Kidney Exchange

with Distributed and Incremental Settings

Michael Romer, Jeff Nelson, Jeff Ng, Utkarsh Vardan

INTRODUCTION

- First US Kidney Exchange surgery was done in 2000.
- First kidney exchange was organised by a patient donor pair meeting in dialysis room.
- Kidney Exchange algorithm was implemented for pairwise exchange in 2005.
- Kidney Exchange is the standard form of transplantation in the U.S.

KIDNEY DISEASE

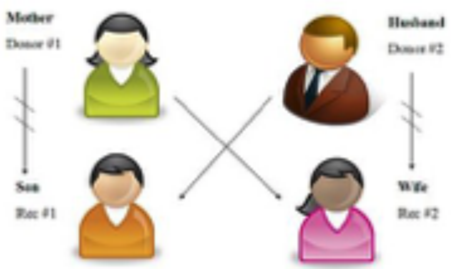
- Kidney is unable to properly filter blood.
- Increases waste in the body.
- Can cause health problems like heart attack, diabetes and high blood pressure.
- Symptoms are developed slowly.
- 14% of the U.S. population has CKD.
- Diagnosed by lab test.

Treatment

- Dialysis
- Transplantation

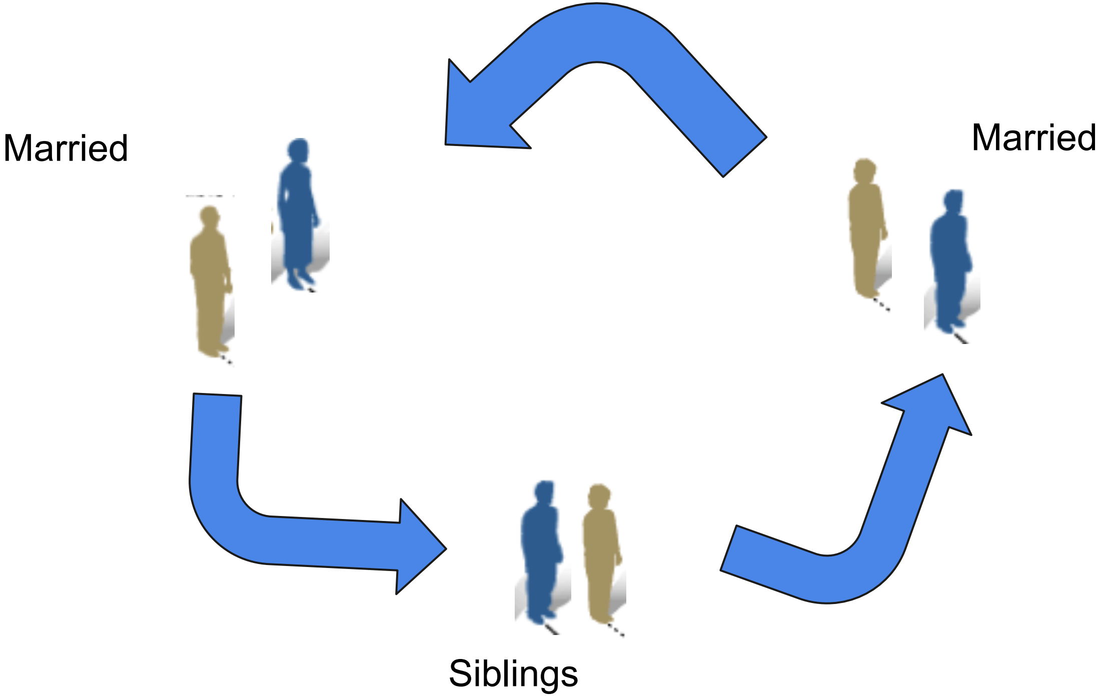
WHAT IS KIDNEY EXCHANGE?

- Kidney Exchange is also known as kidney swap. When a living kidney donor is incompatible with the receiver so it exchanges kidney with another donor receiver pair.



- In general this can be an n-way swap or trading cycle between n donor and n receiver

3-way exchange



CHALLENGES

- 100,000 people are on the waiting list for Kidney Transplant.
- The median waiting list is from 2 to 6 years.
- Only 16,000 transplants are done every year.
- Hospital doesn't have enough resources to perform more surgeries.

Our Approach

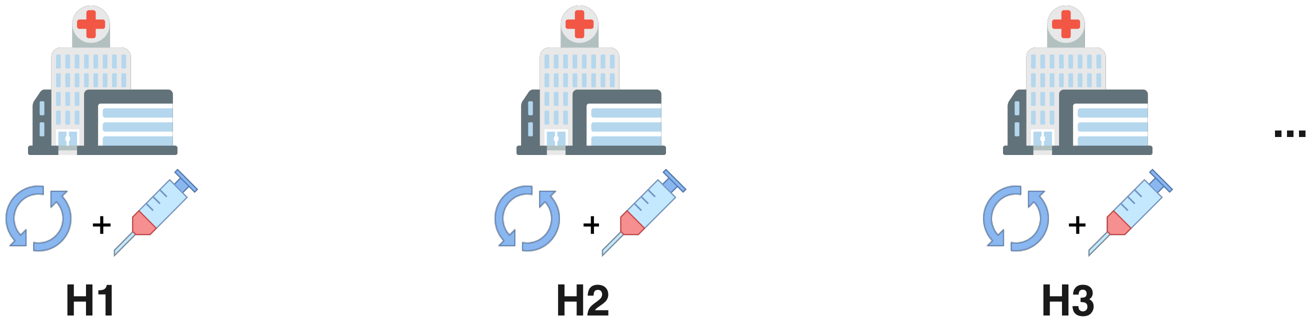
- General solution for KEP
- Extend it with distributed & incremental settings
- **Distributed setting**
  - Sharing information about unmatched patient-donor pairs
  - Transferring patient-donor pairs to higher ranked hospital if matched
- **Incremental setting**
  - Randomly add or remove patient-donor pairs
- 2 different programming domains
  - **Greedy matching**
  - **Matching via Integer Linear programming**

Model settings

- More than 1 ranked hospital
- Hospital
    - List of patient-donor pairs
    - Limited # of surgeries that can be performed in a single time step
    - Contains only local information initially
- Hospitals share info to the immediately higher ranked one
- Configurable # of rounds to run KE

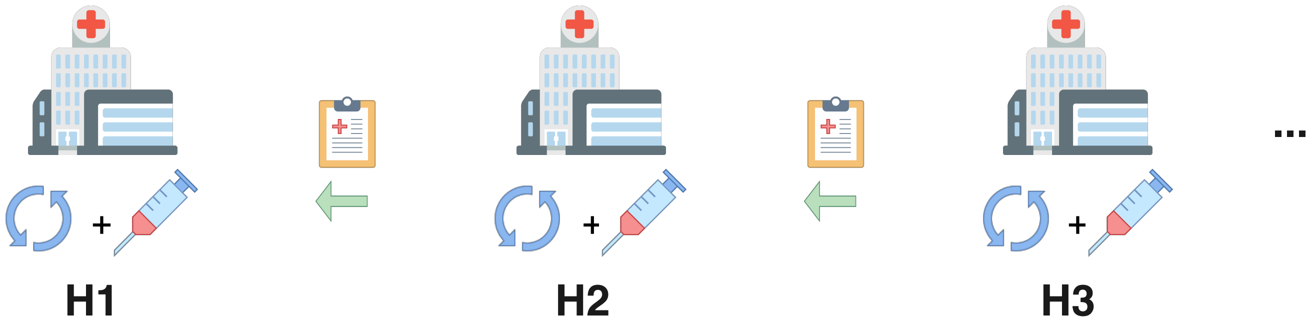
Exchange Workflow (t = 1)

- local matching
- perform surgeries then remove matched pairs



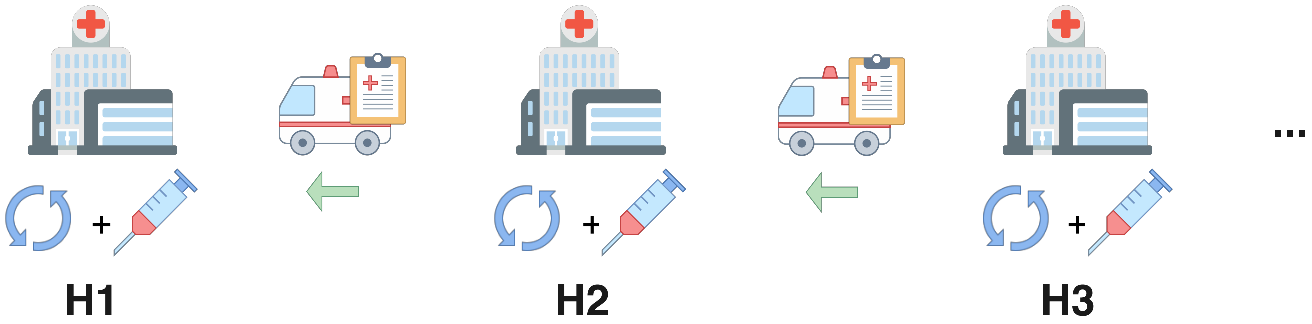
Exchange Workflow (t = 2)

- Share info with immediately higher rank
- Match local and new data *(from lower ranked hospital / incremental setting)*
- Perform surgeries for the patients on the queue, if any



Exchange Workflow (t = 3)

- Share info & send matched pairs
- Match local & new data *(from lower ranked hospital / incremental setting)*
- Perform surgeries for the patients on the queue, if any
- Repeat this until t = # of round set initially



Motivating Algorithms / Issues

1. **Top Trading Cycle** (TTC)
    a. Limited to maximum cycle length
    b. **ExchangePair** only ranks compatible pairs
    c. Order within rank list is irrelevant

1. **Tarjan's Algorithm** - Strongly Connected Components (SCCs)
    a. Must restrict SCCs to maximum cycle length
    b. Node belongs to a single SCC, i.e. a single cycle

Greedy Implementation

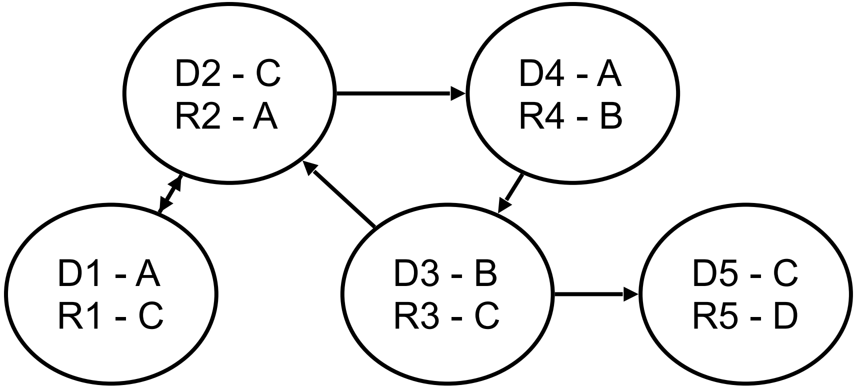
Input: **Hospital** Output: **list of cycles**

`Hospital 1:`
`    1: (A, C)`
`    2: (C, A)`
`    3: (B, C)`
`    4: (A, B)`
`    5: (C, D)`

`[ [2, 4, 3] ]`

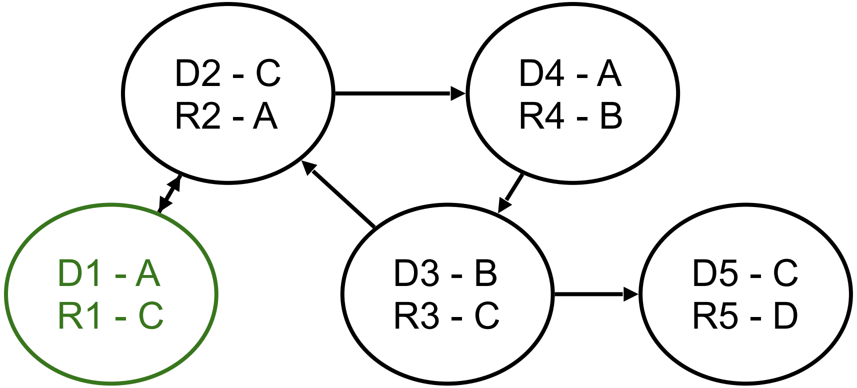
Greedy Implementation

```
greedyMatches(hospital):
  matches = []
  build directed graph
  build iterator list
  while (slots > 0 && iter.next())
    cycle = DFS(iter.next(), 0, slots,
1)
    matches.append(cycle)
    slots -= cycle.size
  return matches
```



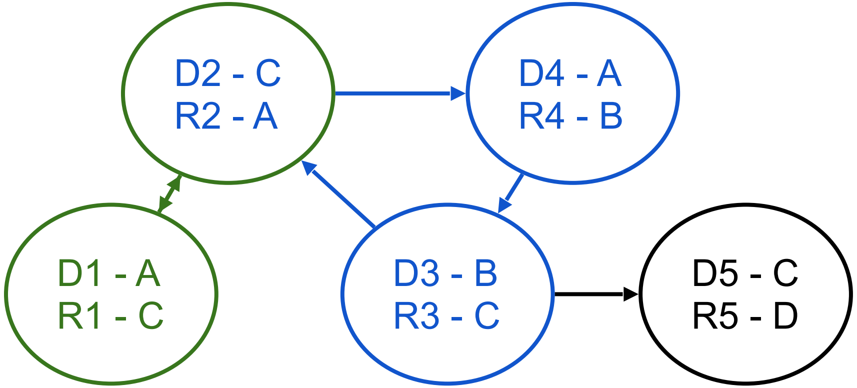
Greedy Implementation

```
DFS(node, parent, max, step):
  visiting.append(node)
  travelMap.put(node, (parent, step))
  for x in node.neighbors:
    if x in visiting:
      if step - travelMap.x.step >
max:
        return cycle
      else: continue
    elif neighbor in finished:
continue
    else: DFS(x, node, max, step)
  visiting.remove(node)
  finished.add(node)
```

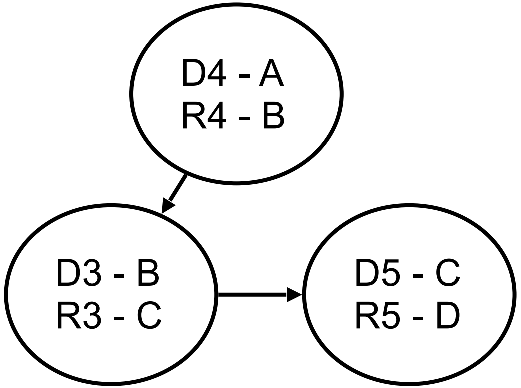


Greedy Implementation

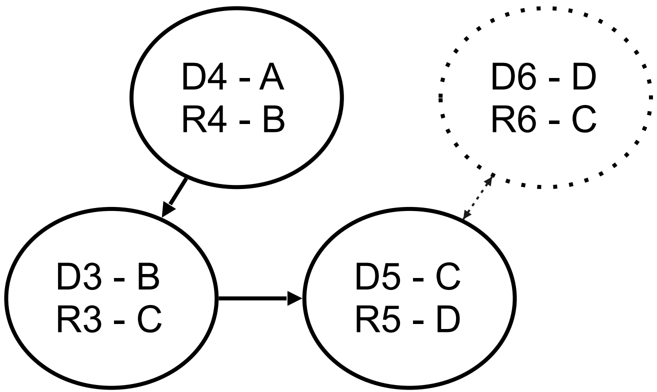
```
DFS(node, parent, max, step):
  visiting.append(node)
  travelMap.put(node, (parent, step))
  for x in node.neighbors:
    if x in visiting:
      if step - travelMap.x.step >
max:
        return cycle
      else: continue
    elif neighbor in finished:
continue
    else: DFS(x, node, max, step)
  visiting.remove(node)
  finished.add(node)
```



Greedy Implementation



Greedy Implementation



Matching via Integer Linear Programming

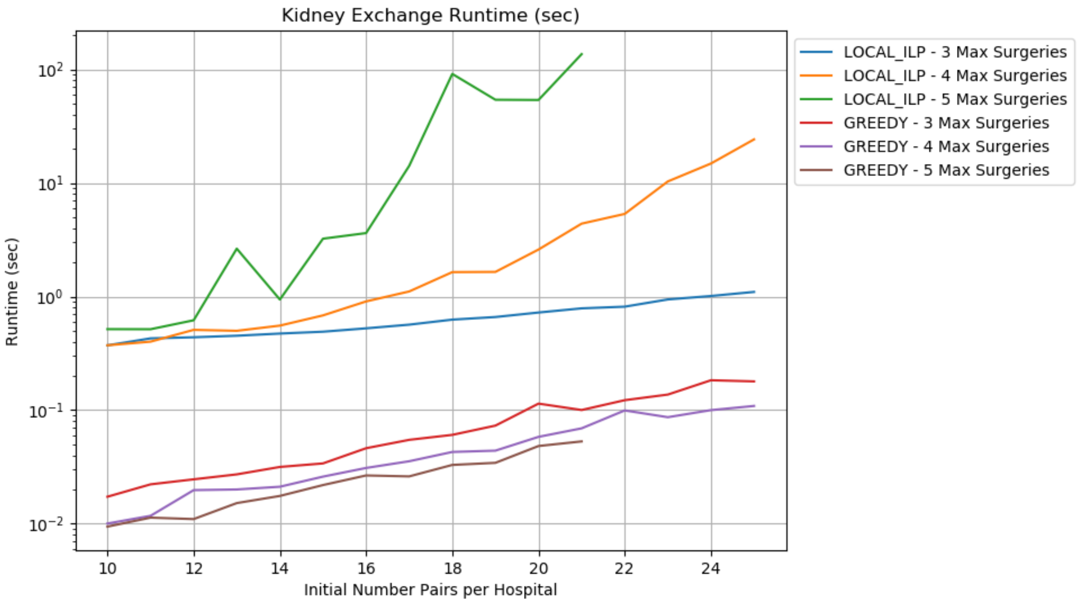
- Key idea: Find an maximally weighted cycle cover for directed graph of patient pairs
- Procedure:
    - Find all “short cycles” in directed graph
    - Assign each cycle a weight (sum of weight of edges in cycle)
        - Weight edges between local pairs more heavily
    - Formulate ILP with respect to key constraints:
        - Each patient pair can contribute to at most one cycle
        - Number of all patient pairs in selected cycles bounded by max number of surgeries

Matching via Integer Linear Programming

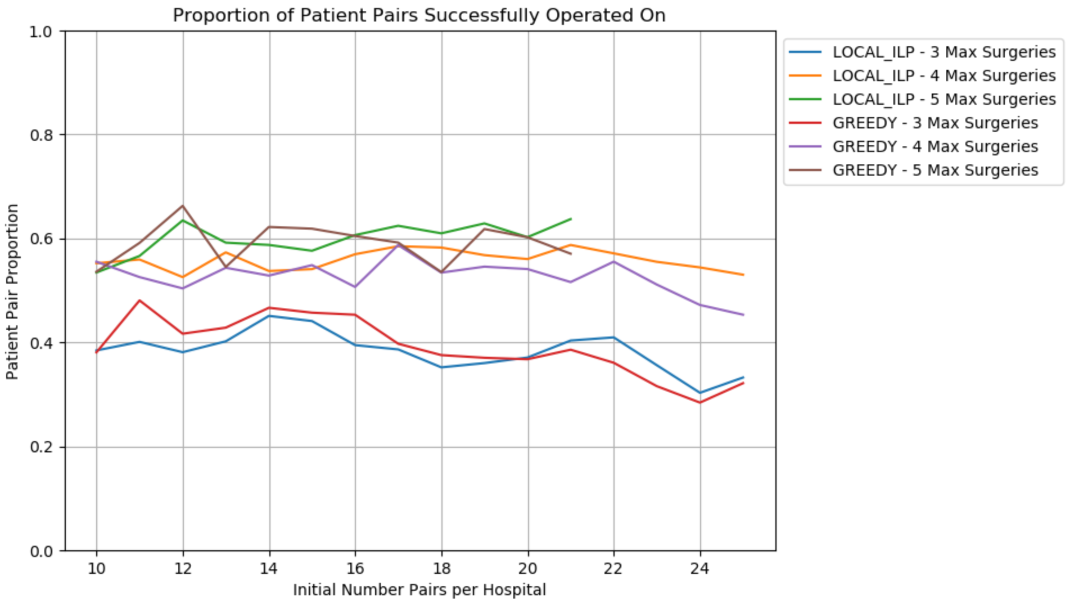
$$
\begin{aligned}
&maximize && \textstyle\sum_{i:c_i \in C} w_i x_i \\
&subject\ to && \textstyle\sum_{i:c_i \in C} l_i x_i \leq S \\
& && \textstyle\sum_{i:v_j \in c_i} x_i \leq 1 \quad \forall v_j \in V \\
& && x_i \in 0, 1
\end{aligned}
$$

DEMO

Performance



Performance



Conclusion

- Greedy algorithm scales better than ILP approach w/ respect to runtime
- ILP algorithm is more locally optimal for each hospital

Extensions

1. Enhance pair transfer (distributed) logic
    a. move to maximize slot usage, minimize patient disruption, etc.
2. Add **time to live** constraint for each patient
3. Add **cost** to move patient from Hospital X to Hospital Y
4. Include new types of **ExchangePairs**:
    a. A pair that is already matched (still needs to consume surgery slot)
    b. A pair with no donor
        i. model as pair with donor that is incompatible with all other real types
    c. A pair with no receiver
        i. model as pair with receiver that is compatible with all donors
5. Extend the compatibility function to model biological effects that make a matching more or less likely to be successful

Questions?