Master Thesis:
Offer Networks: Simulation and Dynamics

Zar Goertzel

August 18, 2017

What is an Offer Network?

- In the literature: "Barter Exchange Networks".

What is an Offer Network?

In the literature: "Barter Exchange Networks".

Basic idea:

- Users exchange services and goods.
- Allow ($n > 2$)-user exchanges (also gifts and chains).
- Use optimization algorithms to suggest exchanges (matches).

What is an Offer Network?

In the literature: "Barter Exchange Networks".

Basic idea:

- Users exchange services and goods.
- Allow ($n > 2$)-user exchanges (also gifts and chains).
- Use optimization algorithms to suggest exchanges (matches).

But why? Isn't money more efficient?

What's wrong with money?

Answer 1: sometimes it's unethical.

- *Only one* country allows organ sale.
- Organ sale is outlawed. It's a *repugnant* transaction.

What's wrong with money?

Answer 1: sometimes it's unethical.

Only one country allows organ sale.

Organ sale is outlawed. It's a *repugnant* transaction.

- Result: Kidney Exchange Network boom!
  - First US kidney exchange: 2000.
  - Now: 140+ transplant center kidney exchange network with biweekly matching.

What's wrong with money?

Answer 2: National Odd Shoe Exchange.

- Otherwise you would pay double.

What's wrong with money?

Answer 3: Ambiguous value:

- Home exchange:
  - Market value doesn't have to match up perfectly to *subjectively* satisfy homeowners.

What's wrong with money?

Answer 3: Ambiguous value:

- Home exchange:
  - Market value doesn't have to match up perfectly to *subjectively* satisfy homeowners.
- Data exchange:
  - What is value of data anyway?
  - May want framework for carefully restricted access . . ..

What's wrong with money?

Answer 4: Negligible value and scale:

- Book exchange.
- Homework help.
- Proofreading (of blogs or dating profiles – not just school essays).
- (Academic) book review exchange:
    - University professors would charge a high fee, unless the favor is returned.

++ I-YOU vs I-IT interactions

What's wrong with money?

Answer 5: Nothing.

- Works pretty well as market mechanism.
- Don't need "coincidence of wants".
  - Bookmooch uses credit sytem
- Recommender systems are still useful.
- *Exchange recommendation also useful?*
  - Elimination/reduction of middle-men.

State of Offer Network research

- Mostly kidney exchange research:

State of Offer Network research

Mostly kidney exchange research:

- 4 types of kidneys.
- Only interest in optimal solutions with small exchanges (≤ 3):
  - NP-Hard and APX-Complete.
  - Unless exchange-size unbounded with only edge-weight constraints.
- Usually Erdos-Renyi graph models

State of Offer Network research

Mostly kidney exchange research:

- Usually Erdos-Renyi graph models
- Domain-specific techniques used to improve Integer Linear Programming speed.

State of Offer Network research

Mostly kidney exchange research:

- Recently: PTIME approximation algorithm achieves $90 - 98\%$.
- Recently: Model incorportating rejection probability leads to $15\%+$ increase in transplants.
- Recently: Model keeping some kidneys for future use performs better (less *myopic*).
    - Latent potential for matches in graph is limited. Preserving useful parts helps.

State of Offer Network research

- Decentralized Offer Network: can selfishly exchanging agents pull off MacDonald's "paper clip for house" exchange chain?

State of Offer Network research

- Decentralized Offer Network: can selfishly exchanging agents pull off MacDonald's "paper clip for house" exchange chain? – Yes!

State of Offer Network research

- Decentralized Offer Network: can selfishly exchanging agents pull off MacDonald's "paper clip for house" exchange chain? – Yes!
- Greedy matching is best in Erdos-Renyi model.
  - Gift chains are better.

State of Offer Network research

- Abbassi test greedy cycle cover and two approximation algorithms on scale-free graph!
- Follow up with a comparison to credit-based system (works much better).
- Rappaz develop recommender system for barter exchange websites.

Question for this thesis

1. Extend Abbassi's experiments, and *include match acceptance probability*.

Question for this thesis

1. Extend Abbassi's experiments, and *include match acceptance probability*.
2. Is greedy matching frequency also best in scale-free graphs?
3. Is there a steady-state size of the Offer Network graph?

Question for this thesis

1. Extend Abbassi's experiments, and *include match acceptance probability*.
2. Is greedy matching frequency also best in scale-free graphs?
3. Is there a steady-state size of the Offer Network graph?
   - How long do users wait?
   - How many users are matched?
4. Does matching marginalize unpopular tasks?

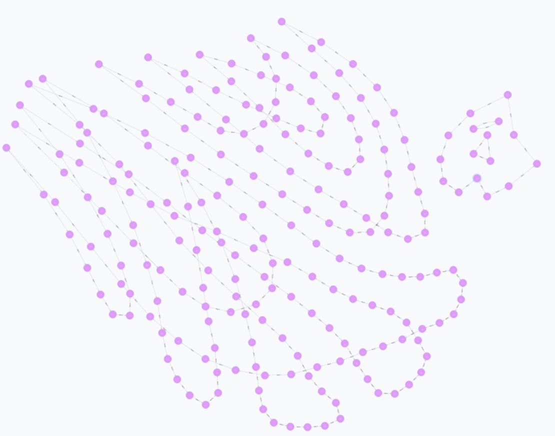
Question for this thesis

1. Extend Abbassi's experiments, and *include match acceptance probability*.
2. Is greedy matching frequency also best in scale-free graphs?
3. Is there a steady-state size of the Offer Network graph?
   - How long do users wait?
   - How many users are matched?
4. Does matching marginalize unpopular tasks?
5. Alternative to rejecting whole match if one user rejects?

Question for this thesis

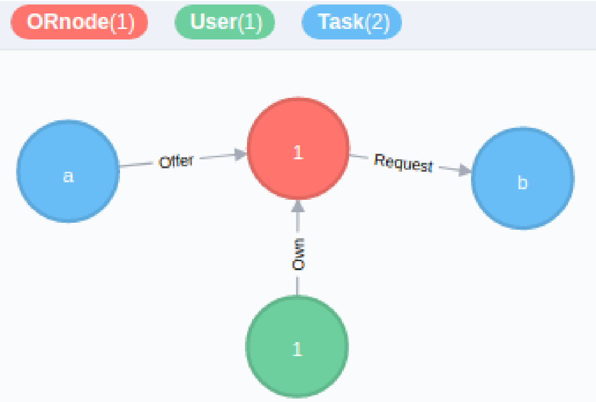
- Alternative to rejecting whole match if some users reject?

Very big exchange in "optimal matching" without size constraints:



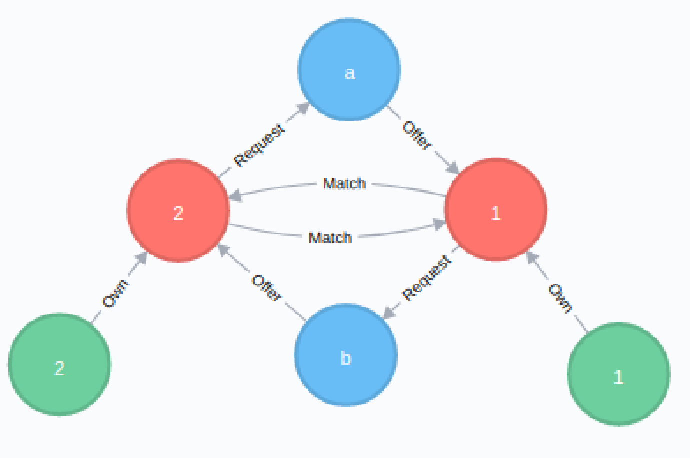
ORpair: (offer, request) pair

User 1 offers task a in exchange for requested task b:

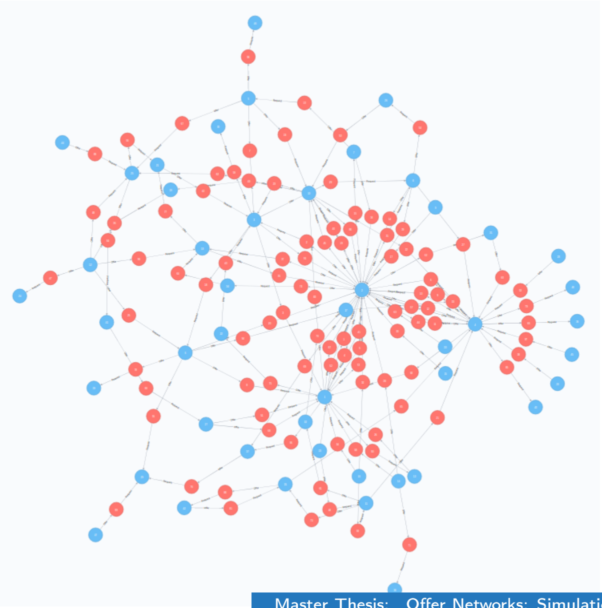


ORpair: (offer, request) pair

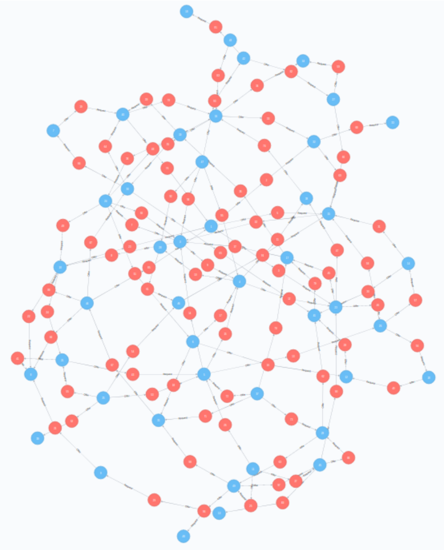
A match between User 1 and User 2's ORpairs:



Scale-Free Graph (RB1) with 100 ORpairs



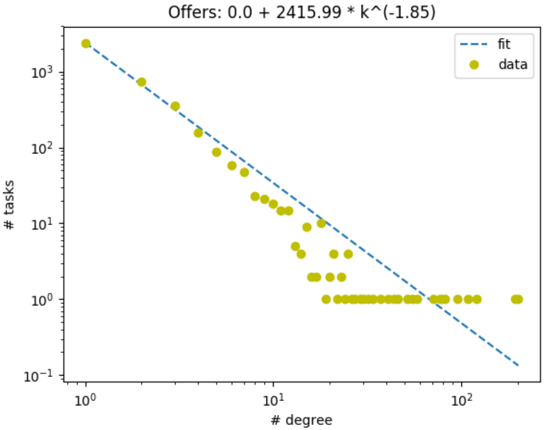
Erdos-Renyi ($p = 0.04$) with 100 ORpairs



Experimental Set-up: Data

- Find distribution of offers and requests in Ratebeer and Bookmooch datasets provided.
- Generate scale-free graphs with similar popularity distributions.
- I use 3 such graphs.

Scale-Free Graph (RB1) with 10,000 ORpairs



Experimental Set-up

- Time measured by new ORpairs
- Every n_{match} ORpairs, run a match algorithm
  - *Assume* uniform acceptance probability p.
- Add $N_{initial}$ ORpairs and run matching algorithm to initialize.
- Run test until N_{end} ORpairs added.

Experimental Set-up

- Time measured by new ORpairs
- Every n_{match} ORpairs, run a match algorithm
  - *Assume* uniform acceptance probability p.
- Add $N_{initial}$ ORpairs and run matching algorithm to initialize.
- Run test until N_{end} ORpairs added.

- Use same series of ORpair updates for each test run in an experiment.

Matching Algorithms

- Maximum 2-way matching: $\mathcal{O}(|\text{2-cycle}|^3)$. Performs horribly.
- Maximum Edge-Weight Matching (MAX). $\mathcal{O}(|\text{ORpair}|^3)$. Optimal if users always accept matches. I use Munkres algorithm implemented in Cython.
- Greedy Shortest Cycle (GSC). $\mathcal{O}((|ORpairs| + |tasks| \ln |tasks|)|ORpairs|)$. Good for $p < 1$?
- Dynamic Shortest Cycle (DYN). GSC only on new ORpairs.

Matching Algorithms

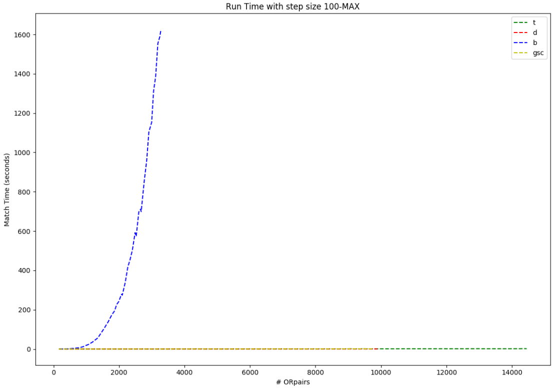
- Maximum 2-way matching: $\mathcal{O}(|\text{2-cycle}|^3)$. Performs horribly.
- Maximum Edge-Weight Matching (MAX). $\mathcal{O}(|\text{ORpair}|^3)$. Optimal if users always accept matches. Uses $|ORpair|^2$ matrix.
- Greedy Shortest Cycle (GSC). $\mathcal{O}((|ORpairs| + |tasks| \ln |tasks|)|ORpairs|)$. Good for $p < 1$?
  - Abbassi: use GSC many times.
  - Jia: use GSC to seed local search.
  - Jia: use product of degree (PoD) order. Bias toward unpopular tasks!

Neo4j Query Implementation of GSC

```
MATCH (o:ORnode)-[reqR:Request]->(req:Task),
p = shortestPath((req)-[link:Offer|:Request*]->(o))
WHERE NOT exists(reqR.matched) AND
ALL (r IN relationships(p) WHERE NOT exists(r.matchec
FOREACH (r IN link | SET r.matched = TRUE)
SET  reqR.matched = TRUE
WITH FILTER(ornode IN nodes(p) WHERE ornode:ORnode) A
UNWIND p as off
MATCH (off)<-[]-()<-[]-(req:ORnode)
WHERE req IN p AND off.offer = req.request
CREATE (off)-[:Match]->(req)
```

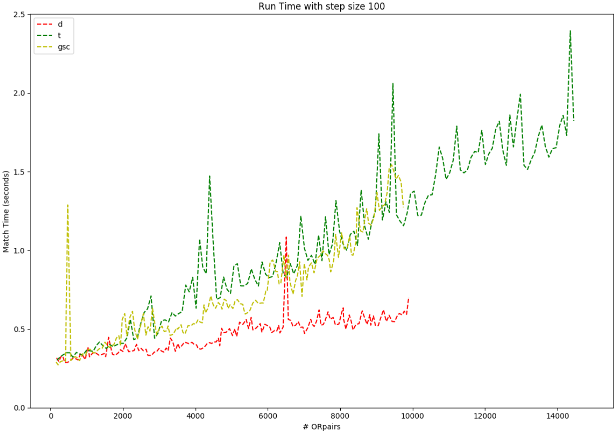
Run Time - MAX

Max run from 100 to 5000 ORpairs. Others 15,000. $n_{match} = 100$.



Run Time

Run from 100 to 15,000 ORpairs. $n_{match} = 20$.

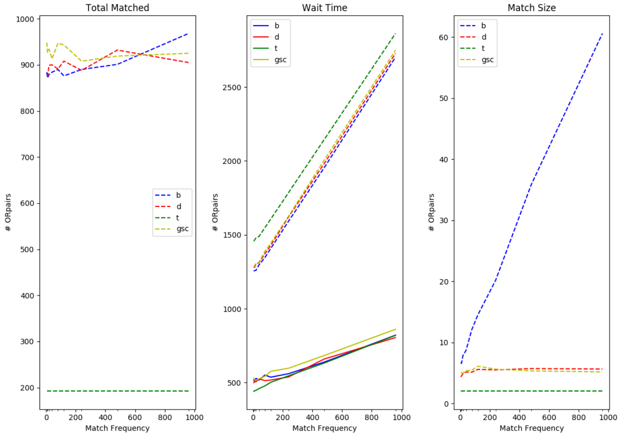


Run Time - GSC-PoD

Run from 100 to 15,000 ORpairs. $n_{match} = 100$.

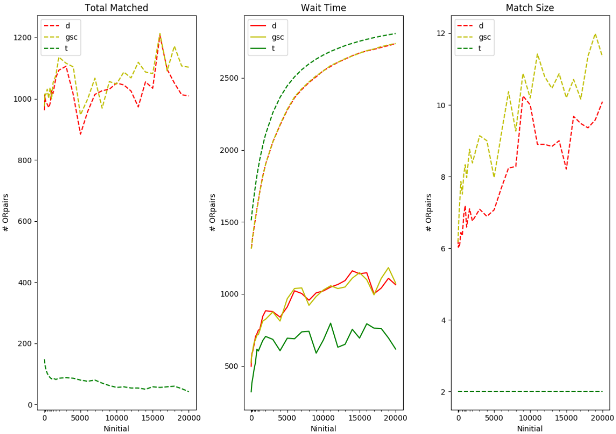
Match Frequency Experiment: $p = 1$

Run from 100 to 3003 ORpairs.



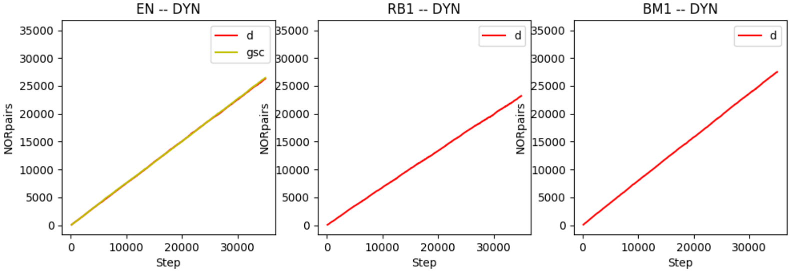
$N_{initial}$ Experiment: $p = 1$

Run for 3000 steps.



Long Run Experiment: $p = 1$

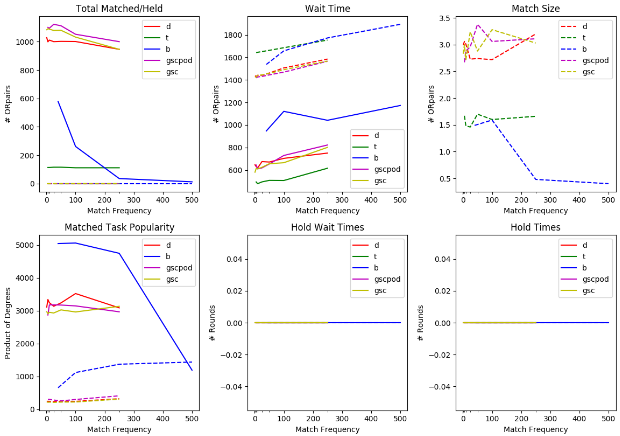
Run from 100 to 35,00 ORpairs. $n_{match} = 20$.



- Matching algorithms are all comparable for $p = 1$.
- Does MAX's performance deteriorate as expected?

Performance Test: $p = 0.9$

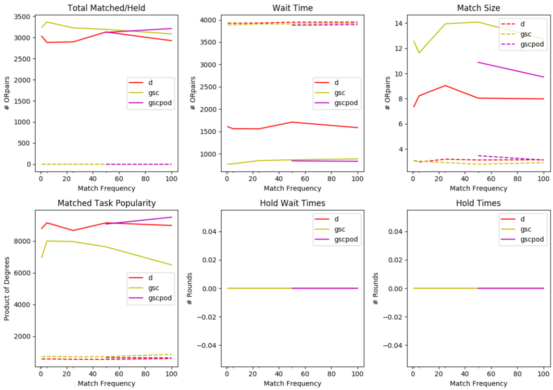
Run from 400 to 3400 (as MAX is slow)



- Matching algorithms are all comparable for $p = 1$.
- Does MAX's performance deteriorate as expected? – Yes.
- What about he other algorithms?
    - Wait time increases as p falls.
    - Accepted match size slowly falls.
    - Unpopular tasks marginalized. PoD less so.
    - Total matched ORpairs decrseases drastically.**

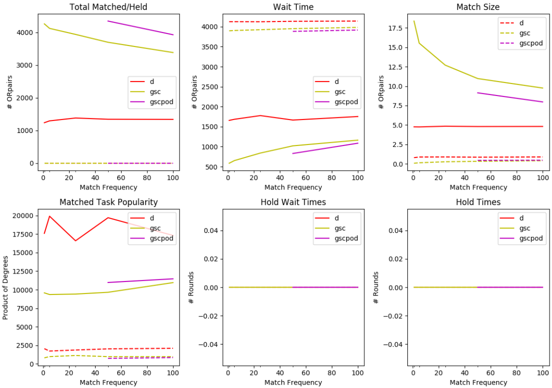
Performance Test: $p = 0.9$

Run from $10,000$ to $15,000$



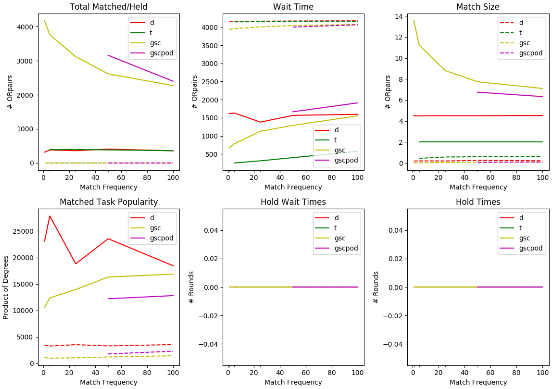
Performance Test: $p = 0.7$

Run from $10,000$ to $15,000$



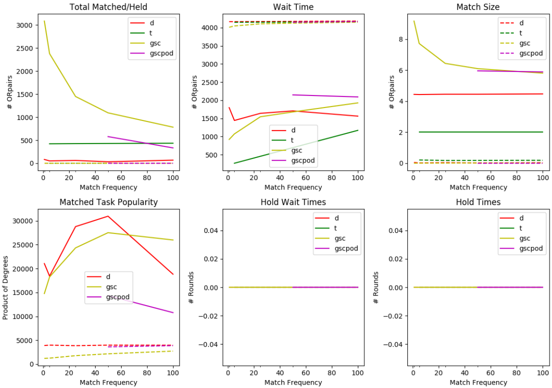
Performance Test: $p = 0.5$

Run from $10,000$ to $15,000$



Performance Test: $p = 0.3$

Run from $10,000$ to $15,000$

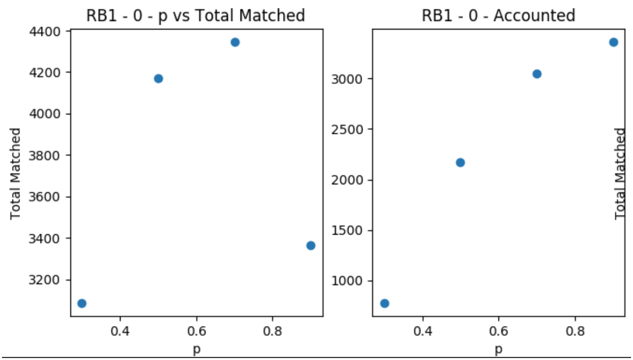


Two Sloppy/Mistaken Assumptions

1. During initialization, $N_{initial}$ nodes are added and then a matching algorithm is run once.
   - Problem: many more matched at $p = 0.9$ than $p = 0.3$. Thus there is more potential for matches left.
   - Measuring the number and accounting, the total matched nodes decrease with p.
2.

Performance Test: p vs total matched

Run from 10, 000 to 15, 000



Two Sloppy/Mistaken Assumptions

1. During initialization, $N_{initial}$ nodes are added and then a matching algorithm is run once.
2. Rematching is allowed.
   - Lazy reasoning behind assumption: there will be better things to do than rematch.
   - Big problem: small step size and shortest cycle matching.
   - However, low acceptance probability cases *don't catch up*.

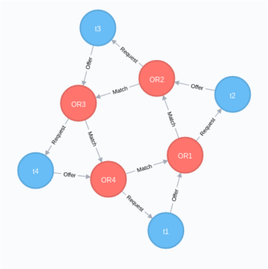
Can we maintain performance for low p?

Hanging ORpair method:

- Assume rough parity among matched tasks.
- Only need to worry about ORpairs bordering a rejected ORpair: combine into one node and re-add to graph.

Held ORpair Example

If OR1 rejects, make new node (offer t2, request t1).



HOR: pros and cons

Pros:

- Algorithmically, ORpair acceptance rate is similar to that for 3-way-matches.
- Held ORpairs are semi-random, but resemble Dickerson's potentials approach?
  I.e., less myopic
- Hint at generalization to asymmetric Offer Network design (or wait-list use). (*Future Work*)

HOR: pros and cons

Theoretical Cons:

- Parity doesn't always hold.
- Acceptance probability becomes p^2? (–Also forgot to implement.)
- Complicates additional features (e.g., task expiration, timing)

HOR: pros and cons

Practical Cons:

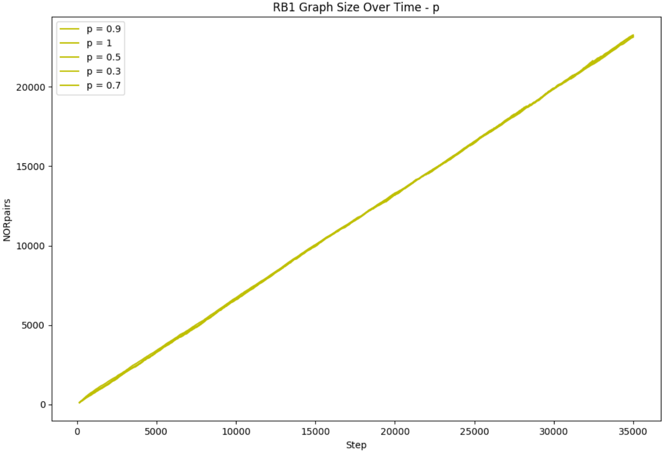
- User has to do more. Risks longer wait time.
- Incentive for user with held offer to stay?
  - Reputation system: negatively rate user for leaving: hard to find new matches.
  - Can be implemented like PoD or with MAX.
- Alteranitvely: add held request to waitlist queue; treat offer as gift.

HOR: Rematching

1. Held ORpair (t2, t1) can immediately be rematched with rejecting (t1,2)!
   - Murphy's law scenario: identical performance to $p = 1$ case with marginal wait time difference.
   - Curiously, potential ORpairs matched faster at lower p.
2. In the long run, latent graph potential for matches results in equal performance.
   - Properly disabling rematching may lead to low-p depleting graph potential.
   - However, this may merely mean postponed potential.

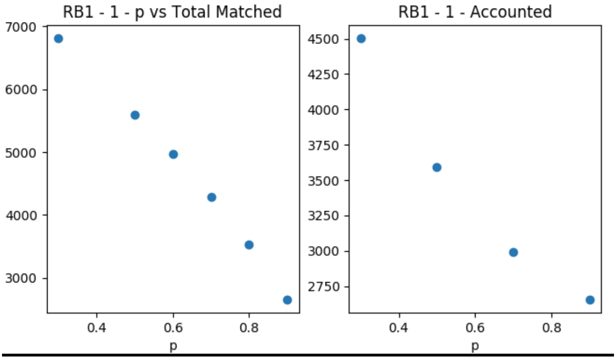
Long Run Graph Size

Run from 100 to $35,000$



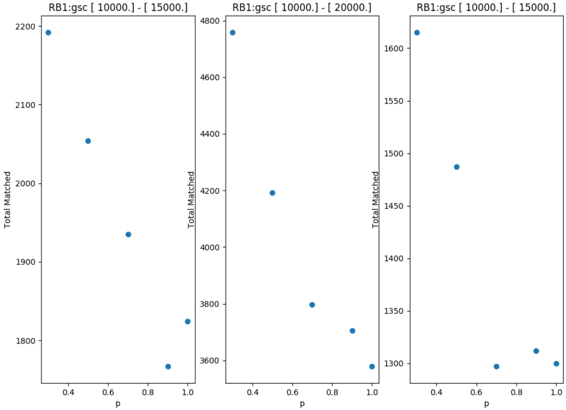
Performance Test: p vs total matched

Run from $10,000$ to $15,000$



Performance Test: p vs total matched

Ran a few runs with fixed, standardized ($p = 1$) matching for initialization.

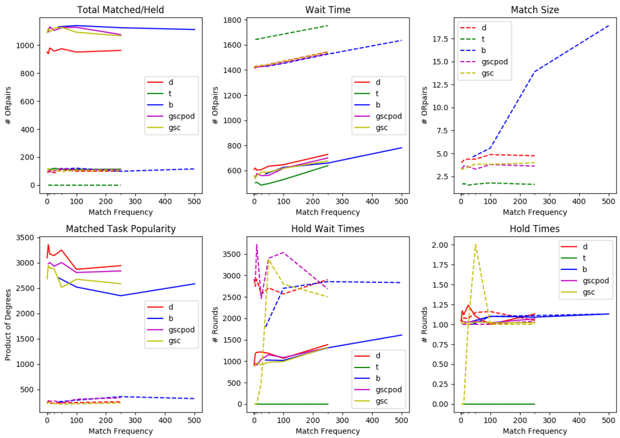


Performance Test: General Results

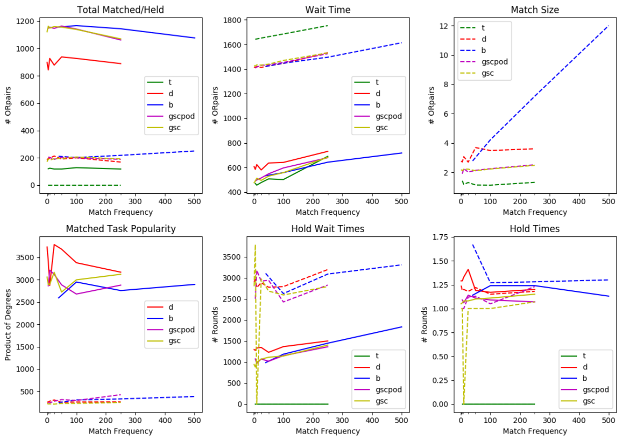
Tests with MAX: 400 to 3400

- With HOR, MAX performs best.
- Larger match size is okay.
- ORpairs held significantly longer
  - Which fits theoretical prediction. Perhaps abuses *rematching* less.
- Surprisingly: includes unpopular tasks best!

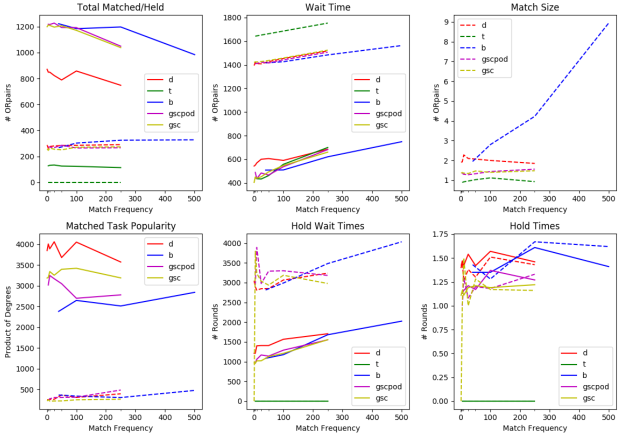
HOR Performance Test: $p = 0.9$



HOR Performance Test: $p = 0.8$

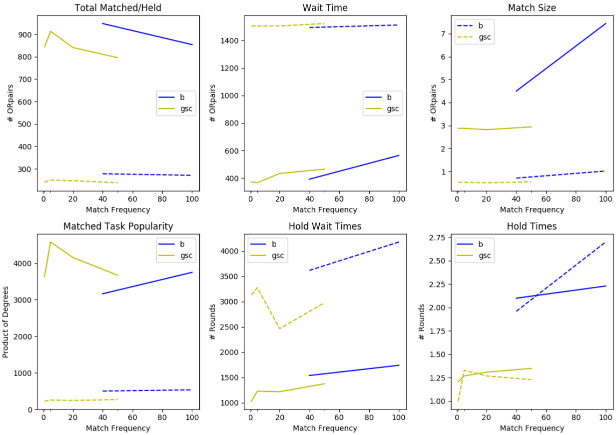


HOR Performance Test: $p = 0.7$



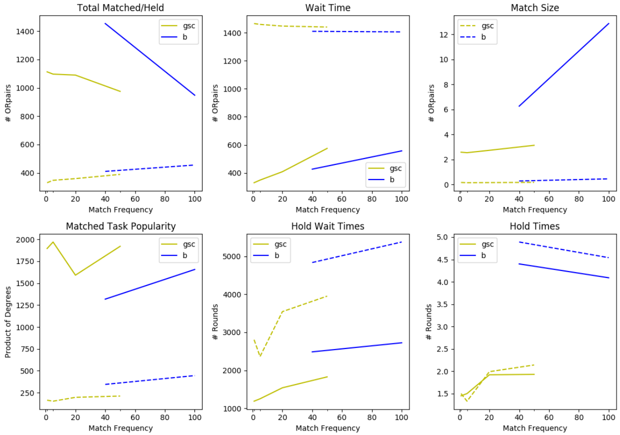
HOR Performance Test: $p = 0.5$

Note: using Bookmooch distribution graph



HOR Performance Test: $p = 0.3$

Note: using EN distribution graph.

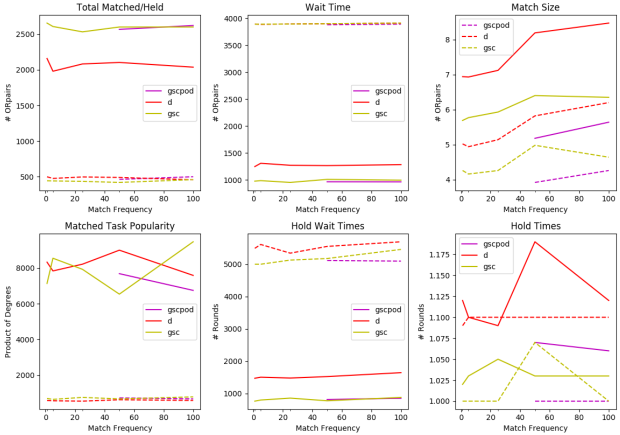


Performance Test: General Results

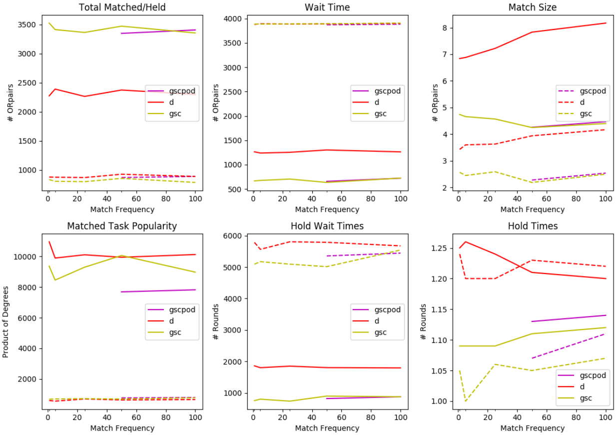
Tests from 10,000 to 15,000 (where wait time stabilizes)

- DYN does poorly below $p = 0.9$.
  - Because rejected/held ORpairs not re-added. (*future work*)
- GSC with Product of Degree order outperforms, marginally, in all metrics.
  - Q: is there a bigger difference without the ability to rematch ORpairs?
  - PoD much better for unpopular tasks at low p!
- Match size moves toward 2 and 3 as p decreases.
- Yet significantly more ORparis are matched.
  - Postponed graph potential + shortening cycles?
  - A user's ORpair can be used in multiple conflicting cycles.

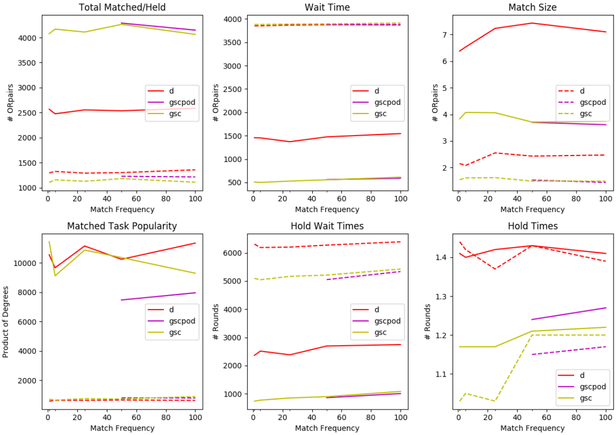
HOR Performance Test: $p = 0.9$



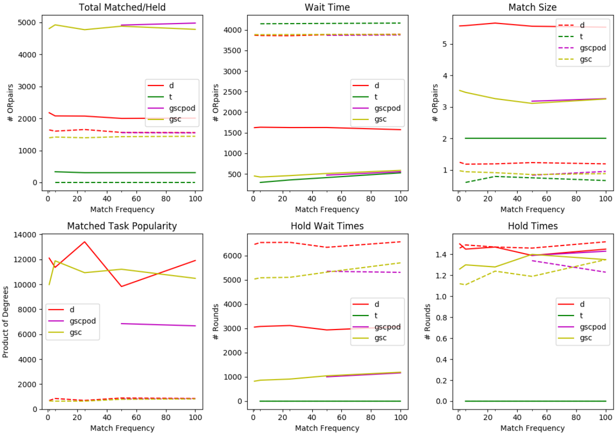
HOR Performance Test: $p = 0.8$



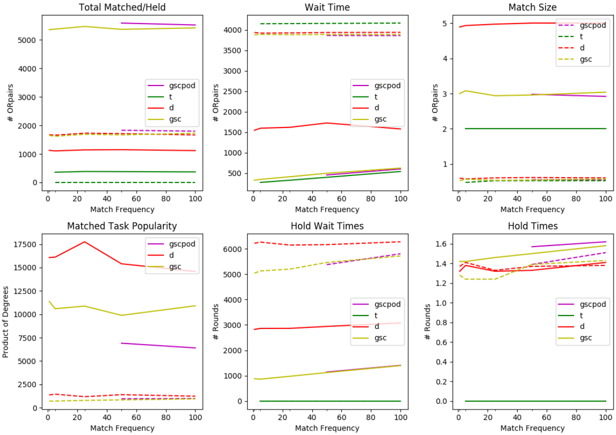
HOR Performance Test: $p = 0.7$



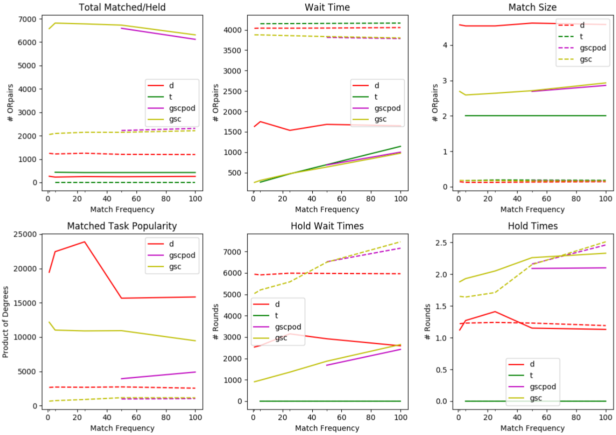
HOR Performance Test: $p = 0.6$



HOR Performance Test: $p = 0.5$



HOR Performance Test: $p = 0.3$



Discussion

So, can Offer Networks work?

1. In the long run, roughly 1/3 of ORpairs are matched.
   - Close to inherent graph limitation.
   - Task similarity based recommendation may help.

Discussion

So, can Offer Networks work?

1. In the long run, roughly $1/3$ of ORpairs are matched.
   - Close to inherent graph limitation.
   - Task similarity based recommendation may help.
2. Wait-time appears to stabilize around 1000 steps.
   - If this holds for much larger graphs, realistic wait time may be ok.

Discussion

So, can Offer Networks work?

1. In the long run, roughly 1/3 of ORpairs are matched.
    - Close to inherent graph limitation.
    - Task similarity based recommendation may help.
2. Wait-time appears to stabilize around 1000 steps.
    - If this holds for much larger graphs, realistic wait time may be ok.
3. Matching Algorithms (beyond swaps) seem to matter more for speed of depleting graph of matches.
    - Good for use in decentralized case.

Discussion

So, can Offer Networks work?

1. In the long run, roughly 1/3 of ORpairs are matched.
   - Close to inherent graph limitation.
   - Task similarity based recommendation may help.
2. Wait-time appears to stabilize around 1000 steps.
   - If this holds for much larger graphs, realistic wait time may be ok.
3. Matching Algorithms (beyond swaps) seem to matter more for speed of depleting graph of matches.
   - Good for use in decentralized case.
4. Hold rounds get high (for MAX) with low-p. Could be cumbersome.
5. Low-p cases work best suggesting very many matches in high frequency

Future Work

1. Disallow rematching, and re-run experiments.
2. Impose match suggestion limtis (irrespective of rematches).
3. Extend HOR to an asynchronous model (with gift chains allowed).
4. Experiment with task similarity and less uniform acceptance probability.
5. Add ORpair expiration.
6. Add reputation or user preferneces to the equation.
7. Conjunctions and disjunctions to allow more intricate offers and requests.
   (Will make the size of MAX's matrix bigger.)

Conclusions

1. Given the framework, offer network style exchange recommendations can supplement traditional money marketplaces, and recommendation systems.
2. But as in this thesis, probably not replace them. (I don't have statistics on how long buyers and sellers wait on Amazon though.)
3. The faster performance of low-p with HOR indicate that myopic matching isn't enough (as Dickerson found in the limited Kidney case.)
4. As with combined organ exchanges (kidney and lung), linking specialized exchange markets should prove beneficial.