistic snapshots. However, when generating instances for this section we rotated through many combinations of the following three simulation parameters (all of these were fixed when generating snapshots previously):

i) We considered several objective functions giving varying degrees of priority for hard-to-match patients.
ii) We considered a maximum chain length of 3, 4, 5, 6, and unbounded.
iii) We varied the number of arrivals between reoptimizations from 1 to 64.

Finally, because we are interested in performance in difficult instances for Table S3 we only retained the instances where at least one algorithm had to generate at least 500 constraints of the type **4** or **7** in the recursive and PC-TSP algorithms, respectively. Thus, these instances represent more of a worst case than an average case for real data.

Table S3 contains the running time of both algorithms on these difficult instances, with a maximum attempted solve time of 20 min. In particular, in all instances with the reported running time less than 20 min the optimal solution was found. Looking at the table, we make the following observations in comparing the performance of the two algorithms:

- Both algorithms are able to solve most instances to optimality quickly.
- The PC-TSP formulation solves all instances to optimality.
- The PC-TSP algorithm is usually faster, although for the easier of these difficult instances the difference is sometimes negligible.
- On several inputs, the PC-TSP algorithm is orders of magnitude faster.

We stress again that these instances are the worst-case instances, in that we only showed results for problems where at least one algorithm had to generate a large number of constraints. These worst-case inputs are only a small fraction of all of the simulated inputs and, generally speaking, both algorithms can solve most of these instances to optimality very quickly.

To demonstrate that our algorithms can solve instances even larger than those occurring in implementing algorithms online we also ran our algorithms on the entire historical datasets for the KPD programs NKR and APD. Each dataset contains around 1,000 patients (although arriving over the span of several years), making these instances much larger than the instances described in Table S3. The running time for our algorithms on these instances is shown in Table S4. We see that *i*) both algorithms can solve both instances and *ii*) the recursive algorithm is much faster.

Although the second point may seem surprising, we do have some explanation as to why this is taking place. First, and most importantly, these instances are substantially different from the instances that KPDs encounter in practice, in that they do not contain a disproportionately high fraction of hard-to-match patients. As a result, there is a very large number of two and three cycles, making the number of variables in the PC-TSP formulation very large. Second, these instances are both "easy" instances, in that very few of the constraints **4** and **7** must be added by the algorithm to solve the integer programs, unlike the instances in Table S3. As we will see in the next section, the advantages of using the PC-TSP algorithm over the recursive algorithm depend on the constraints **4** and **7** being binding in the optimization problem.

For the purposes of comparing algorithms it would be preferable to have more realistic large-scale instances beyond the two described above, but the current historical data do not produce such large-scale instances. In an attempt to produce more realistic large-scale instances from the historical dataset we experimented with removing fractions of the altruistic donors at random. We found that these did not significantly change the performance of either algorithm.

Understanding the Power of the PC-TSP Formulation

We now describe a theoretical result that explains why the PC-TSP algorithm has much better performance than the recursive algorithm on the "difficult" instances from Table S3. To do so, we must first introduce some standard concepts from optimization theory.

Suppose two formulations of an integer program are given. Without the loss of generality, assume the underlying problem is of the maximization type. The linear programming relaxation of an integer program is the value of the optimal solution obtained when the integrality constraints are removed and the problem is solved as a linear programming problem. Let Z_1 and Z_2 be the optimal solutions to the linear programming relaxations for the given two formulations.

Definition 1: *If* $Z_1 \le Z_2$ for every problem instance, then formulation one is defined to be at least as strong as formulation two. We use the notation $Z_1 \preceq Z_2$. If in addition there exists a problem instance such that $Z_1 < Z_2$, then we say that formulation one is stronger than formulation two and use the notation $Z_1 \prec Z_2$.

Very often in practice the stronger formulations greatly reduce the actual running time of the IP problems (19).

We compare the strength of our formulations presented using *Definition 1*. Let the optimal values of the linear programming relaxations of the IP problems corresponding to the recursive algorithm and the PC-TSP–based algorithms be given by Z_{rec} and Z_{tsp}, respectively.

Theorem 2. *The PC-TSP formulation is stronger than the recursive formulation, that is,* $Z_{\text{tsp}} \prec Z_{\text{rec}}$.

The proof can be found in *Supporting Information*.

The idea behind this theorem is demonstrated by a somewhat pathological instance of the KEP shown in Fig. S5. This instance has no altruistic donors and a maximum cycle length of three, and all edges have weight one. Compared with the real instances we considered, this is a small instance with only 30 nodes and 120 edges. The 30 nodes are arranged in a single large directed cycle, with a few additional edges. This graph has two important properties: There are no cycles of length three or less, but it has a very large number of cycles of length 30. Because there are no NDDs or short cycles, the optimal solution is zero. It is easy to see that the linear programming relaxation for the PC-TSP formulation is also zero. Thus, for this instance, solving the IP problem is no harder than solving a single linear programming problem with the PC-TSP formulation. Further, to solve the LP, we only need to generate a single cut from the exponential family of **7** (any cut where $S=P$ will do). As a result, we are able to solve this instance almost instantly with the PC-TSP algorithm. In contrast, for the IP problem corresponding to the recursive algorithm, the linear programming relaxation has the optimal value 29, because it can simply assign value 29/30 to every edge on the length-30 cycle. Worse yet, after 2 h of the running time the best upper bound that the recursive formulation gives is still 30, because the high redundancy in the structure of the graph results in many possible cycles of length 30. The constraints for each of these cycles need to be added to obtain an upper bound of 29. For similar reasons, branch and bound is very ineffective. In the proof, we use two formulations, subtour elimination and a cycles formulation, which are shown to be weaker than the recursive formulation but stronger than the PC-TSP–based formulation (see also Figs. S6–S8).

Conclusion

We provide two new algorithms that solve realistic instances of the KEP encountered in practice. The first algorithm creates an integer program with a variable for each possible transplant and constraints ensuring that each donor and patient are in at most one transplant and then recursively resolves this IP and adds constraints to prevent the long cycles appearing in solutions. The second algorithm uses an IP based on formulation similar to the PC-TSP. To generate realistic instances of the KEP, we simulated dynamics of the patient pool in a kidney exchange using clinical data from the two major KPD Programs: NKR and the APD. Our algorithms have been able to solve all real data instances encountered in practice thus far. Our algorithms find optimal solutions for large-scale practical instances of the KEP with long chains after a very short running time, with the PC-TSP–based algorithm outperforming the recursion-based algorithm in many instances.

1. Roth AE, Sönmez T, Unver MU (2004) Kidney exchange. *Q J Econ* 119(2):457–488.
2. Roth AE, Sönmez T, Unver MU, Delmonico FL, Saidman SL (2006) Utilizing list exchange and nondirected donation through 'chain' paired kidney donations. *Am J Transplant* 6(11):2694–2705.
3. Rees MA, et al. (2009) A non-simultaneous extended altruistic donor chain. *New Engl J Med* 360(11):1096–1101.
4. Sack K (February 18, 2012) 60 lives, 30 kidneys, all linked. *NY Times*. Available at www.nytimes.com/2012/02/19/health/lives-forever-linked-through-kidney-transplant-chain-124.html. Accessed December 23, 2014.
5. Abraham DJ, Blum A, Sandholm T (2007) Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. *Proceedings of the 8th ACM Conference on Electronic Commerce* (Assoc Computing Machinery, New York), pp 295–304.
6. Biró P, Manlove DF, Rizzi R (2009) Maximum weight cycle packing in directed graphs, with application to kidney exchange programs. *Discrete Mathematics Algorithms Applications* 1(04):499–517.
7. Roth AE, Sönmez T, Ünver MU (2007) Efficient kidney exchange: Coincidence of wants in markets with compatibility-based preferences. *Am Econ Rev* 97(3):828–851.
8. Constantino M, Klimentova X, Viana A, Rais A (2013) New insights on integer-programming models for the kidney exchange problem. *Eur J Oper Res* 231(1):57–68.
9. Dickerson JP, Procaccia AD, Sandholm T (2012) Optimizing kidney exchange with transplant chains: Theory and reality. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems* (Assoc Computing Machinery, New York), Vol 2, pp 711–718.
10. Manlove DF, O'Malley G (2012) Paired and altruistic kidney donation in the UK: Algorithms and experimentation. *Experimental Algorithms*, Lecture Notes in Computer Science (Springer, Berlin), Vol 7276, pp 271–282.
11. Glorie KM, Wagelmans APM, Van de Klundert J (2012) Iterative branch-and-price for hierarchical multi-criteria kidney exchange. Econometric Institute Research Paper EI 2012-11 (Econometric Inst, Erasmus Univ, Rotterdam).
12. Klimentova X, Alvelos F, Viana A (2014) A new branch-and-price approach for the kidney exchange problem. *Computational Science and Its Applications – ICCSA 2014*, Lecture Notes in Computer Science (Springer, Berlin), Vol 8580, pp 237–252.
13. Füredi Z, Kahn J, Seymour PD (1993) On the fractional matching polytope of a hypergraph. *Combinatorica* 13(2):167–180.
14. Hazan E, Safra S, Schwartz O (2006) On the complexity of approximating k-set packing. *Computational Complexity* 15(1):20–39.
15. Björklund A, Husfeldt T, Khanna S (2004) Approximating longest directed paths and cycles. *Automata, Languages and Programming*, Lecture Notes in Computer Science (Springer, Berlin), Vol 3142, pp 222–233.
16. Ashlagi I, Gamarnik D, Rees MA, Roth AE (2012) The need for (long) chains in kidney exchange. (National Bureau of Economic Res, Cambridge, MA).
17. Dickerson JP, Procaccia AD, Sandholm T (2013) Failure-aware kidney exchange. *Proceedings of the Fourteenth ACM Conference on Electronic Commerce* (Assoc Computing Machinery, New York), pp 323–340.
18. Chen Y, et al. (2012) Graph-based optimization algorithm and software on kidney exchanges. *IEEE Trans Biomed Eng* 59(7):1985–1991.
19. Bertsimas D, Weismantel R (2005) *Optimization over Integers* (Dynamic Ideas, Belmont, MA), Vol 13.
20. Bienstock D, Goemans MX, Simchi-Levi D, Williamson D (1993) A note on the prize collecting traveling salesman problem. *Math Program* 59(1):413–420.
21. Goemans MX (2009) Combining approximation algorithms for the prize-collecting TSP. arXiv:0910.0553,.

Supporting Information

Anderson et al. 10.1073/pnas.1421853112

Proof of Theorem 1

Proof: Let $(\mathbf{y}, \mathbf{z})$ be the point for which we must determine whether the constrain in Eq. **7** is satisfied. Following the well-known procedure for PC-TSP, first we form a directed weighted graph $\overline{G} = (\overline{V}, \overline{E}, \overline{\mathbf{w}})$ where $\overline{V} = \{s\} \cup V$ where s is an extra node, $\overline{E} = E \cup \{(s, n) | n \in N\}$, and weights $\overline{w}_e$ for $e \in \overline{E}$ are given by

$$\overline{w}_e = \begin{cases} y_e & e \in E, \\ 1 & \text{otherwise}, \end{cases}$$

(the edges with $\overline{w}_e = 1$ each go from the super source to a node in N).

Then, for every $v \in P$ where $f_v^i > 0$ we solve the max flow min cut problem with source s and sink v. If we find a cut of weight less than f_v^i, then by taking S to be the set of nodes on the sink side of the cut we have found a violated constraint. As we are optimizing over all cuts separating v from the super source and then checking all v we in fact check all of the constraints in Eq. **7**.

Proof of Theorem 2: Before proving the result, we introduce two auxiliary IP formulations.

Subtour Elimination Formulation

We propose an alternative formulation on the same set of variables as the PC-TSP formulation, called the subtour formulation. The name is derived from the subtour elimination formulation of the TSP, as in ref. 1. In the subtour elimination formulation, all of the constraints are the same as the PC-TSP formulation except that Eq. **7** is replaced by

$$\sum_{e \in E(S)} y_e + \sum_{\substack{D \in \mathcal{C}_k \\ V(D) \subseteq S}} (|D| - 1) z_D + \sum_{\substack{D \in \mathcal{C}_k \\ V(D) \nsubseteq S}} |D \cap E(S)| z_D \le |S| - 1 \quad S \subset P. \qquad \textbf{[S1]}$$

The idea of the constraint is that subset S of the nodes, the number of edges used to make chains, should be at most $|S| - 1$. The IP formulation would still be correct without the second and third sums; however, their addition strengths the inequality.

Cycles Formulation

We propose another alternative formulation on the same set of variables as the PC-TSP and subtour formulations, the cycles formulation. In the cycles formulation all of the constraints are the same as in the PC-TSP and subtour formulations except that Eq. **2** is replaced by

$$\sum_{e \in C} y_e + \sum_{\substack{D \in \mathcal{C}_k \\ D \neq C}} |D \cap C| z_D \le |C| - 1 \quad C \in \mathcal{C}. \qquad \textbf{[S2]}$$

The idea of the constraint is to prevent the edges of C from forming a cycle, unless the variable z_C is used (should the variable exist). Obviously, if $y_e = 1$ for $e \in C$, they would form a cycle, but the constraint prevents this. Again, the IP formulation would still be correct without the second sum in the constraint, but including this sum makes the formulation stronger. In Fig. S6 we give an example of an instance where, if the second sum were not included, the cycle formulation would be weaker than the recursive formulation.

Proof of Result

Let Z_{cyc} and Z_{sub} be the value of the LP relaxation for the cycle and subtour formulations of the KEP, respectively. Let P_{rec}, P_{cyc}, P_{sub}, and P_{tsp} be the polyhedrons for the LP relaxations of each formulation. We will now show

$$Z_{\text{tsp}} \prec Z_{\text{sub}} \prec Z_{\text{cyc}} \preceq Z_{\text{rec}}.$$

Because the relations above are transitive, this will imply *Theorem 2*.

Proof: First, we show that $P_{\text{tsp}} \subseteq P_{\text{sub}}$, which immediately implies that $Z_{\text{tsp}} \preceq Z_{\text{sub}}$, because the two formulations share the same objective function. It suffices to show that each of the subtour elimination constraints from Eq. **8** are implied by the entire cut set formulation. Fix $S \subseteq P$ and assume that $\mathbf{y}$ is feasible for the cut set formulation. Fix some $u \in S$. First, we claim that

$$\sum_{\substack{D \in \mathcal{C}_k \\ V(D) \subseteq S}} (|D| - 1) z_D + \sum_{\substack{D \in \mathcal{C}_k \\ V(D) \nsubseteq S}} |D \cap E(S)| z_D \le \sum_{\substack{D \in \mathcal{C}_k \\ V(D) \cap S \neq \emptyset}} (|V(D) \cap S| - 1) z_D \qquad \textbf{[S3]}$$

$$\le \sum_{\substack{v \in S \\ v \neq u}} \sum_{D \in \mathcal{C}_k(v)} z_D. \qquad \textbf{[S4]}$$

To justify Eq. **S3**, observe that for cycles D such that $V(D) \subseteq S$ we immediately have $|D| = |V(D)| = |V(D) \cap S|$, so for these z_D terms, we have $|D| - 1 = |V(D) \cap S| - 1$. For D such that $V(D) \nsubseteq S$, we have two cases:

- If $V(D) \cap S = \emptyset$, then $D \cap E(S) = \emptyset$ as well, so these terms can be dropped.
- If D has ℓ vertices in S, where $0 < \ell < |D|$, then at most $\ell - 1$ of the edges of D will have both endpoints in S.

Thus, Eq. **S3** has been shown. To justify Eq. **S4**, by a simple counting argument we have the following:

- If $u \notin V(D)$, then the term z_D will appear $|V(D) \cap S|$ times in Eq. **S4**.
- If $u \in V(D)$, then the term z_D will appear $|V(D) \cap S| - 1$ times in Eq. **S4.**

Thus, Eq. **S4** has been shown. Applying this inequality, we now have

$$\begin{aligned} &\sum_{e \in E(S)} y_e + \sum_{\substack{D \in \mathcal{C}_k \\ V(D) \subseteq S}} (|D| - 1) z_D + \sum_{\substack{D \in \mathcal{C}_k \\ V(D) \nsubseteq S}} |D \cap E(S)| z_D \\ &\le \sum_{e \in E(S)} y_e + \sum_{\substack{v \in S \\ v \neq u}} \sum_{D \in \mathcal{C}_k(v)} z_D \\ &= \sum_{v \in S} f_v^i - \sum_{e \in \delta^-(S)} y_e + \sum_{\substack{v \in S \\ v \neq u}} \sum_{D \in \mathcal{C}_k(v)} z_D \qquad \textbf{[S5]} \\ &= f_u^i - \sum_{e \in \delta^-(S)} y_e + \sum_{\substack{v \in S \\ v \neq u}} \left(f_v^i + \sum_{D \in \mathcal{C}_k(v)} z_D \right) \end{aligned}$$

$$\le \sum_{\substack{v \in S \\ v \neq u}} \left(f_v^i + \sum_{D \in \mathcal{C}_k(v)} z_D \right) \qquad \textbf{[S6]}$$

$$\le |S| - 1, \qquad \textbf{[S7]}$$

where Eq. **S5** follows as for a set of nodes S; all edges incoming to a node in S have their other endpoint either in S or outside of S. Eq. **S6** follows from applying Eq. **S2** (multiplied by -1) for the set S and the vertex u, and Eq. **S7** follows from applying the upper bound from the flow constraint in Eq. **S6** $|S|-1$ times.

Next, we show that $P_{\text{sub}} \subseteq P_{\text{cyc}}$ and thus $Z_{\text{sub}} \le Z_{\text{cyc}}$. It suffices to show that for any cycle C Eq. **S9** is directly implied by Eq. **S8** taking $S = V(C)$. To bound the first term of the left-hand side of Eq. **S9**, we have

$$\sum_{e\in C} y_e \le \sum_{e\in E(S)} y_e.$$

For the second term, we will partition $D \in \mathcal{C}_k, D \ne C$ into two sets, those where $V(D) \subseteq S$ and $D \ne C$, or those where $V(D) \nsubseteq S$, that is,

$$\sum_{\substack{D\in\mathcal{C}_k\\ D\ne C}} |D\cap C| z_D = \sum_{\substack{D\in\mathcal{C}_k\\ V(D)\subseteq S\\ D\ne C}} |D\cap C| z_D + \sum_{\substack{D\in\mathcal{C}_k\\ V(D)\nsubseteq S}} |D\cap C| z_D.$$

For the first sum, we have $|D\cap C| \le |D|-1$, because $D \ne C$ (and $D \not\subset C$ because both D and C are simple cycles). Thus,

$$\sum_{\substack{D\in\mathcal{C}_k\\ V(D)\subseteq S\\ D\ne C}} |D\cap C| z_D \le \sum_{\substack{D\in\mathcal{C}_k\\ V(D)\subseteq S\\ D\ne C}} (|D|-1) z_D \le \sum_{\substack{D\in\mathcal{C}_k\\ V(D)\subseteq S}} (|D|-1) z_D.$$

For the second sum, because $C \subset E(S)$, we have $|D\cap C| \le |D\cap E(S)|$ for all $D \in \mathcal{C}_k$, and thus

$$\sum_{\substack{D\in\mathcal{C}_k\\ V(D)\nsubseteq S}} |D\cap C| z_D \le \sum_{\substack{D\in\mathcal{C}_k\\ V(D)\nsubseteq S}} |D\cap E(S)| z_D.$$

Putting everything together then applying Eq. **S8** we have

$$\begin{aligned}
&\sum_{e\in C} y_e + \sum_{\substack{D\in\mathcal{C}_k\\ D\ne C}} |D\cap C| z_D \\
&\quad\le \sum_{e\in E(S)} y_e + \sum_{\substack{D\in\mathcal{C}_k\\ V(D)\subseteq S}} (|D|-1) z_D + \sum_{\substack{D\in\mathcal{C}_k\\ V(D)\nsubseteq S}} |D\cap E(S)| z_D \\
&\quad\le |S|-1 = |C|-1,
\end{aligned}$$

showing the claim.

To show that $Z_{\text{cyc}} \preceq Z_{\text{rec}}$, consider $(\mathbf{y}^*, \mathbf{z}^*) \in P_{\text{cyc}}$ that is optimal for the cycle formulation (the values of f_v^i and f_v^o are implied by $\mathbf{y}^*$). If we let

$$x_e = y_e^* + \sum_{C\in\mathcal{C}_k,\ e\in C} z_C^*,$$

then we claim that $\mathbf{x} \in P_{\text{rec}}$ (again with the values of the flow variables being determined by $\mathbf{x}$). To show this, it suffices to verify Eqs. **S2**–**S4** hold for $\mathbf{x}$. To obtain Eq. **S2**

$$\sum_{e\in\delta^+(v)} x_e = \sum_{e\in\delta^+(v)} \left(y_e^* + \sum_{C\in\mathcal{C}_k,\ e\in C} z_C^* \right) \qquad \textbf{[S8]}$$

$$= \sum_{e\in\delta^+(v)} y_e^* + \sum_{C\in\mathcal{C}_k(v)} z_C^* \qquad \textbf{[S9]}$$

where in Eq. **S8** we applied the definition of x_e, and in Eq. **S9** we used that $\mathcal{C}_k(v)$, the set of cycles hitting v, is equal to the disjoint union over all e going out of v of the set of cycles containing e (the union is disjoint as each cycle contains exactly one edge out of v). Likewise, we have

$$\sum_{e\in\delta^-(v)} x_e = \sum_{e\in\delta^-(v)} y_e^* + \sum_{C\in\mathcal{C}_k(v)} z_C^*.$$

Thus, Eq. **S6** from the cycles formulation implies Eq. **S2** in the recursive formulation. An analogous argument immediately gives us Eq. **S3** as well. Finally, to obtain Eq. **S4** we have for any cycle C with $|C| > k$,

$$\begin{aligned}
\sum_{e\in C} x_e &= \sum_{e\in C} \left(y_e^* + \sum_{\substack{D\in\mathcal{C}_k\\ e\in D}} z_e^* \right) && \textbf{[S10]}\\
&= \sum_{e\in C} y_e^* + \sum_{\substack{D\in\mathcal{C}_k\\ D\ne C}} |D\cap C| z_D \\
&\le |C|-1, && \textbf{[S11]}
\end{aligned}$$

where in Eq. **S10** we are counting, and using that $|C| > k$ implies that there is no $D \in \mathcal{C}_k$ such that $D = C$, and in Eq. **S11** we are applying Eq. **S9**. Thus, we conclude that $\mathbf{x}$ is feasible. Using feasibility, we can obtain the result as follows:

$$\begin{aligned}
Z_{\text{rec}} &\ge \sum_{e\in E} c_e x_e \\
&= \sum_{e\in E} c_e \left(y_e^* + \sum_{\substack{C\in\mathcal{C}_k\\ e\in C}} z_e^* \right) \\
&= \sum_{e\in E} c_e y_e^* + \sum_{C\in\mathcal{C}_k} c_C z_e^* \\
&= Z_{\text{cyc}}.
\end{aligned}$$

In Fig. S7 we give a family of problem instances where $Z_{\text{tsp}} < Z_{\text{sub}}$. In Fig. S8 we give an instance where $Z_{\text{sub}} < Z_{\text{cyc}}$.

The KEP with Bounded Chain Lengths. We show how to adapt the PC-TSP formulation to allow for a maximum chain length ℓ, although the technique would work for any of the four formulations presented (the adaptation is trivial for the cycle and subtour formulations). For each NDD $n \in N$ and each edge $e \in E$, we introduce auxiliary edge variables y_e^n and likewise $f_v^{i,n}$ and $f_v^{o,n}$ indicating flow that must begin at n. The formulation becomes

$$\max \sum_{e\in E} w_e y_e + \sum_{C\in\mathcal{C}_k} w_C z_C$$

$$(\mathbf{y}, \mathbf{z}, \mathbf{f}^i, \mathbf{f}^o) \in P_{\text{tsp}} \qquad \textbf{[S12]}$$

$$\sum_{n\in N} y_e^n = y_e \quad e\in E$$

$$\sum_{e\in E} y_e^n \le \ell \quad n\in N \qquad \textbf{[S13]}$$

$$\sum_{e\in\delta^-(v)} y_e^n = f_v^{i,n} \quad v\in V,\ n\in N \qquad \textbf{[S14]}$$

$$\sum_{e\in\delta^+(v)} y_e^n = f_v^{o,n} \quad v\in V,\ n\in N \qquad \textbf{[S15]}$$

$$f_v^{o,n} \le f^{i,v} \le 1 \quad v\in V,\ n\in N \qquad \textbf{[S16]}$$

$$
\begin{aligned}
&y_e \in \{0,1\} \quad e \in E\\
&z_C \in \{0,1\} \quad C \in \mathcal{C}_k\\
&y_e^n \in \{0,1\} \quad e \in E, \quad n \in N.
\end{aligned}
$$

The new constraints are briefly explained as follows. From Eq. **S12** we have that each edge used (y_e) must be part of a chain beginning at some NDD n. From Eq. **S13** we obtain that each chain can use at most ℓ edges, thus giving the maximum chain length. In Eqs. **S14** and **S15** we just define auxiliary variables denoting whether an edge used in a chain starting at n comes into/out of v. Finally, in Eq. **S16** we enforce that the edges used in the chain starting at n are consecutive. The remaining constraints are exactly the same as the PC-TSP constraints with no maximum chain length.

Stochastic Optimization for the KEP. Here we present a general framework for dealing with the possibility that after an edge is selected it might become ineligible for the matching, an event we refer to as an "edge failure." Edge failures occur commonly in practice for a variety of reasons (e.g., a donor backs out, a patient dies, or a biological incompatibility is discovered).

We propose a two-phase system for planning exchanges that anticipates edge failures occurring at random and plans to maximize the number of transplants performed once the failed edges have been identified and removed. In the first phase, a subset of the edges in the graph are selected to be tested for edge failures. Operational constraints restrict this set, where the basic idea is that it is not practical to check all of the edges. Some natural examples of phase-one edge sets to test include the following:

- Use at most m edges in phase one.
- Each node has in-degree at most m_i and out-degree at most m_o.
- The edges used in phase one must be a feasible solution to the KEP.

The only restriction on the rule used to select phase one edges is that there exists a polyhedron P such that $\mathbf{y} \in P \cap \mathbb{Z}^{|E|}$ iff $\mathbf{y}$ corresponds to a valid set of phase one edges (i.e., the set of phase-one edges must be describable as a mixed-integer program). After the phase-one selections are made, we learn which of the edges that we tested in phase one failed, and in phase two we solve the regular KEP using only edges that we checked and that did not fail in phase one. Because we do not know which edges will fail before we make our phase-one decision, we use the objective of maximizing the expected weight of our phase-two KEP solution when picking our phase-one solution. Next, we describe the probabilistic framework we use for edge failures, and then the computational technique used to compute our phase-one solution.

We assume that there is a family of random variables X_e for $e \in E$, taking the value one if the edge e can be used in the matching and zero otherwise. We make no assumptions about the independence structure of the variables X_e. However, we do assume that we can jointly sample the vector of X_e variables.

We now define a two-stage stochastic integer optimization problem. We have decision variables y_e for $e \in E$ that indicate the edges we wish to test in stage one. In stage two, we observe our realization $\omega \in \Omega$ of $X_e(\omega)$ for the edges where $y_e = 1$ (the edges we tested), and then we form an optimal cycle packing using only edges that we tested in phase one and where $X_e(\omega) = 1$. We select our phase-one edges $\mathbf{y}$, integer and in P, to maximize the expected size of the phase-two packing.

This problem can be solved using the method of sample average approximation, as described and mathematically justified in refs. 2–4. Suppose that we sample the vector of X_e jointly n times, and let x_e^j for $j = 1, \ldots, n$ be the realization of X_e in the jth sample. Let y_e^j be one if we use edge e in realization j and zero otherwise, and likewise let z_C^j be one if we use cycle C in the jth realization. Let P_{tsp}^j be the cut set polyhedron on the variables y_e^j and z_C^j. Our formulation is then as follows:

$$
\max \quad \sum_{j=1}^{n} \left(\sum_{e \in E} c_e y_e^j + \sum_{C \in \mathcal{C}_k} c_C z_C^j \right) \qquad \textbf{[S17]}
$$

$$
\begin{aligned}
\text{s.t.} \quad & \mathbf{y} \in P,\\
& (\mathbf{y}^j, \mathbf{z}^j) \in P_{\text{tsp}}^j,\\
& y_e^j \le y_e \qquad e \in E, j = 1, \ldots, n,\\
& y_e^j \le x_e^j \qquad e \in E, j = 1, \ldots, n,\\
& z_C^j \le y_e \qquad C \in \mathcal{C}_k, e \in C, j = 1, \ldots, n,\\
& z_C^j \le x_e^j \qquad C \in \mathcal{C}_k, e \in C, j = 1, \ldots, n,\\
& y_e \in \{0,1\} \qquad e \in E,\\
& y_e^j \in \{0,1\} \qquad e \in E, j = 1, \ldots, n,\\
& z_C^j \in \{0,1\} \qquad C \in \mathcal{C}_k, j = 1, \ldots, n.
\end{aligned}
$$

This model has a few very attractive features. First, it allows for a general probabilistic model for edge failures, which in practice should be much more accurate than simply independently and identically distributed edge failures. For example:

- If an edge failed because the donor or receiver became ill or backed out, then all edges involving that donor/receiver would be ruled out simultaneously.
- If an edge failed because a receiver developed a new HLA antibody, then all edges from donors with that HLA antigen would fail simultaneously.
- If an edge failed because a doctor or transplant center deemed a donor to be of inadequate quality for the recipient (e.g., too old), then possibly other edges pointing to the same doctor/transplant center would fail, but not necessarily all of them, because a highly sensitized recipient may have lower standards than a standard recipient.

Clearly, a very sophisticated model could be made to predict edge failures. Further, it will likely be easier to draw samples from such a model than to explicitly work out the joint distribution of edge failures.

Another good feature of this model is that we have a great deal of flexibility in choosing P (the set of edges we are allowed to pick in phase one). Our flexibility in choosing P allows us to adapt to various operational constraints of actually running a kidney exchange. Additionally, we can use P to try and influence "agents" (e.g., donors, recipients, doctors, hospitals, and transplant centers) into taking actions that maximize global welfare. For example, if we select more than one incoming edge to a node in phase one, then the receiver, the doctor, the hospital, and the transplant center may be incentivized to reject the worse of the two edges to try and get a higher quality donor. One very simple fix is to restrict the edges tested in phase one to give each node an in-degree of at most one. Then as no one will receive multiple offers, no one will be incentivized to turn down a kidney they otherwise would have accepted.

Finally, note that it is at times desirable to add additional decision variables to the phase-one problem. For example, if we were to restrict our phase-one solution to be a feasible solution to the KEP, while we could take $P = P_{\text{rec}}$, it is computationally more efficient to use the PC-TSP formulation instead. One way of accomplishing this is as follows: Add decision variables $\tilde{y}_e$ for $e \in E$ and $\tilde{z}_C$ for each cycle $C \in \mathcal{C}_k$, let

$$
y_e = \tilde{y}_e + \sum_{\substack{C \in \mathcal{C}_k\\ e \in C}} \tilde{z}_C,
$$

and then take P to be the PC-TSP polyhedron applied to $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{z}}$, along with the constraint above relating $\mathbf{y}$ to $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{z}}$. Further, note that the cut set constraints for the P^j_{tsp} polyhedrons would automatically be implied by the cut set constraints from P_{tsp} on $(\tilde{\mathbf{y}}, \tilde{\mathbf{z}})$ and thus could be eliminated.

Our model and can easily solve problems with up to 30 scenarios on a desktop computer. Various heuristics designed for the sample average approximation approach can provide close to optimal results for larger problems. One may also consider solving a fully stochastic and dynamic optimization model; however, such a model is not tractable because it would include an infinite horizon stochastic dynamic program with a state for every possible graph and a large decision space for every state. One interesting way to tackle this is through approximate dynamic programming (see e.g., ref. 5). Previous studies have shown that a large class of dynamic algorithms do not improve outcomes significantly beyond greedy algorithms (see, e.g., ref. 6 and references therein), and thus solving the one-shot optimization problems is an important challenge.

1. Bertsimas D, Weismantel R (2005) *Optimization over Integers* (Dynamic Ideas, Belmont, MA), Vol 13.
2. Swamy C, Shmoys DB (2005) Sampling-based approximation algorithms for multi-stage stochastic optimization. *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science* (IEEE, New York), pp 357–366.
3. Kleywegt AJ, Shapiro A, Homem-de Mello T (2002) The sample average approximation method for stochastic discrete optimization. *SIAM J Optim* 12(2):479–502.
4. Ahmed S, Shapiro A (2002) The sample average approximation method for stochastic programs with integer recourse. *SIAM J Optim* 12:479–502.
5. Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009) ParamILS: An automatic algorithm configuration framework. *J Artif Intell Res* 36(1):267–306.
6. Anderson R, Ashlagi I, Gamarnik D, Kanoria Y (2013) A dynamic model for barter exchange. ACM technical report (Assoc Computing Machinery, New York).

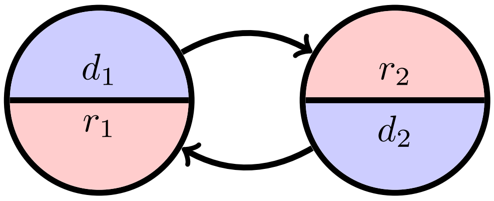

Fig. S1. A cyclic exchange involving two patient–donor pairs. Each pair is represented by a node, where the blue half of the node represents the donor and the red half represents the patient.

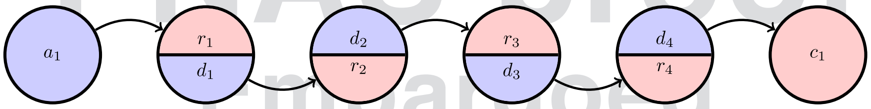

Fig. S2. A chain exchange involving an altruistic donor, d_0, four patient–donor pairs, and a patient with no donor p_5. Each pair is represented by a node, where the blue half of the node represents the donor and the red half represents the patient.

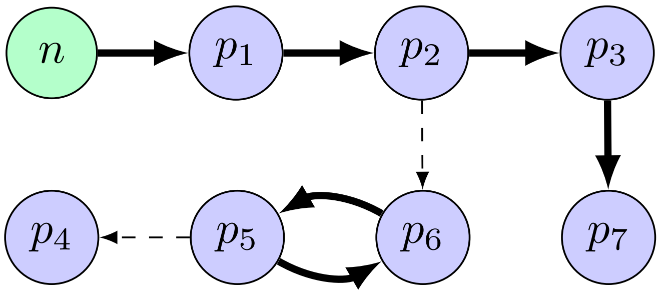

Fig. S3. An example of a KEP instance. The node labeled n is an NDD, and the remaining nodes p_1 through p_7 correspond to patient–donor pairs. Edges indicate possible transplants from the donor in the source node to the patient in the target node. In the optimal solution for this instance, indicated by the bold edges, we form the chain n, p_1, p_2, p_3, p_7, and the two cycle with p_5 and p_6, leaving p_4 unmatched.

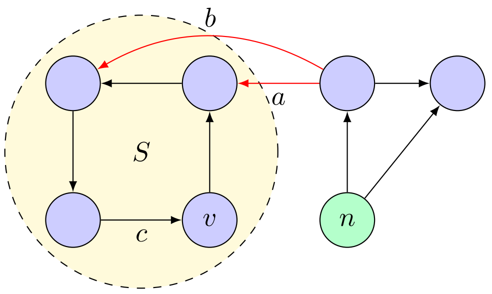

Fig. S4. An example of a cut set constraint. The graph contains a single NDD in green, labeled n. Observe that if node v is to be involved in any chain (i.e., $f_v^i = 1$), then we must use at least one of the edges a or b that go across the cut separating S from the remaining nodes and NDD.

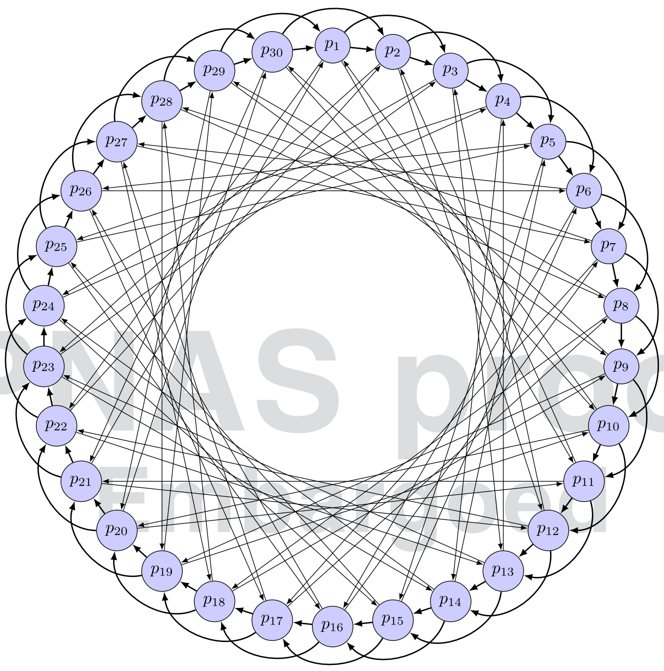

Fig. S5. A pathological instance of the KEP that is very difficult for the direct formulation but is solved trivially by the TSP formulation. The optimal solution is zero.

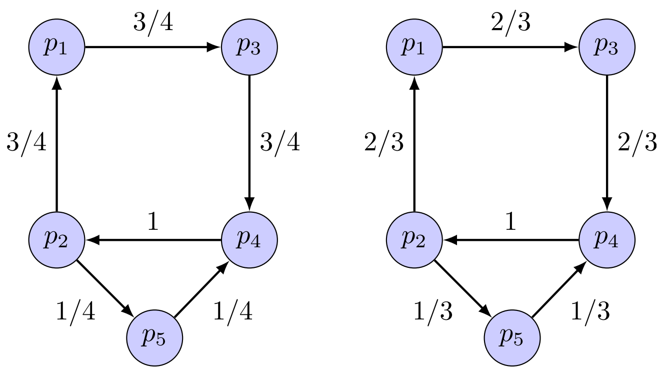

Fig. S6. The figure represents two fractional solutions to demonstrate the necessity of the second sum from the left-hand side of inequality **S9**. In this instance, $P = \{p_1, \ldots, p_5\}$, $N = \emptyset$, the edges are as indicated in the figure above, and all edges have weight one. The numbers next to the edges indicate fractional solutions, namely y_e for the recursive formulation, and $y_e + \sum_{C \in \mathcal{C}_k, e \in C} z_C$ for the cycle formulation. Observe that the solution on the left has greater weight than the solution on the right. The solution on the left is infeasible for the recursive formulation, because the constraint on the cycle $\{(p_1,p_2),(p_2,p_3),(p_3,p_4),(p_4,p_1)\}$ is violated. The solution on the right is optimal for the recursive formulation. For the cycles formulation, letting the cycle $D = \{(3,4),(4,5),(5,3)\}$, without the second sum from the left-hand side of **S9**, we could take $z_D = 1/4$ and $y_e = 3/4$ for $e = (1,2),(2,3),(3,4),(4,1)$ and then fractional solution on the left would be feasible. This would break the result that $Z_{\text{cyc}} \preceq Z_{\text{rec}}$. However, by including the variable z_D in the constraint against the four cycles, we again have that the solution on the right is optimal for the cycles formulation.

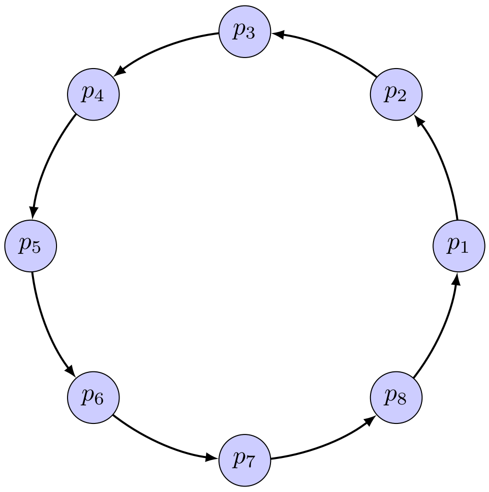

Fig. S7. Consider the family of problem instances on $n \geq 4$ nodes where $P = \{p_1, \ldots, p_n\}$, $N = \varnothing$, there are n edges forming a single cycle of length n, and $w_e = 1$ for every edge. Above is the instance where $n = 8$. The optimal solution for the IP and the PC-TSP LP relaxation are both zero, but the subtour elimination LP relaxation has an optimal solution $n-1$ [each node has $y_e = (n-1)/n$].

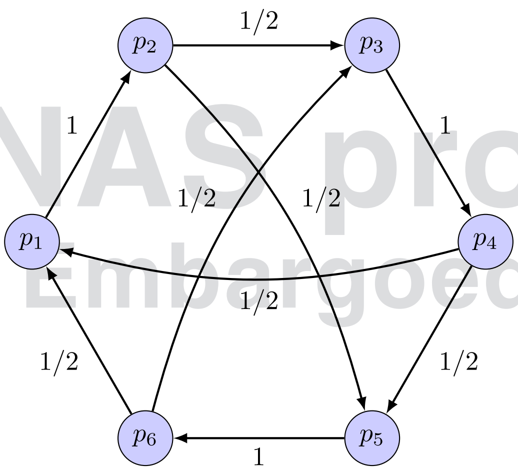

Fig. S8. In the instance on six nodes above, where $k = 3$, $N = \emptyset$, $P = \{p_1, \ldots, p_6\}$, and each edge has weight one, the IP optimum is zero. Taking y_e to be the edge labels in the figure above, we get a feasible solution to the LP relaxation of $Z_{\text{cyc}} = 6$. However, the LP optimum for the subtour formulation is $Z_{\text{sub}} = 5$. We can attain this value by taking $y_{(i,i+1)} = 5/6$ and $y_{(6,1)} = 5/6$. To show that 5 is optimal, we apply constraints **S8** taking $S = P$, to obtain that $\sum_{e \in E(P)} y_e \leq 5$, and then observe that $\sum_{e \in E(P)} y_e$ is equal to the objective function.

Table S1. Average number of chains of size k ($k = 3, 4, 5, 6$) in random pools of various sizes and a single altruistic donor

Nodes	k = 3	k = 4	k = 5	k = 6
150	4,520	69,780	1,063,727	16,116,117
200	5,147	99,046	1,884,160	35,304,432
250	15,407	370,071	8,807,015	207,347,121

Table S2. Additional patients matched for incremental increases in the maximum chain length

Measures	(3,∞), %	(4,∞), %	(5,∞), %	(6,∞), %
Additional highly sensitized (PRA >95) matched	35	27	21	16
Additional patients matched	21	17	14	12
Instances with more highly sensitized matched	35	32	25	23

Table S3. Performance of the recursive and TSP algorithms for "difficult" real-data KEP instances

			Running time, s	
NDDs	Patient–donor pairs	Edges	Recursive	TSP
3	202	4,706	0.148	0.031
10	156	1,109	13.093	0.022
6	263	8,939	59.158	1.655
5	284	10,126	71.066	0.807
6	324	13,175	418.27	0.981
6	328	13,711	474.947	1.947
6	312	13,045	1,200*	0.157*
10	152	1,125	48.56	0.054
3	269	2,642	40.506	0.134
10	257	2,461	67.783	0.258
7	255	2,390	85.475	0.268
6	215	6,145	248.46	0.532
10	255	2,550	216.48	0.126
1	310	4,463	721.66	0.555
11	257	2,502	1,039.105	0.125
6	261	8,915	1,200	4.435
10	256	2,411	587.238	0.114
6	330	13,399	1,200	1.621
10	256	2,347	1,200	0.305
7	291	3,771	1,200*	0.221
8	275	3,158	1,200*	0.224
4	289	3,499	1,200*	0.2
3	199	2,581	1,200*	0.041
7	198	4,882	1,200*	8.204
2	389	8,346	1,200*	0.096

Timeouts (optimal solution not found) are indicated by an asterisk.

Table S4. KEP on very large historical datasets with the recursive and TSP algorithms

				Recursive algorithm		TSP algorithm	
Instance	NDDs	Patient–donor pairs	Edges	Running time, s	RAM, GB	Running time, s	RAM, GB
APD	47	931	190,820	1.79	1	104	25
NKR	162	1,179	346,608	3.074	1	314	37