CS364A: Algorithmic Game Theory
Lecture #10: Kidney Exchange and Stable Matching*

Tim Roughgarden†

October 23, 2013

1 Case Study: Kidney Exchange

Many people suffer from kidney failure and need a kidney transplant. Currently, the US waiting list for kidneys has about 100,000 people on it. An old idea, used also for other organs, is deceased donors — when someone dies and is a registered organ donor, their organs can be transplanted into others. One special feature of kidneys is that a healthy person has two of them and can survive just fine with only one of them. This creates the possibility of *living* organ donors, such as a family member of the patient in need.

Unfortunately, having a living kidney donor is not always enough — sometimes a patient-donor pair is *incompatible*, meaning that the donor's kidney is unlikely to function well in the patient. Blood and tissue types are the primary culprits for incompatibilities. For example, a patient with O blood type can only receive a kidney from a donor with the same blood type, and similarly an AB donor can only donate to an AB patient.

Suppose patient P1 is incompatible with its donor D1 because they have blood types A and B, respectively. Suppose P2 and D2 are in the opposite boat, with blood types B and A, respectively (Figure 1). Even though (P1,D1) may never have met (P2,D2), exchanging donors seems like a pretty good idea — P1 can get its kidney from D2 and P2 from D1. This is called a *kidney exchange*.

A few kidney exchanges were done, on an ad hoc basis, around the beginning of this century. These isolated successes make clear the need for a nationwide kidney exchange, where incompatible patient-donor pairs can register and be matched with others. How should such an exchange be designed? The goal of such an exchange is to thicken the kidney exchange market to enable as many matches as possible.

National kidney exchanges sprang up around the middle of last decade. We'll cover some of the early design ideas for these exchanges, as well as current challenges. These exchanges

†Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

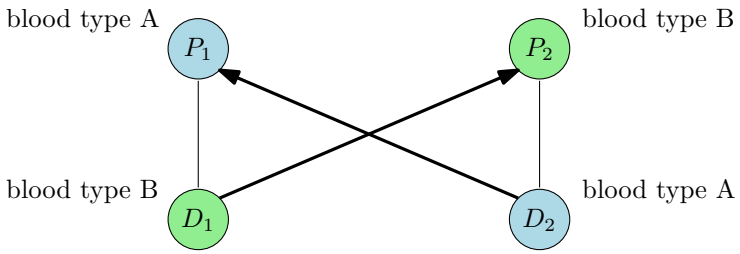

Figure 1: A kidney exchange

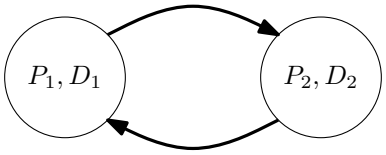

Figure 2: A good case for the Top Trading Cycle Algorithm.

and their underlying algorithms have been quite successful, enabling thousands of successful kidney transplants every year.

Currently, compensation for organ donation is illegal in US (and in every country except for Iran). Kidney exchange is legal. Thus kidney exchange is an ideal application for mechanism design without money, for whatever incentive issues are relevant (and there are a few). It's interesting to speculate about whether or not a monetary market for kidneys will exist in any Western countries in, say, 10 years. (Iran does not have a waiting list for kidneys.)

1.1 Idea #1: Use the Top Trading Cycle Algorithm

This section and the next cover two relatively early papers by Roth, Sönmez, and Ünver [5, 6] that are influential brainstorms about what an incentive-compatible national kidney exchange might look like. In the first paper [5], which covers research done before the authors talked extensively to doctors, advocated the Top Trading Cycle Algorithm (TTCA) from last lecture as a starting point. In its most basic form, the correspondence is between agent-initial house pairs in the housing problem and patient-donor pairs in the kidney exchange problem — the initial house of an agent corresponds to its incompatible living donor. The housing problem assumes that each agent has a total ordering over houses; in kidney exchange, it is natural to order donors according to the estimated probability that their kidney could be successfully transplanted into the patient (based on blood type, tissue type, etc.).

The goal of applying the TTCA is to find cycles like that in Figure 2, where with patient-donor pairs as in Figure 1, each patient points to the other's donor as its favorite. Reallocating donors according to this cycle is the favorable kidney exchange identified earlier. More generally, donors are reallocated to patients so that everyone is collectively as well off as possible (in the senses discussed last lecture).

The actual kidney exchange problem is more complicated in several ways. We first discuss one extension that can be accommodated by the TTCA approach, and then two

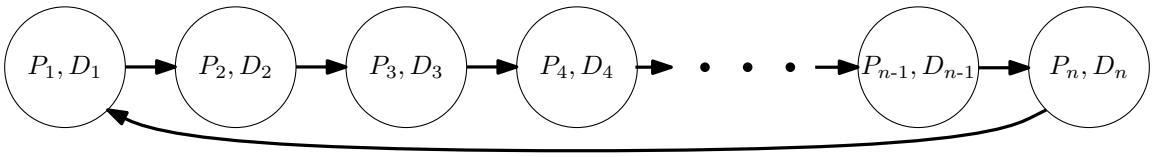

Figure 3: A bad case for the Top Trading Cycle Algorithm.

challenges that motivate reformulating the problem. The extension of the TTCA in [5] handles, in addition to incompatible patient-donor pairs, patients without living donors (an agent without an initial house) and deceased donors (houses without an initial owner). The new algorithm allocates along cycles as in the TTCA, and also along suitable chains (i.e., paths). Provided chains are chosen in the right way, the algorithm remains DSIC — so no patient (or patient's doctor) can misreport information to increase the estimated probability of a successful transplant.[1]

The next issue with the TTCA algorithm is a dealbreaker in the context of kidney exchange: the cycles along which reallocations are made can be arbitrarily long. For example, in the first iteration of TTCA, it is possible that the arcs corresponding to preferred donors form a Hamiltonian cycle on patient-donor pairs (Figure 3).

Why are long cycles a problem? Consider first a cycle of length two (Figure 2). Note this already corresponds to four surgeries — two to extract donors' kidneys, and two to implant them in the patients. Moreover, these four surgeries *must happen simultaneously*. Incentives are the reason: in the example in Figure 1, if the surgeries for P1 and D2 happen first, there is a risk that D1 will renege on its offer to donate its kidney to P2. One problem is that P1 unfairly got a kidney "for free;" the much more serious problem is that P2 is as sick as before and, since its donor D2 has donated its kidney, P2 can no longer participate in a kidney exchange. Because of this risk, no one wants to experiment with non-simultaneous surgeries in kidney exchange.[2] The constraint of simultaneous surgeries — with each surgery needing its own operating room, surgical team, etc. — motivates keeping reallocation cycles as small as possible.

Our final critique of the TTCA approach is that modeling preferences as a total ordering over the set of living donors is overkill: empirically, patients don't really care which kidney they get as long as it is compatible with them. Binary preferences over donors are therefore more appropriate.

[1]This extension was largely worked out earlier in a different context: allocating dorm rooms to students when there is a mixture of incumbents (students who already have a room but might want to upgrade), empty rooms (e.g., from students who graduated), and students without a current room (e.g., new students) [1].

[2]Sequential surgeries are now being used in slightly different contexts. For example, there is a handful of altruistic living donors, who are interested in donating a kidney even though they don't personally know anyone who needs one. An altruistic living donor can be the start of a chain of reallocations. Chains as long as 30 have been implemented [7], and at this scale surgeries have to be sequential. While the first problem of sequential surgeries (getting a kidney "for free") persists in this setting, the much more serious second problem (losing your own living donor) cannot occur in chains initiated by an altruistic living donor.

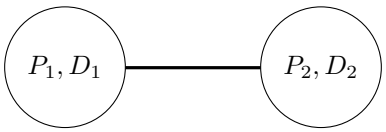

Figure 4: Applying a matching algorithm.

1.2 Idea #2: Use a Matching Algorithm

The second approach [6], motivated by the twin goals of binary preferences and short reallocation cycles, is to use *matchings*. Recall that a matching of an undirected graph (V, E) is a subset of edges $F \subseteq E$ such that no two edges of F share an endpoint.

The relevant graph for kidney exchange has a vertex set V corresponding to incompatible patient-donor pairs (one vertex per pair), and an undirected edge between vertices $(P1, D1)$ and $(P2, D2)$ if and only if P1 and D2 are compatible and P2 and D1 are compatible. Thus, the example in Figure 1 corresponds to the simple undirected graph in Figure 4. We define the optimal solutions to be the matchings of this graph that have maximum cardinality — that is, we want to arrange as many compatible kidney transplants as possible. By restricting the feasible set to matchings of this graph, we are restricting to pairwise kidney exchanges, and hence "only" 4 simultaneous surgeries, like in Figure 1.[3]

Our model for agents is that each vertex i has a true set E_i of incident edges, and can report any subset $F_i \subseteq E_i$ to a mechanism. Proposed kidney exchanges can be refused by a patient for any reason, so one way to implement a misreport is to refuse exchanges in $E_i \setminus F_i$. All that a patient cares about is being matched to a compatible donor. Our mechanism design goal is to compute an optimal solution (i.e., a maximum-cardinality matching) and to be DSIC, meaning that for every agent, reporting its full edge set is a dominant strategy.

Given our design goals, the mechanism must have the following form.

(1) Collect a report F_i from each agent i.

(2) Form the edge set$E = \{(i, j) : (i, j) \in F_i \cap F_j\}$. That is, include edge (i, j) if and only if both endpoints agree to the exchange.

(3) Return a maximum-cardinality matching of the graph $G = (V, E)$, where V is the (known) set of patient-donor pairs.

The mechanism is not yet fully specified, since there can be multiple choices for a maximum matching in step (3). Getting the tie-breaking right is important for achieving DSIC. There are two senses in which the maximum matching of a graph can fail to be unique. First, different sets of edges can be used to match the same set of vertices; see Figure 5. Since a

[3]These days, 3-way exchanges, corresponding to a directed cycle of 3 patient-donor pairs (with D2 compatible with P1, D3 with P2, and D1 with P3), are increasingly common. The reason is that 3-way exchanges are still logistically feasible and allowing them significantly increases the number of patients that can be saved. Empirically, exchanges involving 4 or more pairs don't really help match more patients, so they are not typically done.

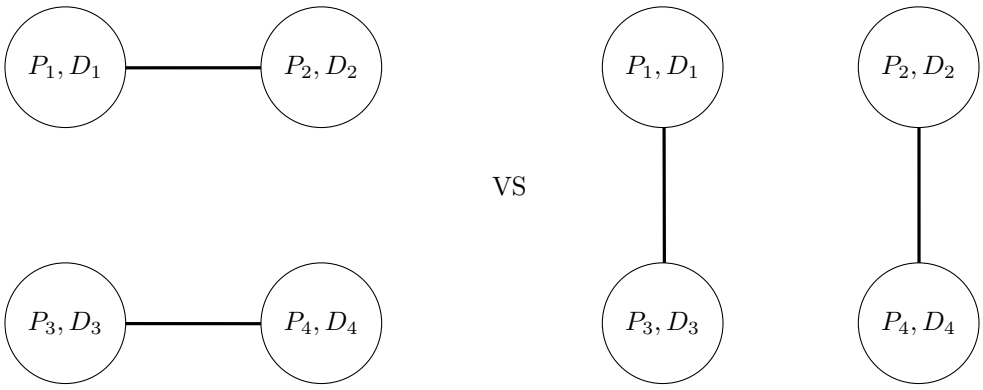

Figure 5: Different matchings can match the same set of vertices.

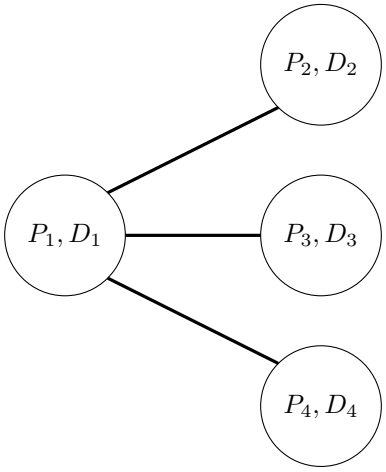

Figure 6: Different maximum matchings can match different subsets of vertices.

vertex does not care whom it is matched to, as long as it is matched, there is no reason to distinguish between different matchings that match the same set of vertices. More significantly, different maximum-cardinality matchings can match different subsets of vertices. For example, in the star (Figure 6), vertex 1 is in every maximum matching, but one and only one of the spokes will be chosen. How should we choose which one to return?

One solution is to prioritize the vertices before the mechanism begins. Without loss of generality, assume that the vertices $1, 2, \ldots, n$ are ordered from highest to lowest priority.[4] Then, we implement step (3) as follows:

(3a) Let M_0 denote the set of maximum matchings of G.

(3b) For $i = 1, 2, \ldots, n$:

[4]One appeal of this approach is that most hospitals already rely on similar priority schemes to manage their patients. The priority of a patient on a waiting list is determined by numerous factors, such as the length of time it has been waiting on the list, the difficulty of finding a compatible kidney, etc.

(3b.i) Let Z_i denote the matchings in M_{i-1} that match vertex i.

(3b.ii) If $Z_i \neq \emptyset$, set $M_i = Z_i$.

(3b.iii) Otherwise, set $M_i = M_{i-1}$.

(3c) Return an arbitrary matching of M_n.

That is, in each iteration i, we ask if there is a maximum matching that respects previous commitments and also matches vertex i. If so, then we additionally commit to matching i in the final matching. If previous commitments preclude matching i in a maximum-cardinality matching, then we skip i and move on to the next vertex. By induction on i, M_i is a non-empty subset of the maximum matchings of G. Every matching of M_n matches the same set of vertices — the vertices i for which Z_i was non-empty — so the choice of matching in step (3c) is irrelevant.

Theorem 1.1 *For every collection $\{E_i\}_{i=1}^n$ of edge sets and every ordering of the vertices, the priority matching mechanism above is DSIC: no agent can go from unmatched to matched by reporting a strict subset F_i of E_i rather than E_i.*

We leave the proof to you; see the exercises.[5]

1.3 Cutting-Edge Challenges

Current research is focused on incentive problems at the *hospital* level, rather than at the level of individual patient-donor pairs. This change is well motivated because many patient-donor pairs are reported to national kidney exchanges by hospitals, rather than by the pairs themselves. The objectives of a hospital (to match as many of its patients as possible) and of society (to match as many patients overall as possible) are not perfectly aligned. The key incentive issues are best explained through examples.

Example 1.2 (The Need for Full Reporting) Suppose there are two hospitals, each with three patients (Figure 7). Edges represent patient-donor pairs that are mutually compatible, as with the matching mechanism above. Each hospital has a pair of patient-donor pairs that it could match internally, without bothering to report them to a national exchange. We definitely don't want the hospitals to "greedily" execute these internal matches in this example — if H_1 matches 1 and 2 internally and only reports 3 to the exchange, and H_2 matches 5 and 6 internally and only reports 4 to the exchange, then the exchange can't match 3 and 4 so no further matches are gained. By contrast, if H_1 and H_2 both report their full sets of 3 patient-donor pairs to the national exchange, then 1, 2, and 3 can be matched with 4, 5, and 6, respectively, and so all patients get matched. In general, the goal is to incentive hospitals to report all of their patient-donor pairs, to save as many lives as possible.

[5]The keen reader is encouraged to read the deeper exploration in [6] of how the maximum matching selected varies with the chosen priority ordering over vertices. There is a very clean answer that goes through the Gallai-Edmonds decomposition, a beautiful result from matching theory that characterizes the structure of maximum-cardinality matchings in undirected graphs.

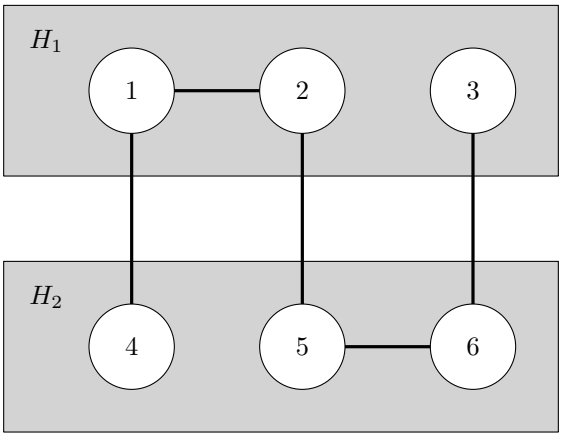

Figure 7: Example 1.2. Full reporting by hospitals leads to more matches than with only internal matches.

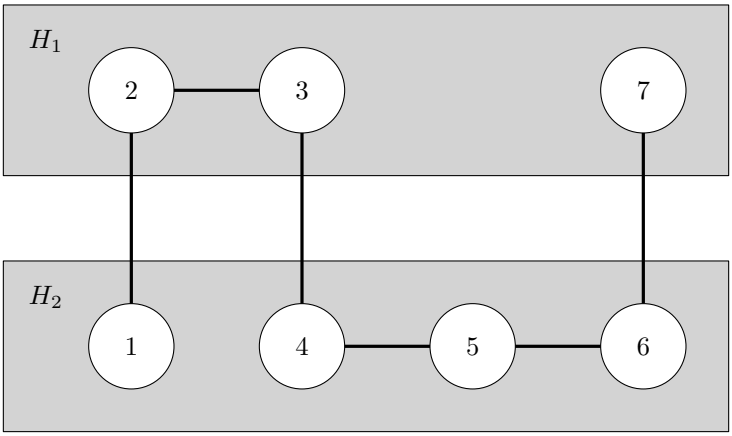

Figure 8: Example 1.3. Hospitals can have an incentive to hide patient-donor pairs.

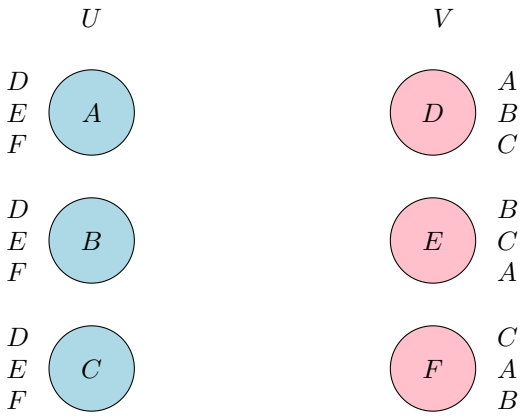

Figure 9: An instance of stable matching

Example 1.3 Consider again two hospitals, but now with 7 patients (Figure 8). Observe that, with an odd number of vertices, every matching leaves at least one patient unmatched. However, if H_1 hides patients 2 and 3 from the exchange (while H_2 reports truthfully), then H_1 guarantees that all of its patients are matched. The unique maximum matching in the report graph matches patient 6 with 7 (and 4 with 5), and H_1 can match 2 and 3 internally. On the other hand, if H_2 hides patients 5 and 6 while H_1 reports truthfully, then all of H_2's patients are matched. In this case, the unique maximum matching in the graph of report matches patient 1 with 2 and 4 with 3, while H_2 can match patients 5 and 6 internally.

The upshot is that there is irreconcilable tension between social and hospital incentives: there cannot be a DSIC mechanism that always computes a maximum-cardinality matching in the full graph.

In light of Example 1.3, the revised goal should be to compute an approximately maximum-cardinality matching so that, for each participating hospital, the number of its patients that get matched is approximately as large as in any matching, maximum-cardinality or otherwise. Understanding the extent to which this is possible, in both theory and practice, is an active research topic [2, 8].

2 Stable Matching

Stable matching is the canonical example of mechanism design without money. Killer applications include assigning medical school graduates to hospitals and students to elementary schools. The following model and algorithm are directly useful for these and other applications with amazingly few modifications.

We consider two sets U and V — the "men" and "women" — with n vertices each. Each vertex also has a total ordering over the vertices of the other set. For example, in Figure 9, the men all agree on the ranking of the women, while the women have very different opinions about the men.

Definition 2.1 A *stable matching* is a perfect bipartite matching — i.e., each vertex is matched to precisely one vertex from the other set — so that there is no "blocking pair," meaning:

(*) if $u \in U, v \in V$ are *not* matched, then either u prefers its mate v' in the matching to v, or v prefers its mate u' in the matching to u.

A perfect matching that failed condition (*) would spell trouble, since the blocking pair u and v would be tempted to run off with each other.[6]

We next discuss the famous proposal algorithm for computing a stable matching. Gale and Shapley [3] formalized the stable matching problem and gave this algorithm. Incredibly, it was later discovered that essentially the same algorithm had been used, since the 1950s, to assign medical residents to hospitals (as it is today) [4]!

The Proposal Algorithm:

- While there is an unattached man $u \in U$:
  - u proposes to its favorite woman who has not rejected him yet.
  - Each woman entertains only her best offer (from her perspective) thus far.

Example 2.2 Consider the instance in Figure 9. Suppose in the first iteration we choose the man C, who promptly proposes to his first choice, D. Woman D accepts because she currently has no other offers. If we pick man B in the next iteration, he also proposes to the woman D. Since woman D prefers B to C, she rejects C in favor of B. If we pick man A next, the result is similar: D rejects B in favor of A. A possible trajectory for the rest of the algorithm is: man C now proposes to his second choice, E; man B then also proposed to E, and E then rejects C in favor of B; and finally, C proposes to his last choice F, who accepts.

Several comments about the algorithm and its properties. First, it is underdetermined, leaving open how the unattached man is chosen. Second, note that each man systematically goes through his preference list, from top to bottom. Third, the men to which a woman is engaged only improve over the course of the algorithm. Fourth, the algorithm maintains the invariant that each man is matched to at most one woman and each woman is matched to at most one man.

Stable matchings and the proposal algorithm have an astonishing number of elegant properties. We begin with the most basic ones.

Theorem 2.3 (Computation of a Stable Matching) *The Proposal Algorithm terminates in at most n^2 iterations with a stable matching.*

[6] There is a clear similarity to the idea of a core allocation, introduced last lecture in the context of the housing allocation problem. If a matching is not stable, then it is not in the core — the pair u, v for which condition (*) is violated would be better off seceding from the mechanism. It's not hard to prove that the core allocations — matchings such that no subset of vertices could secede and do better — are precisely the stable matchings.

Figure 10: Example with multiple stable matchings

Corollary 2.4 (Existence of a Stable Matching) *For every collection of preference lists for the men and women, there exists at least one stable matching.*

We emphasize that Corollary 2.4 is *not* obvious a priori. Indeed, there are some simple variants of the stable matching problem for which a solution is not guaranteed.

Proof of Theorem 2.3: The bound on the number of iterations is easy to prove. Since each man works his way down his preference list, never proposing to the same woman twice, he makes at most n proposals. This makes for at most n^2 proposals (and hence iterations) overall.

Next, we claim that the proposal algorithm always terminates with a perfect matching. For if not, some man must have been rejected by all n women. A man is only rejected by a woman in favor of being matched to a better man, and once a woman is matched to a man, she remains matched to some man for the remainder of the algorithm. Thus, all n women must be matched at the end of the algorithm. But then all n men are also matched at the end of the algorithm, a contradiction.

Finally, we claim that the perfect matching computed is stable. To see why, consider a man $u \in U$ and a woman $v \in V$ who are not matched to each other. This can occur for two different reasons. In the first case, u never proposed to v. Since u worked its way down its list starting from the top, this means that u ended up matched to someone he prefers to v. If u did propose to v at some point in the algorithm, it must be that v rejected u in favor of a man she preferred (either at the time of u's proposal, or later). Since the sequence of men to which v is matched only improves over the course of the algorithm, she ended up matched to someone she prefers to u. ■

We noted earlier that the proposal algorithm does not specify how to choose among the unattached men in an iteration. Do all possible choices lead to the same stable matching? In Figure 9 there is only one stable matching, so in that example the answer is yes. In general, however, there can be more than one stable matching. In Figure 10, the men and the women both disagree on the ranking of the others. In the matching computed by the proposal algorithm, both men get their first choice, with A and B matched to C and D, respectively. Giving the women their first choices yields another stable matching.

We prove a stronger statement to resolve whether or not the output of the proposal algorithm is unique. For a man $u \in U$, let $h(u)$ be the highest-ranked woman (in u's

preference list) that u is matched to in *any* stable matching. Then:

Theorem 2.5 (Male-Optimal Stable Matching) *The stable matching computed by the proposal algorithm matches every man $u \in U$ to $h(u)$.*

Theorem 2.5 immediately implies that the output of the proposal algorithm is independent of which unattached man is chosen in each iteration. It also implies that there exists a "male-optimal" stable matching — a stable matching in which every man simultaneously attains his "best-case scenario" — no trade-offs between different men are necessary. A priori, there is no reason to expect the $h(u)$'s to be distinct and therefore induce a perfect matching.

Proof of Theorem 2.5: Let $R = \{(u, v)\}$ denote the pairs such that v rejects u at some point in a fixed execution of the proposal algorithm.

Since each man systematically works his way down his preference list, if u is matched to v at the conclusion of the algorithm, then $(u, v') \in R$ for every v' that u prefers to v. Thus, the following claim would imply the theorem: for every $(u, v) \in R$, *no* stable matching pairs up u and v.

We prove the claim by induction on the number of iterations of the proposal algorithm. Initially, $R = \emptyset$ and there is nothing to prove. For the inductive step, consider an iteration of the proposal algorithm in which v rejects u in favor of u' — thus one of u, u' proposed to v in this iteration.

Since u' systematically worked his way down his preference list, for every v' that u' prefers to v, (u', v') is already in the current set R of rejected proposals. By the inductive hypothesis, no stable matching pairs up u' with a woman he prefers to v — in every stable matching, u' is paired with v or someone less preferred. Since v' prefers u' to u, and u' prefers v to anyone else he might be matched to in a stable matching, there is no stable matching that pairs u with v (otherwise u', v would form a blocking pair). ■

It can also be shown that the proposal algorithm outputs the worst-possible stable matching from the perspective of the women — the algorithm matches each $v \in V$ to the lowest-ranked man $l(v) \in U$ that v is matched to in any stable matching.[7]

Suppose the preference lists of the vertices are private — is the proposal algorithm DSIC? That is, does a man or woman ever wind up with a better mate under misreported preferences than under truthfully reported preferences? As the male-optimality property of the computed stable matching might suggest, the proposal algorithm is DSIC for the men but not for the women. The former property is not trivial to prove (see Problem Set #3) while the latter can be verified in simple examples (see Exercise Set #5).

References

[1] A. Abdulkadiroğlu and T. Sönmez. House allocation with existing tenants. *Journal of Economic Theory*, 88:233–260, 1999.

[7] Of course, one can change the algorithm to let the women make the proposals and the men the rejections to reverse both of these properties.

[2] I. Ashlagi, F. A. Fischer, I. A. Kash, and A. D. Procaccia. Mix and match: A strategyproof mechanism for multi-hospital kidney exchange. In *Proceedings of the 11th ACM Conference on Electronic Commerce (EC)*, pages 305–314, 2010.

[3] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[4] A. E. Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, 92:991–1016, 1984.

[5] A. E. Roth, T. Sönmez, and M. U. Ünver. Kidney exchange. *Quarterly Journal of Economics*, 119:457–488, 2004.

[6] A. E. Roth, T. Sönmez, and M. U. Ünver. Pairwise kidney exchange. *Journal of Economic Theory*, 125:151–188, 2005.

[7] K. Sack. 60 lives, 30 kidneys, all linked. *New York Times*, February 18 2012.

[8] P. Toulis and D. C. Parkes. A random graph model of kidney exchanges: Efficiency, individual-rationality and incentives. In *Proceedings of the 12th ACM Conference on Electronic Commerce (EC)*, pages 323–332, 2011.