The Kidney Exchange Problem: Finding Candidate Exchange Algorithms to Facilitate Kidney Paired Donation (KPD)

Alex Hoganson, Bryson Banks, Jose G. Vaso, Trevor Anderson

ABSTRACT

The quality of medical processes within hospital environments determine the quality of medical care; proper procedures prevent catastrophes. Specifically within the field of Kidney transplants there are many data points between donor/donee that must be synced before performing a removal/replacement procedure. Research into medical history demonstrates an evolving Kidney Paired Donation process, with the introduction of altruistic/non-family donors and the fading of autoimmune complications. To mitigate these challenges, researchers developed algorithms to more efficiently pair kidney exchange participants such that transplant surgeries could be scheduled quickly and without introducing surgery risk factors. We explored various design alternatives according to these algorithm types, and focused specifically on attempting to replicate Branch-and-Price with Polynomial-Time Pricing for Cycles. Using simulated patient record data as a method for evaluation, our team identifies several metrics to identify algorithm effectiveness as well as challenges that affected our results.

1 INTRODUCTION

The Kidney is a vital part of the Urinary and Endocrine systems in the human body, filtering out harmful toxins within the bloodstream. In essence, a human being does not actually require two Kidneys to survive, since just one has the capability to adequately filter out these toxins. Evolution somehow created a redundancy configuration in the endocrine system such that if one Kidney fails, the system may still function. In the event that both Kidneys are non-operational, that might seem like a death sentence, yet in the modern age there are two main avenues of mitigation: dialysis and Kidney exchanges. For dialysis, a patient without Kidneys travels daily to a clinic to have toxins filtered by a mechanical device that serves as an external kidney. However, this practice significantly diminishes quality of life, blocking out several hours every day for mandatory blood filtration with the consequence of death if unfulfilled.

Therefore the concept of Kidney transplants has been acknowledged as a better long term solution to fix complete Kidney failure. In the present day Kidney transplants are rather common, but history demonstrates a rather tumultuous learning period. In 1933 surgeon Yuriy Voroniy of the Soviet Union attempted to transplant the Kidney of a diseased person into a living person, but that person died two days later due to the body's rejection over incompatible blood types[1]. In the 1950's immunosuppressive therapy became medically ready, paving the way for more successful transplant attempts. Donor/Donee pairs originally consisted of identical twins to mitigate tissue rejection risks, but these therapy improvements eventually paved the way for Family to Family transplants in the 1960's and finally altruistic/non-family transplants in the 1980s.

In the current Kidney exchange market, ⅔ of Kidneys exchanged come from deceased sources while ⅓ come from alive sources[2]. As noted by medical professionals, there are many more benefits associated with using live donation sources including quicker organ response rates, shorter surgery wait times, and convenient surgery scheduling[3]. With more living kidney exchanges taking place between unrelated people the task of maintaining compatibility between donor/donee became paramount. Algorithm architects recognized this challenge and developed techniques to manage concepts called cycles and chains in order to swap donor/donee combination pairs effectively. Cycles in this concept are very similar to the cycles seen within the housing allocation problem, while chains are a collection of dependent donor/donee pairs separated by bridging pairs[4].

This paper presents a candidate algorithm and implementation in section 2 that attempts to identify and resolve donor/donee incompatibilities, and evaluates the implementation in sections 3 and 4.

2 DESIGN

This section reasons why BNP-POLY was selected as the solution method to finding compatible kidney exchange pairings, defines the implementation for that method and provides another design concept that takes surgery scheduling failure probabilities into account.

2.1 Design Alternatives Considered

We considered cycle vs. edge Integer Programming (IP) formulations. We choose the cycle formulation with branch and price to avoid the need to hold an exponential number of variables in memory and to provide one the fastest algorithms to date for the kidney exchange problem [6]. In, particular, we considered BNP-DFS, BNP-POLY, CG-TSP, all Kidney exchange algorithms implemented by other developers. We chose BNP-POLY because it has polynomial-time pricing for cycles [5], and it allows to set the added cycle cap constrain which is the setting of our paper. The BNP-POLY method is based on an algorithm created in Glorie et al.[7].

2.2 BNP POLY Design

Using the branch-and-price approach to solve the kidney exchange problem means solving the pricing problem at every node in a branch-and-bound search tree. The pricing problem mentioned is finding a positive price cycle, set of cycles, or proving that none exist. Some implementations of branch-and-price IP solvers for the kidney exchange problem exhaustively check all the possible cycles to solve the pricing problem [8]. This what the considered branch-and-price DFS method does. However, that means doing a depth-first-search for every node in the graph which is computationally expensive, and it leads to an exponential-time pricing algorithm [5].

In order to make our implementation faster, we used a variant of the branch-and-price approach with a polynomial-time pricing algorithm developed by Glorie et al., and later modified by Plaut [5]. In order to increase efficiency solving the pricing problem, Glorie et al. reduced the problem

of generating positive price cycles to finding negative weight cycles in a directed graph. Then, they adapted the Bellman-Ford algorithm to find the negative weight cycles in the directed graph[7].

The reduced graph has the same vertices and edges as the original problem graph. However, the weights on the edges are different. They are defined as follows. Consider an edge $e = (u, v)$ which has a weight w_e in the original graph and the dual value of its vertex v is δ_v, then the reduced weight of the edge in the reduced graph is $r_e = \delta_v - w_e$ [5]. This transforms the pricing problem into a minimization problem, where a negative cycle in the reduced graph is a positive price cycle in the original graph. A negative cycle is a cycle where the sum of its weights is negative. For example, consider a path (v_1, v_2, v_3) with reduced weight r_1. If we add an edge $e = (v_1, v_3)$ with reduced weight r_2, and $r_1 + r_2 < 0$, then we have negative cycle[7].

For the pricing algorithm we have the added constraint that we are searching for cycles with length of at most L. We can adapt the Bellman-Ford to find all negative cycles with capped length starting from each vertex of the reduced graph[7]. Plaut notes the Bellman-Ford algorithm might reuse edges already included in the current path while looking for cycles[5]. Therefore, an additional check is needed to prevent this behavior. This check is implemented before updating the distance to a vertex v through an edge $e = (u, v)$. First we check through the predecessor vertices in the path to u, and we only update the distance if v is not already in the path to u. If v is already on the path, we would create a loop by reusing that edge.

The Bellman-Ford algorithm has a complexity of $O(|E|L)$, since it runs for at most $L - 1$ steps examining $O(|E|)$ edges at each step. This algorithm has to be run at each vertex, which increases the complexity to $O(|V||E|L)$. Moreover, we added a check of the predecessors to avoid reusing edges, which we need to run at each update to check $O(L)$ predecessor vertices. Therefore, the overall complexity is $O(|V||E|L^2)$.

We provide the pseudocode for this method written by Plaut in Algorithm 1. Plaut proved the correctness of this algorithm[5]. The GetNegativeCycles() method takes in the reduced graph $G = (V, E)$, maximum cycle length L, and the reduced edge weight vector w. However, in our implementation we choose to include the weights already in the graph class. The $pred_i(v)$ is the predecessor list of vertex v at step i. The $d_i(v)$ is the distance from the vertex s being considered in this iteration to vertex v at the i-th step of the Bellman-Ford algorithm. Here, distance is defined as the sum of the weights of the edges of the current path.

```
function GetNegativeCycles(G = (V, E), L, w)
    C ← ∅                                              ▷ Accumulator set for negative weight cycles
    for each s ∈ V do
        pred_0(v) = ∅  ∀v ∈ V
        d_0(s) = 0                                     ▷ Distance from source to source is zero
        d_0(v) = ∞  ∀v ≠ s ∈ V                         ▷ Distance at step 0 to other vertices is infinite
        for i ∈ {1, ..., L − 1} do
            d_i(v) = d_{i−1}(v)  ∀v ≠ s ∈ V
            pred_i(v) = pred_{i−1}(v)  ∀v ≠ s ∈ V
            for each (u, v) ∈ E do
                if v ∉ 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) ← d_{i−1}(u) + w(u, v)              ▷ Update to shorter distance
                        pred_i(v) ← (u, i − 1)                     ▷ Store correct predecessor
        for each v ≠ s ∈ V do                          ▷ Find negative weight cycles with s as the source
            if d_{L−1}(v) + w(v, s) < 0 then
                C ← C ∪ TraversePreds(v, pred, L − 1)
    return C
function TraversePreds(v, pred, n)
    c ← []                                             ▷ Start with an empty list (representing a cycle)
    curr ← v
    while curr ≠ ∅ do                                  ▷ Until we reach the source node ...
        c ← curr + c                                   ▷ Add predecessor to path
        (u, i) ← pred_n(curr)                          ▷ Get predecessor of predecessor
        curr ← u;  n ← i
    return c
```

Algorithm 1: Modified polynomial-time Bellman-Ford search for negative weight cycles [5]

2.3 Failure Aware Searching

Plaut et al. describe a method whereby the realistic probability of a failure can be accounted for in Kidney-Exchange cycles [6]. There are many reasons this could occur, whether it being a procedural delay, the deterioration of a patient's condition, or a participant receiving a donation from an alternative source. A break in a cycle means that a cycle cannot execute, and thus a new cycle cannot be found. Initially this problem was addressed with varying probabilities, however this led to a paradigm that the "most fit" candidates would have a higher chance of being returned in successful cycles. While this at first seems intuitive, it easily allows for discrimination to occur. For that reason it is better to apply a single general probability of failure to all donation edges, rather than individual ones.

To address post-match edge failure, we sought to implement the Polynomial-time Bellman-Ford failure-aware pricing for cycles algorithm 2 [6]. Due to the scope of our resources and time for this project, we were ultimately unsuccessful at generating a proper implementation for this aspect of the Kidney Exchange algorithm.

The basis of the algorithm is that it runs O(L) iterations for each source vertex - one for each possible cycle length. On each iteration, it can assume a probability of failure and remove certain edges according to that probability. We know how much the weights are, and we can use the reduced graph for the discounted pricing problem. Each cycle length and probability combines to give a different result of cycles. This assures that with a realistic probability, backup

cycles can be determined for the graph. On smaller datasets, with relatively few edges, there may be only one set of cycles which can actually create a full cover of all vertices. In the case that there are multiple solutions, that algorithm will return all of them.

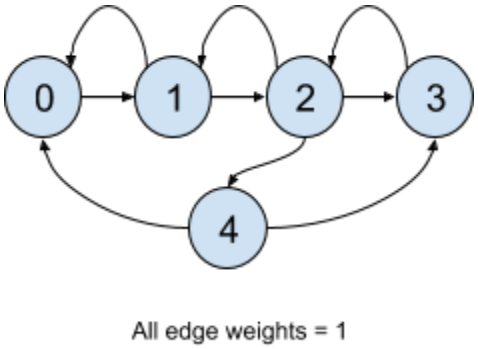

Figure 1: Determining Feasible cycles and optimal cycles

The Plaut et al paper is mostly concerned with the algorithm, proof of correctness, and efficiency results of their implementation. As such, it was left to us to understand the output generated by the algorithm and to determine the action to take with it.

Algorithm 1 returns a list of all cycles found in the graph with positive utility, or weight. It does not, however, determine what set of cycles to use for the optimal solution to the Kidney Exchange problem. For example, in Figure 1, the following cycles were returned:

S: {(1 -> 0 ->1), (2 -> 1 -> 2), (3 -> 2 -> 3), (3 -> 4 -> 2 -> 3)}

This presents a problem in that if cycle (2 -> 1 -> 2) or (3 -> 2 -> 3) were selected, a constricted graph remained, and we could not create a cycle from the graph to include all other vertices. Thus, we needed a way to reduce the set of all cycles to a list of feasible cycle sets.

To do this a method was created which iterated through each cycle as a starting point, then moved through all other cycles, adding them to the set if they contained a unique group of vertices (no vertex can be used twice). If all vertices were included after going through the other cycles, the set of cycles is added to the feasible list.

This algorithm takes a greedy approach, and adding one cycle which is not included in a feasible solution might bar the inclusion of another cycle which uses some of the same vertices but contributes to a feasible solution. To remedy this, we add another layer of iteration, meaning that we use every cycle as a starting point, then, looping through all other cycles as the first potential addition, go through all other cycles $O(N^2)$, giving a total complexity of $O(N^3)$.

The graph in figure 1 gives one set of feasible cycles:

S: [{ (3 -> 4 -> 2 -> 3), (1 -> 0 -> 1) }]

From there, we calculate the weight of each cycle based on the weight of its edges, and select the set of cycles with the highest combined weight, returning the optimal solution. It should be noted that should an edge fail as described in the section about failure aware kidney matching, we could alternately find the feasible cycle set in which the cycle containing the failing edge has the lowest weight, and use it as a starting point for reassignment.

3 EVALUATION STRATEGY

To model a Kidney Exchange network similar to that of one in real life where different donors and recipients have varying degrees of compatibility ranging from 0 (0%) to 1 (100%), we decided to create the following simplified model to generate our test data.

We consider five types of donor-recipient pairs categorized by different blood type and protein combinations. We label these different pair types by colors: BLUE, ORANGE, GREEN, RED, and BLACK. The below table maps the compatibility factors between each pair type:

	BLUE	ORANGE	GREEN	RED	BLACK
BLUE	100%	50%	20%	0%	0%
ORANGE	50%	100%	0%	70%	0%
GREEN	20%	0%	100%	60%	0%
RED	0%	70%	60%	100%	0%
BLACK	0%	0%	0%	0%	0%

The compatibility factors in the table correspond to the probability of success given a kidney transplant between a donor and recipient of the corresponding types. For example, a kidney transplant between a donor of RED type and a recipient of ORANGE type would have a 70% chance of success. In our test data, this particular exchange would be represented by an edge of weight 0.7 between the RED and ORANGE nodes. Note that each type, when matched with its own type, has a 100% chance of success except for those of type black. Nodes of type black can not be matched to anyone.

Using this model we developed test cases of varying degrees in size: 5 nodes, 25 nodes, 50 nodes, 100 nodes, and 150 nodes.

4 RESULTS

Algorithm 1 successfully finds all cycles in a graph in polynomial time. This is a vast improvement to many other strategies for finding cycles. Because in real settings the datasets

are often very sizable, this is a beneficial advancement. In our solution, the limiting factor became the selection of which cycles to include in an optimal solution. Given more time, a better implementation could have helped speed this process up, but our current implementation has a $O(n^3)$ complexity.

To find the failure aware pricing, the time complexity of the algorithm is increased by a factor of L, the max length of cycle allowed in a solution. This algorithm also produced all cycles, but on a small dataset such as the ones we had available, our algorithm did not produce an output unique to the basic negative cycle algorithm. Given more time and resources to spend on the project, we could alter the implementation to ensure that small datasets were not an exception.

Additionally, the formation of the linear problem for the input graph was a massive task for any datasets larger than single digits. This meant that the size of sample graph we could process with our implementation was nowhere near the size done in the Plaut paper. That being said, for small graphs, the algorithm gave the optimal solution in a negligible amount of time.

With more time, better hardware, and more resources, this implementation could faithfully recreate the results of the paper in which the algorithms and theories were first presented.

5 CONCLUSION

The Kidney exchange problem is an important problem to solve in the context of facilitating better medical care. In our approach towards implementing a BNP Poly design we found functional success in identifying cycles, and selecting cycles that belonged to the optimal solution. Non-functionally there were challenges associated with producing a more efficient implementation, with a certainly improvable time complexity. In addition to the primary algorithm as the focus to this paper we also developed a technique to find failure aware pricing, but encountered issues during testing that required more hardware resources.

REFERENCES

[1]: Matevossian, E. , Kern, H. , Hüser, N. , Doll, D. , Snopok, Y. , Nährig, J. , Altomonte, J. , Sinicina, I. , Friess, H. and Thorban, S. (2009), Surgeon Yurii Voronoy (1895–1961) – a pioneer in the history of clinical transplantation: in Memoriam at the 75th Anniversary of the First Human Kidney Transplantation. Transplant International, 22: 1132-1139. doi:10.1111/j.1432-2277.2009.00986.x

[2]: Kidney Exchange Facts
https://www.mayoclinic.org/tests-procedures/living-donor-kidney-transplant/pyc-20384838

[3]: Live Kidney Exchange Benefits
https://transplant.surgery.ucsf.edu/conditions--procedures/living-donor-kidney-transplant.aspx

[4]: Chain Figure
https://stanfordhealthcare.org/medical-treatments/k/kidney-transplant-surgery/types/chain-donation.html

[5]: Plaut, Benjamin. 2016. Algorithms for Social Good: Kidney Exchange. School of Computer Science Honors Undergraduate Research Thesis. Carnegie Mellon University.

[6]: John P. Dickerson, David F. Manlove, Benjamin Plaut, Tuomas Sandholm, and James Trimble, 2016. Position-Indexed Formulations for Kidney Exchange. ACM V, N, Article A (January YYYY), 18 pages. DOI = 10.1145/2940716.2940759 http://doi.acm.org/10.1145/2940716.2940759

[7]: Kristiaan M. Glorie, J. Joris van de Klundert, 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 16(4):498-512. https://doi.org/10.1287/msom.2014.0496
[8]: 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.