CS 333 – Algorithm Analysis

Spring 2020

Project Report

Group members: Can Yılankıran, Gökberk Aslantaş, Kıvanç Yılmaz,

Title: Clearing Algorithms for Barter Exchange Markets: Enabling Nationwide Kidney Exchanges

ABSTRACT

The problem addressed in this article is about the patients who have kidney disease, need treatment like kidney transplant, but there is a long waiting list for those kidneys and the demand for kidneys surpasses the supply of those kidneys. For instance, 4,052 people lost their lives in that waiting process in the United States in 2005. To overcome this problem, patients may choose to find a living donor who volunteers to denote her/his kidney. The issue with those donations is some of the potential donors and his/her kidney may be incompatible with the patient in terms of blood-type or tissue-type. This issue lowers the number of live donations, however these days patients with the kidney disease swaps their incompatible donors to obtain the kidney which is compatible for them. This exchange process can be named as barter exchange. In barter exchange agents try to swap their items with each other in order to increase their productivity. In our case, agents are patients and items are incompatible donors. These exchange processes include cycles of patients(agents), each of them obtaining the next incompatible donor(item) of the next patient in the cycle. Developing an optimized clearing algorithm for barter exchange markets to get a national kidney-exchange market, will reduce deaths which are caused by kidney disease, but in order to get that solving the clearing problem is necessary. The goal in the clearing problem is trying to find a social welfare which has maximum exchange with the restriction that the maximum length of a cycle is fixed, the goal is trying to find a social welfare which has maximum exchange. The reason that long cycles is prohibited, all transplant operations in the cycle must be done simultaneously. The main challenge appears when we try to build a national kidney exchange organization, this clearing problem with cycle-length restriction is NP-hard. While constructing this algorithm the key is using incremental problem formulation. We will adapt two models to the algorithm which are constraint generation and column generation to accomplish the task. To improve runtime and memory usage, developed methods are used for each model. After the tests, we obtained that column generation scales strongly better than constraint generation.

1. Introduction

Kidney failure is one of the most important diseases seen in our age. Kidney failure is a loss of function of the kidneys for various reasons. In addition to diseases like diabetes, hypertension, drugs and alcohol also result in kidney failure. There are two remedies for kidney failure. The first is that the patient enters the dialysis machine 2 or 3 times a week. Dialysis machine is a machine developed for patients suffering from kidney failure, which allows the dirty blood taken from the patient to be cleaned and returned to the patient. When this is said, even though this machine looks like an angel of goodness, it actually has many difficulties. One of the biggest challenges is that the patient spends a long time in the dialysis machine and repeats it several days a week.In fact many patients prefer to withdraw from dialysis because the quality of life on dialysis can be highly low, leading to a natural death. Only 12% of dialysis patients survive 10 years [6].

The second is to transfer a healthy kidney from a suitable donor to the sick person. Kidney transplants are by far the most common transplant. Unfortunately, the demand for kidneys far outstrips supply. For example in the United States in 2005, 4,052 people died waiting for a life-saving kidney transplant. During this time, almost 30,000 people were added to the national waiting list, while only 9,913 people left the list after receiving a deceased donor kidney. The waiting list currently has over 70,000 people, and the median waiting time ranges from 2 to 5 years, depending on blood type. This number has increased as we come to 2011 [1]. Kidney transplants can come from both deceased and living donors. Deceased donor kidneys are allocated to patients by means of a waiting list, which in the US currently contains 108,571 patients and has an average waiting time of 4 years [5].Unfortunately, the number of kidneys available for transplantation is still largely insufficient to meet demand: only about 17,000 US patients can receive a transplant each year [5].

For many patients with kidney disease, the best preference is to find a living donor, that is, a healthy person willing to donate one of his/her two kidneys. Although there are marketplaces for buying and selling living-donor kidneys, the commercialization of human organs is almost universally regarded as unethical, and the practice is often explicitly illegal, such as in the US. However, in most countries, live donation is legal, provided it occurs as a gift with no financial compensation. In 2005, there were 6,563 live donations in the US [1]. This number of donations was not enough at the time, and is still not enough today. In 2014, 17,107 kidney transplants took place in the US. Of these, 11,570 came from deceased donors and 5,537 came from living donors[7]. As seen in the examples, while the number of live donors should increase, unfortunately it has decreased over the years.

Now we will focus on barter exchange markets. Agents (patients) want to exchange their items (incompatible donors) with each other. These exchange processes include cycles of patients (agents), each of them obtaining the next incompatible donor (item) of the next patient in the cycle. These exchanges can be seen in every field of life. To give some examples there are some websites to trade books PaperBackSwap [2], and BookMooch[3]. Another website for trading is TradeStuff [4], which allows agents to swap items for their own desire.

While encoding a barter exchange market we used directed graph as $G = (V, E)$. For each agent a vertex is constructed. If v_x seeks to get the item of v_y, we add a weighted edge e from agent v_x to v_y. This weighted edge shows the benefit to v_x of getting the v_y's item. If a cycle c occurs in the graph, that means a possible swap opportunity has been formed. In this cycle each agent gets the next agents' item. The sum of edge weights in the cycle c is represented as w_c. The collection of disjoint cycles is an exchange. The sum of its cycle weights is the weight of an exchange.

In figure 1 we showed an example of a market with 5 agents, $\{v_1, v_2, v_3, ..., v_5\}$, and all edges $\{e_1, e_2, e_3, ..., e_8\}$ have weight 1. This market example has 4 cycles, $c_1 = (v_1, v_2)$, $c_2 = (v_2, v_3)$, $c_1 = (v_3, v_4)$ and $c_4 = (v_1, v_2, v_3, v_4, v_5)$, and there is two maximal exchanges, which are $M_1 = \{c_4\}$ and $M_2 = \{c_1, c_3\}$. In this figure, M_1 includes the most edges. Also it has both maximum height and maximum cardinality.

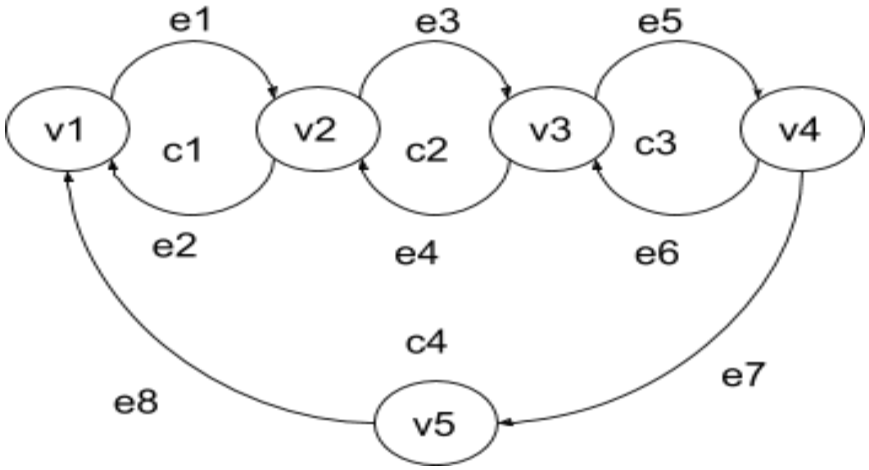

Figure 1: An example of barter exchange market.

Now we will move on to the clearing problem, to goal is finding a maximum-weight exchange consisting of cycles with length at most some small constant L. This cycle-length limitation increases for several reasons. In our case, in a kidney exchange, performing the operations in a cycle needs to be done

simultaneously. That means a donor might back down from after his incompatible partner has received a kidney. That's why these operations need to be done at the same time. Even if all the donors are operated at the same time, a k-cycle requires between $3k$ and $6k$ doctors, around $4k$ nurses, and $2k$ operating rooms.

Under such a resource limitation, the national kidney exchange market will allow only cycles of length 2 and 3. If we use these short cycles there are some advantages as well. When there is a failure fewer patients (agents) are affected. There can be a case like before the operation of the transplant, in the last-minute testing can reveal that new incompatibility that was not detected in the initial testing. To make it a more general expression, a patient (agent) might back down from out of a cycle when his/her preferences have changed. Considering the trading website which we gave as an example of a barter exchange market, if an agent forgets to send the item which is offered to be transferred to the other agent, that will drop him/her out of the cycle.

We will also show why is the clearing algorithm NP-complete for $L \geq 3$ Section 3. This algorithm is based on an integer-linear program (ILP) formulation, which is solved using specialized tree search, for 2 reasons.

1) First, if there is a loss of optimality that could lead to unnecessary patient deaths.

2) Second, an attractive feature of using an ILP formulation is that it allows one to easily model a number of variations on the objective, and to add additional constraints to the problem. For example, if 3-cycles are believed to be more likely to fail than 2-cycles, then one can simply give them a weight that is appropriately lower than 3/2 the weight of a 2-cycle. Or, if for various (e.g., ethical) reasons one requires a maximum cardinality exchange, one can at least in a second pass find the solution (out of all maximum cardinality solutions) that has the fewest 3-cycles. Other variations one can solve for include finding various forms of “fault tolerant” (non-disjoint) collections of cycles in the event that certain pairs that were thought to be compatible turn out to be incompatible after all.[1, p 2]

Further in this paper, we will see the details of this algorithm which is capable of clearing these markets. While constructing this algorithm the key is using incremental problem formulation. Adapting two models to the algorithm which are constraint generation and column generation for the task. To improve runtime and memory usage, developed methods are used for each model.

In Section 2, the market clearing decision problem is proven to be NP-complete. Sections 3 and 4 each contain an ILP formulation of the clearing problem.

2. Background

We will prove that the market clearing problem with short cycles is NP-complete. We need to give some explanation about NP-hard and NP to understand NP-complete problems. A problem is assigned to the NP class if it is solvable in polynomial time by a nondeterministic Turing machine[10]. A problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H. Let's assume that problem H's solutions takes 1 unit time, H's solution can be used to solve L in polynomial time [8]. Now we can define what NP-complete problems means. If a problem is both in NP and NP-hard, this problem is NP-complete [9]. The following theorem is proven in [1].

Theorem 1. Given a graph $G = (V, E)$ and a integer $L \geq 3$, the problem of deciding if G admits a perfect cycle cover containing cycles of length at most L is NP-complete.

Proof. They claimed that this problem is clearly in NP. When it comes to proving this problem is NP-hard, they tried to reduce from 3D-Matching, which is a NP-complete problem of, given disjoint sets X, Y , Z of size q , and a set of triples $T \subseteq X \times Y \times Z$, deciding if there is a disjoint subset M of T size q. [1] So we are trying to a disjoint subset of T which is $T \subseteq X \times Y \times Z$, with size q.

The first idea was to construct a tripartite graph with vertex sets $X \cup Y \cup Z$ and directed edges $(x_a, y_b), (y_b, z_c)$ and (z_c, x_a) for each triple $t_i = \{x_a, y_b, z_c\} \varepsilon T.$ But they realized that this encoding fails because a perfect cover might include a cycle without corresponding triple.

Instead we used the following reduction. Given an instance of 3D-Matching, add one vertex for each element in X, Y and Z. For each triple, $t_i = \{x_a, y_b, z_c\} \varepsilon T$ gadget in Figure 2. They showed that the gadgets intersect only on vertices $X \cup Y \cup Z$. This construction can be done in polynomial time.

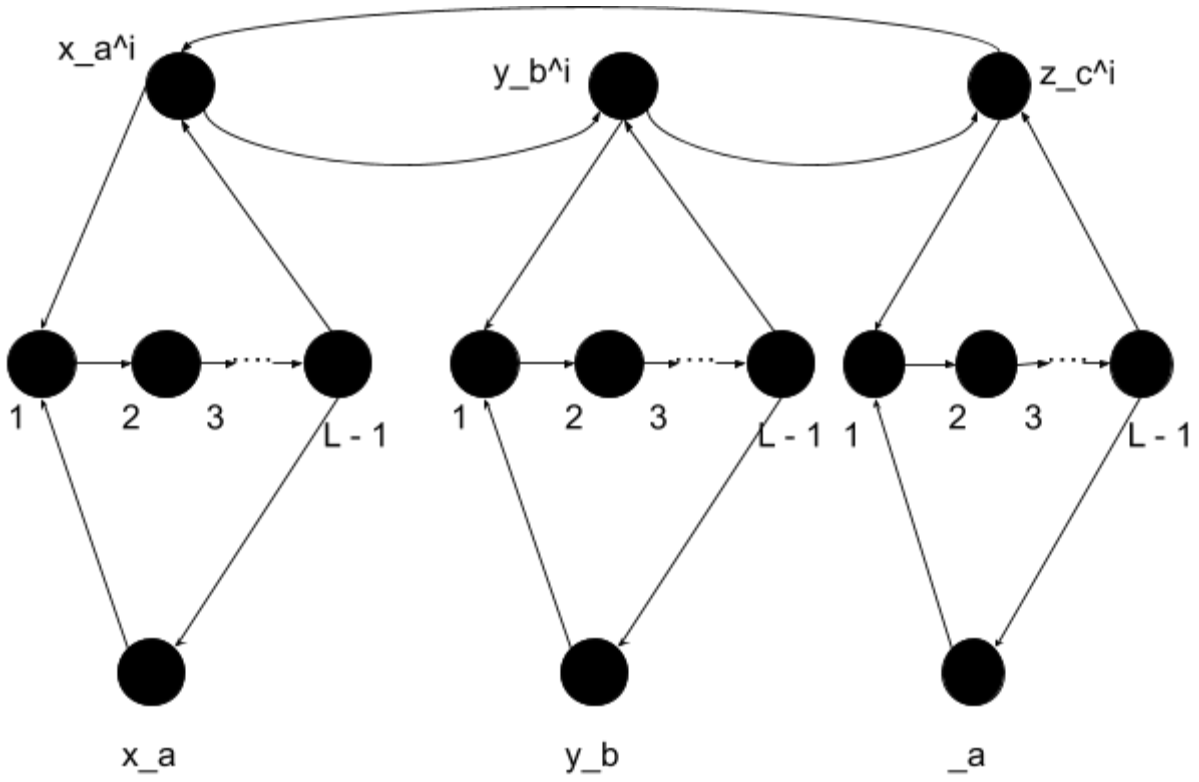

Figure 2 : NP-completeness gadget for triple t_i and maximum cycle length L.

3. SOLUTION BASED ON AN EDGE FORMULATION

In this section to get a solution, we used ILP (integer linear programming) formulation of the clearing problem with one variable for each edge. In Section 1, given a market $G = (V, E)$, now our goal is to get a perfect matching in a bipartite graph. To achieve that we add an edge between the patient (agent) and with his own incompatible donor (item). Now we have got a perfect matching. Let's add edge e with w_e between agent v_i and the item which he/she seeks to obtain. We can call it as v_j's item. When v_j's item is taken by v_i, v_j needs to obtain another patients' (agents) item. Solving his unlimited clearing problem is possible now by finding a maximum-weight perfect matching. Figure 3 is the bipartite graph encoding of Figure 1. In Figure 3 the dashed lines illustrate edges with zero weight. Bold edges indicates the swapping process.

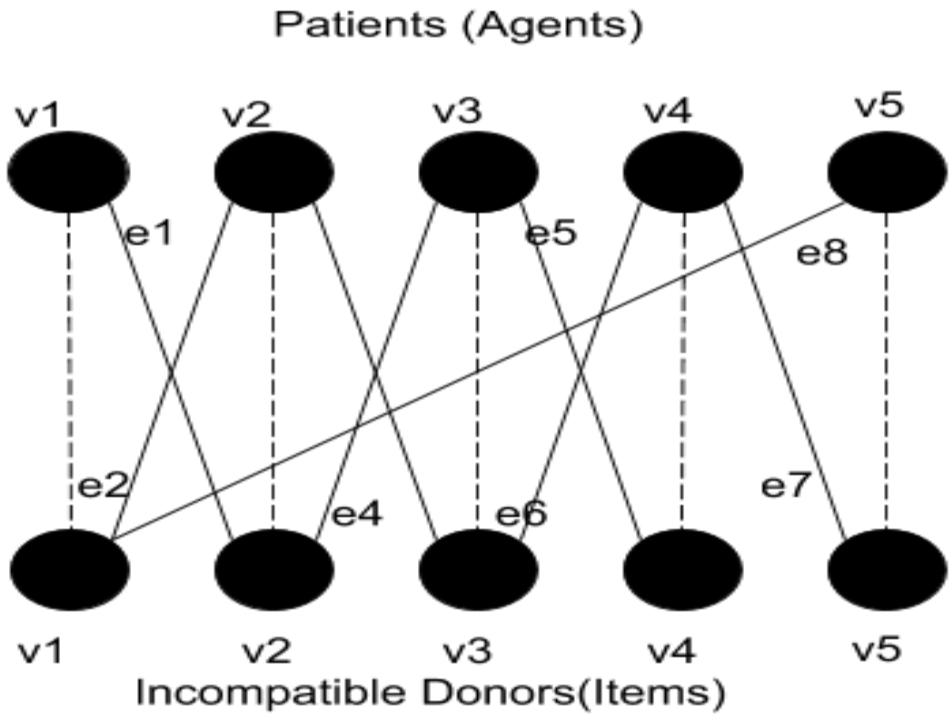

Figure 3 : Perfect matching encoding of the market in Figure 1.

There is an alternative way to solve the problem which is encoding it as an ILP like we did at first but we are going to use original market graph G, to deal with cycle length limitations.

$$max \sum_{e \varepsilon E} w_e e$$

When we consider the process of the swapping items between agents, the edges going outside from each agent to the other agent needs to be equal to edges which are coming in from the other agent.

$$\sum_{e_{out}=(v_i,v_j)} e_{out} - \sum_{e_{in}=(v_j,v_i)} e_{in} = 0$$

And an agent can't do 2 swapping in our case.

$$\sum_{e_{out}=(v_i,v_j)} e_{out} \leq 1$$

When the cycles are allowed to have length at most T, adding constraint $e_{p1} + e_{p2} + ... + e_{pL} \leq L - 1$, for each length-L path $p = \left(e_{p1}, e_{p2}, ... , e_{pL}\right)$ will do the work. But this approach fails, in a market of only 1000 patients, the number of length-3 paths exceeds 400 million, that means this ILP formulation cannot even be constructed without running out of memory.

Therefore, using a tree search with an incremental formulation approach. We use CPLEX for this operation. The CPLEX optimizer is used to solve integer programming problems and very large linear programming problems. We will begin with only a small subset of the constraints in the ILP. When this ILP becomes small, it is solvable by CPLEX. Then, we will check wherever any of the missing constraints are violated by the fractional solution. If there are some constraints under this condition, we add them to the ILP and repeat this operation. When all constraints are satisfied, there may be no integral solution matching the fractional upper bound or LP solver might not find it.

In these cases, CPLEX branches on a variable, and a new search node is generated for all corresponding children. At each node of the search tree is visited, these operations are repeated. This approach yields an optimal solution once tree search finishes. We need to determine which constraints to begin with and which violated constraints to include. We will describe them in the following subsections.

3.1 Constraint Seeder

Constraint seeder determines which constraints to begin with. We forbade any path of length $L - 1$. So that there is not an edge closing the cycle from its head to its tail. Finding these constraints is computationally expensive.

3.2 Constraint Generation

In the first constraint generator, our goal was to search for length-L paths with value sum more than $L - 1$. For those paths, we set their value sum to be at most $L - 1$.

In the second constraint generator, for the cycles which have the sum of edges less than $[|c|(L-1)/L]$, we add one constraint.

After this we detect that this generator makes the algorithm slower. So as a final generator, we came up with the following way. It adds one constraint per violating path p. Also, for each path with the same interior vertices without counting their endpoints, the generator adds a constraint. This made the algorithm faster.

3.3 Performance After These Improvements

Even with those improvements, the edge formulation approach cannot clear a kidney exchange with 100 vertices. So a constraint generation based approach cannot be a feasible solution. For that reason, in the rest of paper, we will focus on the cycle formulation and the column generation based approaches.

4. SOLUTION BASED ON A CYCLE FORMULATION

In this section we are using ILP formulation as we did in section 3, but the only difference is instead of edge we will use cycles. Like we did at section 3, consider a market $G = (V, E)$, we only take cycle c of length 2 and using them to make a graph. Therefore, the market clearing problem with $L =$ 2 can be solved in polynomial time by finding a maximum-weight matching.

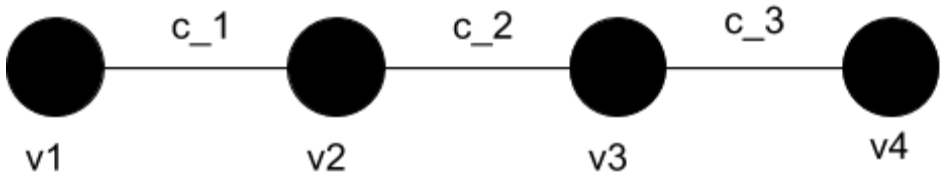

Figure 4 : Maximum weight matching encoding of the Figure 1.

With an arbitrary L to generalize this encoding, let $C(L)$ be the set of all cycles of G with length at most L, this ILP finds the maximum-weight cycle in the $C(L)$.

$$max \sum_{c \varepsilon C(L)} w_c c$$

$$subject\ to \sum_{c:v_i \varepsilon c} c \leq 1 \quad \forall v_i \varepsilon V$$

$$with\ c\ \varepsilon\ \{0,1\} \quad \forall c\ \varepsilon\ C(L)$$

4.1 Column Generation for the LP

We used an incremental formulation problem to avoid the problem of running out of memory.

The first step in LP-guided tree search is to solve the LP relaxation. This will fail because cycle formulation does not fit in memory. With the observation that an exchange solution can include tiny friction of cycles, we found an approach of using column (in our case cycle) generation.

We will start with a restricted LP that has only a small number of columns (cycles). Then repeatedly add columns until an optimal solution to this formulated LP is an optimal solution to the master (original) LP.

A corresponding LP with $L = 2$ in figure 1.

$$Primal\ P$$

$$max \quad 2c_1 + 2c_2 + 2c_3$$

$$\begin{aligned} subject\ to \quad & c_1 && \leq 1 \ |(v_1) \\ & c_1 + c_2 && \leq 1 \ |(v_2) \\ & + c_2 + c_3 && \leq 1 \ |(v_3) \\ & c_3 && \leq 1 \ |(v_4) \end{aligned}$$

$$with \quad c_1, c_2, c_3 \geq 0$$

$$Dual\ D$$

$$min \quad v_1 + v_2 + v_3 + v_4$$

$$subject\ to \quad v_1 + v_2 \geq 2 \ |(c_1)$$

$$v_2 + v_3 \quad \geq 2 \quad |(c_2)$$

$$v_3 + v_4 \quad \geq 2 \quad |(c_3)$$

$$\textit{with} \quad v_1, \; v_2, \; v_3, \; v_4 \quad \geq 0$$

Figure 5: LP formulation

Let P' be the restriction of P containing columns c_1 and c_3 only. Let D' be the dual of P' that is, D' is just D without the constraint c_2. Because P' and D' are small, we can solve them to obtain $OPT(P') = OPT(D') = 4$, with $c_{OPT(P')} = [c_1 = c_3 = 1]$ and $v_{OPT(D')} = [v_1 = v_2 = v_3 = v_4 = 1]$.

In general we can see that $OPT(D) = OPT(D')$, but we cannot prove this because D and P are too large to solve. Instead, because constraint c_2 is violated, we add column c_2 to P, update D', and repeat. The problem of finding a violated constraint is called the pricing problem [1].

4.1.1 Pricing Problem

Maintaining a collection of all feasible cycles is possible when we try to find them for smaller instances. By this approach pricing problems are easy and efficient to solve. To solve it we will traverse this collection of cycles and try to find cycles with positive prices. Unfortunately this approach doesn't scale the way we need. Consider a market which, has 5000 patients, can have 400 million cycles of length at most 3. We can't keep that many cycles in memory. Thus, when we work with larger instances, we need to find feasible cycles while looking for a cycle with a positive price. Depth-first search algorithm is used on the market graph at Figure 1. Exploring the vertices in non-decreasing order will make the search faster. Because these vertices have a greater chance to belong to cycles with positive weight. We added some extra rules

obtaining that the current path can lead to a positive weighted cycle or not. Even with these rules, column generation is a bottleneck. So we added the following optimizations.

When a vertex is proven to not belong to any positive price cycle, we mark the vertex and do not use it as a root for depth-first search. That reduces the unnecessary repetition.

4.1.2 Column Seeding

While column seeding trying to improve the objective value of the restirected LP, each of these iterations cost us very expensively. Because we must solve the pricing problem, then re-solve the restricted LP. Seeding the LP with enough columns to achieve an objective value is not too far from the master LP will be much faster. But, of course we cannot add so many columns that will make us run out from memory.

We experimented with some column seeders. At last we find our best column seeder. It constructs a random collection of feasible cycles. A market with 5000 patients can have 400 million feasible cycles. Of course we cannot generate and traverse all feasible cycles. Trying this will be too long. Instead of that, we can perform a random walk on the market graph (see figure 1). At every step of the walk, we test if there is an edge back onto our path that forms a feasible cycle or not. If the answer is yes, it will be included in the restricted LP. Then we will do the same operation again. In our experiments, we will use this algorithm to seed LP with 400,000 cycles.

4.1.3 Proving Optimality

Our goal is finding an optimal solution to the master LP relaxation. If we use column generation, proving that a restricted-primal solution is optimal, is possible under one condition. All columns must have non-positive prices. But, our clearing problem has the tailing-off effect. Tailing-off effect means that to prove optimality we need a large number of additional iterations. Like eliminating all positive-price columns. There is no feasible solution to the tailing-off effect. But, we know that the unrestricted clearing problem can be solved in polynomial time. Using that we can find an upper bound for master LP relaxation. That gives us some power when we try to prove optimality. Because when the restricted primal reached the upper bound, we also proved the optimality. Even without eliminating all positive-price columns. This made our algorithm faster.

4.1.4 Column Management

Assume that optimal value of initial restricted LP is P' and the master LP is P. If this P' is far from the P we need a large number of columns to close the gap between them. For instance this can bring up some issues on markets with 4,000 patients or more. With that memory issue, we also consider the problem that column iterations become slow while solving a larger LP.

To solve these issues, we need to set a limitation for the size of restricted LP. Whenever we add columns to the LP, we need to check the threshold number of columns to see it contains more or less. If we clarify that it has more columns than threshold number, we select some columns to remove until it is less than threshold number. By the way, the threshold number is set to 400,000, based on the memory size. We need to be careful while determining the threshold number. If we try to balance the performance gains based on per-iteration, it will increase the number of iterations needed to get an appropriate column set in the LP.

We never delete columns with a non-zero LP value, we delete the ones with the lowest price. This column management approach works really well, it gave us the opportunity to clear markets with 10,000 patients in it.

5.3 Branch-and-Price Search for the ILP

With the column generation improvements listed above, we can successfully solve a LP relaxation to optimality for a large market. Unfortunately, the solutions we came up with are usually fractional. To find an optimal integral solution we will use branch-and-price tree search.

The idea behind branch-and-price is, we will set some fractional variables to 0 or 1 this is called brach. After this operation, the master LP and the restriction we are working with are changed. After that, to prove that this changed restriction is optimal for master LP, we must apply column generation at each node of the tree.

We can avoid the expensive node pricing step by; when the changed restricted LP has the same optimal value as its parent in the tree search, we can easily prove the LP optimality as we did in section 4.1.3. Also we don't have to include any extra columns to the restricted LP.

5.3.1 Primal Heuristics

We use some primal heuristics to obtain a feasible integral solution before branching a fractional variable. The solutions we constructed are the lower bounds for the final optimal integral solutions. Mentioning that a primal heuristic should be efficient and find some tight lower bounds, to call it an effective heuristic.

We used two primal heuristics to see the results. First one we used is a simple rounding algorithm. It includes all cycles with a fractional value at least 0.5 then it adds other cycles in a greedy way. It also ensures feasibility. While we were experimenting this heuristic we found that the lower bounds did not really enable us to prune. Pruning is a technique used to reduce the size of a tree by removing some sections of the tree.

We also used CPLEX as a primal heuristic. It is possible to convert any node in the restricted LP to an ILP by reintroducing the integrality constraints. In CPLEX there are some built-in primal heuristics which can be applied to this ILP. Using CPLEX's own tree search we can find an optimal integral solution. We observed that this tree search is much faster than our own way. When CPLEX finds an integral solution equal to the upper bound at the root node of the tree, that shows we are done. If it can't find any integral solution that means there is no integral solution exists, or the cycles in the restricted LP are not in the right combination. We see that kidney-exchange markets, the second one is more possible. Thus, while we add more columns to the tree search as a result of branching, CPLEX heuristic will eventually find an optimal integral solution.

CPLEX tree search is faster but it is not enough to apply it to every node in our search tree. So we did some enhancements to make it faster. Firstly, we add a constraint to make sure that the objective value of ILP has to be large as the fractional target. Otherwise, we want to abort and keep generating more columns with our branch-and-price search. Secondly, if we found that no integral solution exists, CPLEX takes a very long time to prove that. In order to avoid this kind of time waste, we set a limit of the number of nodes in CPLEX's search tree. Eventually, if a node has a different set of cycles from its parent, we apply the CPLEX heuristic at that node.

The benefit of using CPLEX as a primal heuristic is it makes the search tree smaller. That avoids expensive pricing work at nodes which are not generated in this smaller tree. So it makes the tree smaller, this eliminates the expensive pricing operations on the nodes which are removed from the tree when we use CPLEX primal heuristic.

5.3.1 Cycle Brancher

We tried two branching strategies, both of them selects one variable per node. In the first strategy, it picks a variable from those whose LP value is closest to 1 randomly.

It is called branching by certainty. The second one, branching by uncertainty, does the same operation as the first one but this one uses a value of 0.5 instead of 1. In either case, two subtrees are generated by the two children of the node. One of the substrees variable is set to 0, the other one is set to 1. Our depth-first search always chooses the subtree, which has the value of the variable that is closest to its fractional value, to be explored.

For the clearing problem, we found that branching with uncertainty is superior to branching by certainty, and it rarely requires backtracking.

6. Discussion

We used CPLEX 10.010 for solving the LP and as a primal heuristic. We generated 10 markets and tried to clear them using CPLEX's algorithm and our algorithm. The CPLEX's algorithm is on the full cycle formulation and it cannot clear any markets with 1000 patients or more. In addition to this, the runtime on the markets which are smaller than 1000 patients, is worse than our algorithm. We will show the settings we used in our algorithm in the following table.

Category	**Setting**
Column Seeder	Combination of greedy exchange and maximum-weight matching heuristics, and random walk seeder (400,000 cycles).
Column Generation	One column at a time.
Column Management	On, with 400,000 column limitation.
Optimality Prover	On.
Primal Heuristic	Rounding & CPLEX tree search.
Branching Rule	Uncertainty.

With this combination of these optimization allows us to easily clear markets with over 10,000 patients between 3000 and 3250 seconds. CPLEX is able to clean a market with 1000 patients between 2000 and 2500 seconds. We also turned off some optimization to see the outcomes. When we restrict the seeder so that begins with

10,000 cycles. We observed that this setting is faster for smaller markets. Because smaller LP relaxations are solved fastly. This algorithm clearied a market with 4000 patients faster than the algorithm which all optimizations are on. But after 5000 vertices, these effects start to be offset by the additional column generation that must be performed. So for larger instances than 6000 patients, the restricted seeder is clearly worse.

Secondly, we restored the settings of column seerder, but we removed the optimality prover this time. When we compare it with our optimized algorithm, we see that we manage to clear a market with 10,000 patients using our optimized algorithm in about the same time with the algorithm which doesn't have optimality prover, cleared a market with 6000 patients.

7. Conclusion and Future Work

In this we have accomplished to develop scalable exact algorithms for barter exchange, with a special focus on the upcoming national kidney-exchange market. In this market patients with kidney disease will be able to match with compatible donors by swapping their own willing but incompatible donors. With over 70,000 patients already waiting for a cadaver kidney in the US, this market is seen as the only ethical way to significantly reduce the 4,000 deaths per year attributed to kidney disease. [1]

This algorithm is fully capable of clearing these kidney-exchange markets on a nationwide scale. It can optimally solve the kidney exchange clearing problem with 10,000 donor-donee pairs. Comparing the best prior technology (vanilla CPLEX) which cannot even handle instances larger than 900 donor-donee pairs, it is a breakthrough. The main problem of CPLEX is, it runs out of memory. We imporeved our algorithm based on incremental problem formulation. We used two methods for the task which are constraint generation and column generation. For each of them we used some techniques to improve the runtime and memory usage. After that we reach the observation that column generation scales way better than constraint generation. For column generation we did some improvements which are pricing techniques, column seeding techniques, some techniques for proving optimality and lastly column removal techniques. We also did some enhancements for the branch-and-price search. It includes primal heuristics and we also compared and

saw the differences between branching strategies. We observed that branching with uncertainty is superior to branching by certainty.

This algorithm has some generalizations for kidney-exchange markets. These generalizations include multiple alternative donors per patient, weighted edges in the market graph (weights are generated by degrees of compatibility, patient age and weight etc..) and additional issues. This kidney-exchange market powered by our algorithm will definitely decrease the ratio of deaths caused by kidney disease.

8. References

[1]D. Abraham, A. Blum, and T. Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. ACM EC, (07), 2007.

[2] “Trade Used Books with PaperBack Swap (the World’s Largest Book Club).” *Www.Paperbackswap.Com*, www.paperbackswap.com/index.php.

[3] “BookMooch: Trade Your Books with Other People.” *Bookmooch.Com*, www.bookmooch.com/. Accessed 5 June 2020.

[4] “TradeStuff - Secure, Efficient P2P Barter on Blockchain.” *Trade Stuff*, www.tradestuff.com/. Accessed 5 June 2020.

[5]“Find and Compare Transplant Programs.” *Srtr.Org*, 2019, www.srtr.org/. Accessed 5 June 2020.

[6] “USRDS Database.” *Www.Usrds.Org*, www.usrds.org/. Accessed 5 June 2020.

[7] Health Resources & Services Administration (HRSA). https://www.hrsa.gov/ Accessed 5 June 2020.

[8] “NP-Hardness.” *Wikipedia*, 19 May 2020, https://en.wikipedia.org/wiki/NP-hardness Accessed 5 June 2020.

[9]“NP-Completeness.” *Wikipedia*, 31 May 2020, https://en.wikipedia.org/wiki/NP-completeness Accessed 5 June 2020.

[10]Weisstein, Eric W. "NP-Problem." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/NP-Problem.html