Stable Matchings

Michael St. Jules

November 16, 2013

1 Stable Marriages

The stable marriage problem is the following: given n men and n women, each man with a list ranking the women in order of preference and each woman with a preference list of the men, find a *stable matching* of the men and women, i.e. n man-woman pairs that is *stable*.

Definition 1.1. An **instance** of the stable marriage problem is a set of M of men and a set W of women, each of size $n \in \mathbb{N}$ and for each $m \in M$, an ordered list $L(m)$ that is a permutation of the women in W and for each $w \in W$, an ordered list $L(w)$ that is a permutation of the men in M.

We will use $m, m', m'', m_1, m_2, m_3, \ldots, m_n$ to denote men, and similarly for women with w.

The lists $L(m)$ and $L(w)$ are m's and w's **preference lists** (or simply **lists**), respectively.

We say m **prefers** w **to** w', or w is a **better partner** for m than w' or w' is a **worse partner** for m than w, if w appears before w' in $L(m)$. We do the same for w, m and m'.

The following is an instance:

$$\begin{array}{ll} m_1\ [w_1\ w_3\ w_2] & w_1\ [m_2\ m_3\ m_1] \\ m_2\ [w_3\ w_1\ w_2] & w_2\ [m_3\ m_1\ m_2] \\ m_3\ [w_3\ w_1\ w_2] & w_3\ [m_1\ m_2\ m_3] \end{array}$$

In practice, the instance would be represented as two $n \times n$ matrices, one with the men's lists filling out the rows and the other with the women's lists filling out the rows.

Definition 1.2. A **matching** for an instance of the stable marriage problem is a set $\mathcal{M} \subseteq M \times W$ such that each $m \in M$ and each $w \in W$ appear in exactly one ordered pair in the set $\mathcal{M}$.

Hence each man is paired with exactly one woman and each woman, with exactly one man. In fact, $\mathcal{M} : M \to W$ defines a bijection, with $\mathcal{M}(m)$ as the partner of $m \in M$ in the matching and $\mathcal{M}^{-1}(w)$ being that of $w \in W$.

Definition 1.3. A pair $(m, w) \in M \times W$ is a **blocking pair** for the matching $\mathcal{M}$ if m and w both prefer each other to their own partners in $\mathcal{M}$.

Definition 1.4. A matching $\mathcal{M}$ is **stable** or a **solution**, if there is no blocking pair in $M \times W$ for $\mathcal{M}$.

The motivation for such a definition is that given a blocking pair (m, w), m and w would *leave* their partners for each other.

Definition 1.5. $(m, w) \in M \times W$ is a **possible pairing** if $(m, w) \in \mathcal{M}$ for some stable matching $\mathcal{M}$. Furthermore, m is called a **possible partner** for w, and similarly w for m.

Definition 1.6 A stable matching $\mathcal{M}$ is **man-optimal** (**man-pessimal**) if each $m \in M$ is paired with the $w \in W$ he prefers most (least, respectively) of all possible partners. We let $best(m) = w$ ($worst(m) = w$, resp.), to define a function $best : M \to W$ ($worst : M \to W$, resp.).

Define **woman-optimal**, **woman-pessimal**, $best : W \to M$ and $worst : W \to M$ similarly.

We will prove that the following algorithm, originally from "College Admissions and Stability of Marriage" by Gale and Shapley, always computes a stable matching (so that one always exists), and in $O(n^2)$ time. We shall also see that the stable matching returned is man-optimal and woman-pessimal, so that it does not depend on the order in which men are chosen to propose. The following more general version of the algorithm is adapted from "Algorithm Design" by Kleinberg and Tardos, and its analysis is from Gale and Shapley's original paper and some proofs adapted from "An Efficient Algorithm for the "Stable Roommates" Problem" by Irving:

Stable marriage algorithm

While there remains a man $m \in M$ who has no proposal to another held, do
    $w :=$ the next women on m's list to whom m has not proposed
    m proposes to w
    if w holds no proposal, then
        w holds m's proposal
    else $m' :=$ the man from whom w holds a proposal
        if w prefers m' to m, then w rejects m
        else
            w rejects m'
            w holds m's proposal
return the set $\mathcal{M} := \{(m, w) \in M \times W |\ w$ holds a proposal from $m\}$

Let's consider running the algorithm on the following instance:

$$\begin{array}{ll} m_1\ [w_1\ w_3\ w_2] & w_1\ [m_2\ m_3\ m_1] \\ m_2\ [w_3\ w_1\ w_2] & w_2\ [m_3\ m_1\ m_2] \\ m_3\ [w_3\ w_1\ w_2] & w_3\ [m_1\ m_2\ m_3] \end{array}$$

m_1 proposes to w_1, who accepts: (m_1, w_1)
m_2 proposes to w_3, who accepts: $(m_1, w_1), (m_2, w_3)$
m_3 proposes to w_3, who rejects m_3: $(m_1, w_1), (m_2, w_3)$
m_3 proposes to w_1, who accepts m_3 and rejects m_1: $(m_3, w_1), (m_2, w_3)$
m_1 proposes to w_3, who rejects m_1: $(m_3, w_1), (m_2, w_3)$
m_1 proposes to w_2, who accepts: $(m_3, w_1), (m_2, w_3), (m_1, w_2)$
Then $\mathcal{M} := \{(m_3, w_1), (m_2, w_3), (m_1, w_2)\}$ is a stable matching.

Lemma 1.7. The stable marriage algorithm runs in $O(n^2)$-time.

Proof. We assume that preference checking by women and finding the next person on a preference list by men are $O(1)$, and so given the women's preference lists, we can scan them and form a ranking matrix for the women, where the entry ij corresponds to w_i's ranking of m_j, i.e. m_j's position on $L(w_i)$. Similarly, given a ranking matrix for men, we can construct the men's preference lists by copying the ranking matrix, pairing each entry with the corresponding woman ranked, and then sorting the women on each row with respect to rank. This preprocessing takes $O(n^2)$ time.

Observe that each man proposes at most once to each woman, so at most n^2 proposals are made, and each iteration of the while loop corresponds to a single proposal and takes time $O(1)$. Hence $O(n^2)$ time in all.

□

Note also that $O(n^2)$ is *linear* in the input size.

Lemma 1.8. After the moment $w \in W$ is first proposed to, w always holds a proposal.

Proof. Once w holds a proposal, she only rejects it for a better one, and there is no 'unproposing'.

□

In fact, each man does worse with each proposal made, while women are never worse off after a proposal than before.

Lemma 1.9. The set $\mathcal{M}$ returned at the end of the stable marriage algorithm is a matching.

Proof. Suppose, by way of contradiction, that $\mathcal{M}$ is not a matching. First, since women can hold only a single proposal, each $m \in M$ and $w \in W$ appears at most once in $\mathcal{M}$, so it follows that some $m \in M$ or some $w \in W$ is unpaired. These two conditions are equivalent, again since women can only hold a single proposal, and also because $|M| = |W| = n$. Hence, we have both an unpaired man $m \in M$ and an unpaired woman $w \in W$. Since m is unpaired, he had no proposal held at the end of the procedure and so must have proposed to w before termination. Then, by Lemma 1.8, w could not have been unpaired.

□

Lemma 1.10. If $w \in W$ rejects $m \in M$ in the stable marriage algorithm, then m and w cannot be partners in any stable matching.

Proof. Suppose m's rejection by w is the first to occur such that (m, w) is a pairing in a stable matching $\mathcal{M}$. w rejected m because she already held or later received a proposal from a better partner, say m'. Then, so that (m', w) is not a blocking pair for $\mathcal{M}$, contradicting stability, it must be that m' prefers his own partner, w', in $\mathcal{M}$ to w. However, since m' prefers w' to w, m' must have proposed to and been rejected by w' before w, contrary to m's rejection by w being the first such.

□

Corollary 1.10.1. If in the stable marriage algorithm, $m \in M$ proposed to $w \in W$, then, in any stable matching

(i) m cannot have a better partner than w;

(ii) w cannot have a worse partner than m.

Proof. m proposed to w since he was rejected by all the women he preferred to w and hence could not be paired with (by Lemma 1.10), so (i) follows.

Suppose, by way of contradiction, that $(m', w) \in M \times W$ is a pair in some stable matching $\mathcal{M}$, where w prefers m to m'. Then m is paired with some $w' \in W$ and by (i), m prefers w to w'. Hence (m, w) is a blocking pair for $\mathcal{M}$, contradicting stability.

□

Theorem 1.11. The stable marriage algorithm computes a man-optimal and woman-pessimal stable matching in $O(n^2)$-time.

Proof. The $O(n^2)$ upper bound follows from Lemma 1.7.

The set $\mathcal{M}$ returned is a matching by Lemma 1.9.

$\mathcal{M}$ is stable, since if $(m, w) \in M \times W$ were a blocking pair, then m prefers w to $\mathcal{M}(m)$, contradicting Corollary 1.10.1.(i).

That $\mathcal{M}$ is man-optimal and woman-pessimal also follows from Corollary 1.10.1.

□

Corollary 1.11.1. A stable matching $\mathcal{M}$ is man-optimal if and only if it is woman-pessimal.

Corollary 1.11.2. There always exists a stable matching, i.e. $\mathcal{M} := \{(m, best(m)) \in M \times W | m \in M\} = \{(worst(w), w) \in M \times W | w \in W\}$.

2 Hospitals/Residents (College Admissions)

The hospitals/residents problem is a generalization of the stable marriage problem in which residents and hospitals are considered instead of men and women, allowing multiple residents to be matched to a hospital up to its quota. The problem was presented as the college admissions problem and solved by Gale and Shapley in the same paper as the stable marriages problem, with essentially the same algorithm, the chief difference being that colleges (hospitals) can hold multiple applications, up to their quota. In this setting, we do not require that every applicant is assigned to a college or that every college meets its quota, and we also allow the possibility that a college may prefer not to accept a particular applicant even when its quota is not met or that an applicant may prefer not to apply to a particular college even if it would mean not being accepted anywhere.

In the applicant-optimal procedure, when a college receives a new application while its currently held applicants would meet its quota, it rejects the worst application from the new one and the ones it already held.

In the college-optimal procedure, colleges make offers to applicants which have not rejected them up until they have enough offers held to fill their quotas.

Definition 2.1. An **instance** of the college admission problem consists of a set A of n applicants, a set C of m colleges with a quota $q_c \geq 1$ for each $c \in C$ and the (possibly incomplete) preference lists for all $a \in A, c \in C$.

Definition 2.2. An **assignment** is a set $\mathcal{A} \subseteq A \times C$ such that each applicant $a \in A$ appears in at most one pair, and each college $c \in C$ appears in at most q_c pairs.

Definition 2.3. A **blocking pair** $(a, c) \in A \times C$ for the assignment $\mathcal{A}$ is a pair such that a and c are in each others' preference lists and one of the following holds:

(i) c's quota is met, a prefers c to his current college and c prefers a to at least one of its applicants;

(ii) c's quota is not met and a prefers c to his current college.

Definition 2.4. An assignment $\mathcal{A}$ is **stable** if there is no blocking pair for it in $A \times C$.

It's clear that if we had a blocking pair (a, c), then in case (i), c would like to reject its worst student in order to accept a, and in case (ii), c would simply accept a because it has the room to do so.

Lemma 1.10 generalizes to both the applicant- and college-optimal algorithms, and it follows from it that the assignment is stable and, in fact, applicant- or college-optimal, respectively.

For the applicant-optimal procedure, we can represent the list of applicants to a college using a heap data structure with keys as the applicant ranking for that college, since we only need access to the worst applicant on the list when a college must reject one. Then this lookup is $O(1)$ and insertion and deletion are $O(log(q_c))$. For simplicity's sake, we assume that $\forall a \in A, c \in C, a \in L(c) \Rightarrow c \in L(a)$ to avoid having each college rank every applicant, even those which it would never accept. Otherwise, when $a \in A$ applies to $c \in C$, we must check if $a \in L(c)$ and we'd like to do be able to do this in constant time (in particular, without having to traverse $L(c)$). For $q :=$ the largest quota greater than or equal to n (or 2 if none, since we only need to reject applications if a college's quota is met), the running time will be $O(log(q)(n + \Sigma_{a \in A}|L(a)|)) \subseteq O(log(q)nm)$.

For the college-optimal procedure, there is no need to store each college's list of applicants; we need only know how many offers each college has made. Here, we assume that $\forall a \in A, c \in C, c \in L(a) \Rightarrow a \in L(c)$ to avoid having each applicant rank every college, even those to which they would never apply. The college-optimal algorithm's running time will be $O(m + \Sigma_{c \in C}|L(c)|) \subseteq O(n + m + \Sigma_{a \in A}|L(a)| + \Sigma_{c \in C}|L(c)|) \subseteq O(nm)$, where the second of these three is linear.

3 Stable Roommates

The stable roommates problem also generalizes stable marriages: instead of separate sets of men and women, there is only a single set of n individuals, each with a list the other $n - 1$ individuals sorted by preference. In the paper "An Efficient Algorithm for the "Stable Roommates" Problem", Irving gave an $O(n^2)$-time algorithm to find a stable matching if one exists, and reporting that none exists, otherwise.

Definition 3.1. An **instance** of the stable roommates problem is a set of X of individuals, with size $n \in \mathbb{N}$, and for each $x \in X$, a preference list $L(x)$, that is a permutation of the remaining individuals in X.

We may assume $X = \{1, 2, \ldots, n\}$.

Again, we say x **prefers** y **to** z, if y appears before z in $L(x)$.

Preference lists (not necessarily the original ones) together are called a (preference) **table**, and a table T' is a **subtable** of a table T, if $x \in X$ is in $y \in X$'s list in T' implies the same in T and x comes before z on y's list in T' implies the same in T.

A **table for the instance** is a subtable of the original table. Since the instance is understood to be fixed, we will usually just refer to a table for the instance just as table.

Definition 3.2. A **matching** for an instance of the stable roommates problem is a bijection $\mathcal{M} : X \to X$ such that $\mathcal{M}(x) \neq x$ for all $x \in X$ and $\mathcal{M}(x) = y$ iff $\mathcal{M}(y) = x$ (or $\mathcal{M}(\mathcal{M}(x)) = x$ for all $x \in X$, i.e. $\mathcal{M}$ is its own inverse).

Note that $\mathcal{M} = \{(x, \mathcal{M}(x)) \in X \times X | x \in X\}$.

It follows from this definition that n must be even for a matching to exist, since we do not allow $x \in X$ to be alone ($\mathcal{M}$'s domain is X) or matched with himself ($\mathcal{M}(x) \neq x$).

Equivalently, we can define a matching as a set of unordered pairs in $\mathcal{P}(X)$ (the power set of X) where each individual appears in exactly one pair. In this case, any matching must have size $n/2$.

Blocking pair and **stable matching/solution** and **possible pairing** (**possible partner**) are defined again as in Definitions 1.3, 1.4 and 1.5.

Any stable matching for a stable marriage instance will be a solution to the stable roommates instance constructed by adding to the end of each man's preference list the remaining men and doing the same for the women. Like the stable marriage problem, then, a stable roommates instance may have multiple solutions. Unlike the stable marriage problem, however, not every instance of the stable roommates problem has a solution, e.g.

1 [2 3 4]
2 [3 1 4]
3 [1 2 4]
4 [1 2 3]

There are three matchings, but each has a blocking pair:

{(1,2), (3,4)} is blocked by (2,3); {(1,3), (2,4)}, by (1,2); and {(1,4), (2,3)}, by (1,3).

Note that by the symmetry of the instance, 4's list can actually be an arbitrary permutation of {1, 2, 3} , and the same will still hold.

Irving's algorithm is split up into two phases, the first of which is similar to the stable marriage algorithm, but we now allow anyone to propose or hold proposals (and even both). In pseudocode:

Phase 1

while there remains $x \in X$ who has no proposal held and has not proposed to everyone, do

$next$:= the next on x's list to whom x has not yet proposed

x proposes to $next$

if *next* holds no proposal, then
    next holds x's proposal
else $y :=$ the individual from whom *next* holds a proposal
    if *next* prefers y to x, then *next* rejects *proposer*
    else
        next rejects y
        next holds x's proposal

In words, we let those who have no proposal held to propose to the individual they prefer most of those to whom they've not already proposed, if one exists, but if there is no such individual left, then we stop the procedure.

We assume again that we have a ranking matrix for the individuals as we described in Lemma 1.7, and if not, we can construct one in $O(n)$ time. Then, we can use a linked list to implement the set of individuals who have no proposal held, so that insertion and deletion are $O(1)$, resulting in $O(n)$ preprocessing time to copy X. Alternatively, we can traverse X letting *proposer* be each individual in turn with a nested while-loop inside that updates *proposer* to y if y is rejected by *next*, so that rejected individuals have priority in proposing. In either case, the operations inside the loop(s) are $O(1)$ all together and each iteration corresponds to a single proposal (except possibly the last, if there is no stable marriage), of which most $(n-1)^2$, so the procedure takes $O(n^2)$ time.

Results corresponding to Lemma 1.8, Lemma 1.10, Corollary 1.10.1, which were adapted from Irving's original paper, hold again in this setting with nearly identical proofs (so they are omitted here):

Lemma 3.3. After the moment $x \in X$ is first proposed to, x always holds a proposal.

Lemma 3.4. If $y \in X$ rejects $x \in X$ in Phase 1, then x and y cannot be partners in any stable matching.

Corollary 3.4.1. If during Phase 1, $x \in X$ proposed to $y \in X$, then, in any stable matching

(i) x cannot have a better partner than y;

(ii) y cannot have a worse partner than x.

Corollary 3.4.2. If after Phase 1, $x \in X$ has no proposal held and has proposed to everyone on his list, then no stable matching exists.

Proof. x was therefore rejected by everyone, so by the Lemma, cannot be partners with anyone in a stable matching.

□

Then we would exit reporting so, as we should, but after we find that an individual's list becomes empty after the following reductions to the original table:

Corollary 3.4.3. If, after Phase 1, $y \in X$ holds a proposal from $x \in X$, then y's preference list can be reduced (without eliminating possible partners) by deleting from it

(i) all those to whom y prefers x (i.e. all those after x);

(ii) all those who hold a proposal from a person whom they prefer to y (including those who have rejected y, i.e. all those before x);

In the resulting table,

(iii) y is first on x's list; and x, last on y's;

(iv) in general, b appears on a's list if and only if a appears on b's.

Proof. (i) and (ii) follow from Corollary 3.4.1.(ii); and (iii), from (i) and the parenthesized part of (ii). For (iv), suppose a holds a proposal from c; and b, from d. Then a is on b's list $\iff$ b does not prefer d to a (by (i)), and a does not prefer c to b (by (ii)) $\iff$ a is on b's list (by (ii) and (i)).

□

We will refer to this table as the Phase 1 table.

In fact, for a particular instance, the Phase 1 table is always the same. When no list is empty, the resulting table is an example of a *stable table*:

Definition 3.5. A table T for an instance is **stable** if, in T:

(i) x is first in y's list, abbreviated $f_T(x) = y$, iff y is last in x's list, abbreviated $l_T(y) = x$;

(ii) for $x, y \in X$, x is not in y's list iff y is not in x's iff x prefers $l_T(x)$ to y or y prefers $l_T(y)$ to x (in the original table);

(iii) no list is empty.

(i) and (iii) imply that f_T and l_T are inverse functions $X \to X$, and so must both be bijective, i.e. injective and surjective.

Note that once all of the $f_T(x)$ are determined (or all of the $l_T(x)$ are), then the whole table is determined by (ii).

It should be clear that any table can be made so that (i) and (ii) hold by reducing the lists, although a list may become empty in doing so. In fact, any table for which (i) and (ii) hold is a subtable of the Phase 1 table, and the reductions to obtain the Phase 1 table are exactly those necessary to establish (i) and (ii):

Lemma 3.6. Definition 3.5.(i) and (ii) hold for the Phase 1 table, and so if no list is empty, the table is stable.

Furthermore,

Lemma 3.7. If each list in a stable table T contains exactly one person, then they specify a stable matching.

Proof. By Definition 3.5.(i), the lists specify a matching.

Suppose that $(x, y) \in X \times X$ but x and y are not paired in the matching. Then x and y are not in each others' lists in T. Then, by Definition 3.5.(ii), x prefers the last (i.e. sole) person on his list to y, or y, the last (i.e. sole) person on his list, to x, so that (x, y) cannot be a blocking pair. The matching is therefore stable.

□

Then, using Lemma 3.6, the following holds:

Corollary 3.7.1. If each list in the Phase 1 table contains exactly one person, then they specify a stable matching.

Before we include Phase 2 of the algorithm in pseudocode, we describe and justify it.

We prove that at each step until termination, Definition 3.5.(i) and (ii) hold. The base case is established by Lemma 3.6. Hence, if the reduced lists contain exactly one person each, then we are done by Lemma 3.7 (Corollary 3.7.1). Otherwise, at least one is empty, in which case, we will see that there is no stable matching so that we end the procedure, or at least one has more than one person in it. In this last case, we will search for a *rotation* (an "all-or-nothing cycle" in Irving's original paper) and *eliminate* it: we reduce the preference lists again with respect to this rotation in such a way that Definition 5.(i) and (ii) continue to hold. If any list becomes empty, we exit reporting that no stable matching exists. Otherwise, the inductive step holds.

Definition 3.8. A **rotation** in a stable table is a cyclic sequence $(a_1, b_1), (a_2.b_2), \ldots, (a_r, b_r) \in X \times X$ of pairs of individuals such that, where $a_{r+1} = a_1$ and $b_{r+1} = b_1$:

(i) the a_i are distinct;

(ii) b_i is first on a_i's list ($f_T(a_i) = b_i$) for each i; equivalently, a_i is last on b_i's list for each i ($l_T(b_i) = a_i$);

(iii) b_{i+1} is the second on a_i's list for each i.

First, note that a_i is in b_{i+1}'s list for each i.

Then, a rotation can be depicted in the following way with respect to the reduced preference lists in the stable table:

$$
\begin{array}{rcl}
a_1 & [\ b_1 \quad b_2 \ldots] & b_1[\ \ldots \quad a_r \ldots \quad a_1] \\
a_2 & [\ b_2 \quad b_3 \ldots] & b_2[\ \ldots \quad a_1 \ldots \quad a_2] \\
a_3 & [\ b_3 \quad b_4 \ldots] & b_3[\ \ldots \quad a_2 \ldots \quad a_3] \\
\vdots & & \vdots \\
a_i & [\ b_i \quad b_{i+1} \ldots] & b_i[\ \ldots a_{i-1} \ldots \quad a_i] \\
\vdots & & \vdots \\
a_{r-1} & [\ b_{r-1} \quad b_r \ldots] & b_{r-1}[\ \ldots a_{r-2} \ldots a_{r-1}] \\
a_r & [\ b_r \quad b_1 \ldots] & b_i[\ \ldots a_{r-1} \ldots \quad a_r]
\end{array}
$$

We observe that a rotation is completely determined once one a_i or b_i is known: once a_i is known, (iii) gives us b_{i+1} and (ii), a_{i+1}; once b_i is known, (ii) gives us a_i and we repeat the previous argument. It then follows that since the a_i are all distinct, so must be the b_i.

Also, note that the length of the rotation is $r \geq 2$, since otherwise $b_1 = b_2$, so that $b_1 = b_2$ is both first and second on a_1's list, a contradiction.

Lemma 3.9. A stable table in which a list has length at least 2 contains a rotation.

Proof. To find a rotation, we let $p_1 \in X$ be an individual whose list has length at least 2, and define, inductively:

q_{i+1} = the second in p_i's reduced list

p_{i+1} = the last in q_{i+1}'s reduced list (so q_{i+1} is first in p_{i+1}'s)

until this sequence repeats some p_s, so that $p_{s+r} = p_s$. It follows that each p_i will have at least 2 individuals on his reduced list, for if p_i has only q_i on his list and i is the least such that this occurs, then q_i must have only p_i on his list by Definition 3.5.(i), because q_i is both the first and the last on p_i's list. By uniqueness, it follows that $p_i = p_{i-1}$, contradicting i being the least. Hence, we can, in fact, choose q_{i+1} at each step.

Then, we let $a_i = p_{s+i-1}$ for $i = 1, \ldots, r$ and $b_i = q_{s+i-1}$, for $i = 1, \ldots, r$ to give us a rotation, and we call $p_1, \ldots, p_{s-1}$ the **tail** for the rotation.

□

Tails are not generally unique for a rotation, since instead of starting our search from p_1, starting from any other p_j would have also lead to the same rotation. Also, tails may be empty, as is the case when $p_1 = a_1$.

Then, having found a rotation, we *eliminate* the rotation: we force each b_i to reject a_i and have each a_i propose to b_{i+1}, reducing the preference lists again to satisfy Definition 3.5.(i) and (ii) and proving the inductive step in our reductions. Hence $x \in X$ and $y \in X$ are removed from each other's lists in the rotation elimination if (and only if) one of them is b_i for some i and the other succeeds a_{i-1} in b_i's list. The result for the cycle is that, for each i:

a_i [b_i $b_{i+1} \ldots$] becomes a_i [$b_{i+1} \ldots$], and

b_i [$\ldots a_{i-1} \ldots a_i$] becomes b_i [$\ldots a_{i-1}$].

Additionally, we remove b_i from the lists of the successors of a_{i-1} in b_i's list, so that the removals remain symmetric.

Lemma 3.10. If there exists a rotation $(a_1, b_1), \ldots, (a_r, b_r)$ in a stable table T and the table resulting from the elimination of this rotation, T', has no empty list, then T' is a stable subtable of T':

(i) $f_{T'}(a_i) = b_{i+1}$ for each i;

(ii) $l_{T'}(b_i) = a_{i-1}$ for each i;

(iii) $f_{T'}(x) = f_T(x)$ for $x \in X, x \neq a_i$ for each i, and $l_{T'}(x) = l_T(x)$ for $x \in X, x \neq b_i$ for each i;

(iv) if $p_1, \ldots, p_{s-1}$ is a tail (as described in Lemma 3.9) for the rotation and the 2nd on p_j's list changes after the rotation elimination, then either p_j is matched in T' or $j = s - 1$ (i.e. the last on the tail), but not both.

Proof. Stability of T' and (i)-(iii) follow from the way we eliminate rotations.

For (iv), suppose p_j's 2nd value changed from $q_{j+1} \in X$ after the rotation elimination. Then this occurred either because $f_T(p_j)$ was removed from p_j's list and vice-versa (so $f_{T'}(p_j) = q_{j+1}$ or q_{j+1} was also removed) or p_j and q_{j+1} were removed from each other's lists, so that one of them is b_i for some i and the other succeeded a_{i-1} in b_i's list before the rotation was eliminated.

If $f_T(p_j)$ was removed from p_j's, then by (i) and (iii), it follows that $p_j = a_i$ for some i, contrary to p_j being in the tail. Hence, $p_j = b_i$ or $q_{j+1} = b_i$ for some i.

If $p_j = b_i$, then p_j prefers a_{i-1} to q_{j+1}, where both were on his list but q_{j+1} was 2nd, so that a_{i-1} was first, i.e. $f_T(p_j) = a_{i-1}$, and since $l_{T'}(p_j) = l_{T'}(b_i) = a_{i-1}$, a_{i-1} is the only individual remaining on p_j's list and vice-versa by Definition 3.5.(i), so that $p_j = b_i$ and a_{i-1} are matched.

Otherwise $q_{j+1} = b_i$, i.e. the second on p_j's list was b_i and so p_{j+1} was last on b_i's list, i.e. $p_{j+1} = a_i$. Then p_j was not among the a_k but p_{j+1} was, so that p_{j+1} must have been the first, i.e. $p_{j+1} = a_1$ and so $j = s - 1$ (and $i = 1$).

Why not both? I was unable to prove this (or find a proof).

□

The following lemma, whose proof we leave as an exercise, justifies rotation elimination:

Lemma 3.11. Let $(a_1, b_1), \ldots, (a_r, b_r)$ be a rotation in a stable table T. Then,

(i) in any stable matching contained in T, either a_i and b_i are partners for all values of i or for no value of i;

(ii) if there is such a stable matching in T in which a_j and b_j are partners, then modifying it so that a_i and b_{i+1} are instead partners for each i also gives a stable matching.

Then, the following two results follow immediately, by induction on the Phase 2 steps outlined earlier:

Corollary 3.11.1. If the original problem instance admits a stable matching, then there is a stable matching contained in any of the tables in the sequence of reductions.

Corollary 3.11.2. If one or more among the lists in one of the tables in the sequence is empty, then the original problem instance admits no stable matching.

With that above two corollaries, we've proven the correctness of the procedure, whose remainder we now include:

Phase 2

reduce the preference lists as described in Corollary 3.4.3
if any list becomes empty during the reductions, then
    exit the procedure reporting that there is no stable matching
if, after the reductions, each list has a unique individual in it, then
    return $\mathcal{M} := \{(x, y) \in X \times X \mid y \text{ is in } x\text{'s reduced list}\}$
while there remains an individual in X with at least 2 in his reduced list, do

find a rotation using the tail of the previous one (if any)
eliminate the rotation
if any list becomes empty during the reductions, then
exit the procedure reporting that there is no stable matching
return $\mathcal{M} := \{(x, y) \in X \times X \mid y \text{ is in } x\text{'s reduced list}\}$

The only issue that remains to implement Phase 2 efficiently.

In the first reduction after Phase 1, we keep track of the first individual in X whose reduced list has size at least 2, called $first\_unmatched$.

When Phase 2 starts, to find rotations, we first store an array of length n with entry i indicating whether we've 'visited' $i \in X$ in our search for a rotation. We of course initialize each entry to 'unvisited'. Then, at each step in which we search for a rotation, we store a linked list of visited individuals, the p_i, for $i = 1, \ldots, s + r - 1$, and mark them as visited in the array until we find one that's already been visited, starting with $p_1 = first\_unmatched$ for the first rotation. We eliminate the rotation we've found and then 'unvisit' all of the a_i and remove them from our list.

If there is no tail from the previous rotation, we start at $first\_unmatched$ (which may have to be updated beforehand).

If there is a tail, we start our search from p_{s-1}, the last individual in the tail, to avoid repeating the sequence $p_1, \ldots p_{s-1}$. This is justified by Lemma 3.10, since we have two possibilities. The first is that our new search starting from p_1 will lead back to p_{s-1} as the 2nd individuals on the p_j's lists up to but not necessarily including $j = s - 1$ (by (iv)) as well as the last individuals on the q_js' lists (up until the rotation) remained the same (by (iii)). The second possibility is that once p_j is repeated for some $j > s - 1$ in our new search, it will follow that each p_k, for $k \geq j$, will have at least 2 on his list by induction (in the same way we guaranteed each p_i had at least two on his list for each i to prove Lemma 3.9), so that those p_k are still unmatched and hence the 2nd on their lists remained the same after eliminating the rotation (by (iv)).

Furthermore, reducing the preference lists can be done implicitly by storing the indices of the first, second and last person on an individual's list that should remain in the reduced list and updating these as necessary. We store arrays for these, where $f[x]$, $s[x]$ and $l[x]$ are the corresponding values for x. Then, with the ranking matrix, each removal can then be done in constant time, since it just means comparing the rank of an individual to be removed with the indices of the first and last and updating these appropriately. We only update the value of $s[x]$ during the search for a cycle, and the new value can be found by traversing x's list from $s[x]$ until we find some p who prefers x to $l[p]$, i.e. p and x are in each others' reduced lists, and we update $s[x] := p$.

Theorem 3.12. The running time of Irving's algorithm is $O(n^2)$.

Proof. Phase 1 takes $O(n^2)$ time. Since each removal takes constant time and there are at most $n(n-2)+1$ removals (n for each individual, $n-2$ to reduce an individual's list to a single individual and the $+1$ in the case that one list becomes empty and we stop there), so all removals together in all reductions require $O(n^2)$ time. Checking whether a stable matching is found and constructing it also takes $O(n^2)$ time, as we only scan the list X and pair each $x \in X$ with $f[x] = l[x]$.

$first\_unmatched$ is updated at most n times and $s[x]$, at most $n - 1$ times for each x, corresponding to traversing the lists from left to right at most once, so together, this is again $O(n^2)$.

All that remains is to show that the time spent finding rotations is also $O(n^2)$:

If m rotations are found and eliminated during the entire procedure, we let t_i and r_i denote the lengths of the tail and the rotation for the ith rotation. For $i = 1, \ldots m$, finding the ith rotation took at most $c + d(r_i + t_i - t_{i-1}) + er_i$ operations, for some constants, $c, d, e > 0$, since $r_i + t_i$ is the number of individuals in the list of p_j, but the first t_{i-1} were already visited in the search for the $i-1$th, so we do not revisit them, and we only visit $r_i + t_i - t_{i-1}$ new individuals at this step (where $t_0 = 0$, hence $d(r_i + t_i - t_{i-1})$. Also, after the search, we unvisit the a_j of the cycle, hence er_i.

Note also that since the elimination of the ith rotation means at least $2r_i$ removals from the preference lists, and $r_i \geq 2$ for each i, so that

$$4m \leq 2\Sigma_{i=1}^{m} r_i \leq n^2$$

Then, taking the sum over all the rotations, we get at most

$$\begin{aligned}\Sigma_{i=1}^{m}[c + d(r_i + t_i - t_{i-1}) + er_i] &= cm + (d+e)\Sigma_{i=1}^{m} r_i + d(t_m - t_0) \\ &\leq \frac{c}{2}n^2/4 + \frac{d+e}{2}n^2 + dn,\end{aligned}$$

which is $O(n^2)$.

□

4 Alternate Algorithm for Stable Roommates

Phase 1 and the first part of Phase 2 corresponding to the reductions for Corollary 3.4.3 can be replaced by the following equivalent algorithm (i.e. it produces the same table for any given instance):

Phase 1

while there remains an individual $x \in X$ who has no proposal to another held, do
    if x's list is empty, then
        exit the procedure reporting that no stable matching exists
    $y :=$ the first individual left on x's list
    x proposes to y
    if y holds a proposal from z, then
        y rejects z's proposal
        for each successor x' of x in y's list (which includes z)
            remove x' and y from each other's lists
    y holds x's proposal

The deletions ensure that no one worse than y's current partner is even allowed to propose to y, so that, in a sense, y rejects them before they even get to chance to propose, since if they were to propose, they would be rejected, anyway.

Additionally, proposals made by an individual are again made to worse partners at each step, as in the stable marriage algorithm. Proposals received are also better after each one, since we prevent worse proposals to be made.

We note that if y holds a proposal from z, then z successors in y's list were removed, and y was removed from their lists. Then if x later proposes to y, it means that x was not among z's successors, and also that z's successors in y's list are among x's. Hence, even if x had proposed before z (preventing z from later proposing), the result would have been the same, i.e. the removal of x's successors in y's list and of y from their lists. From this it follows that the table resulting from Phase 1 is always the same, no matter the order in which proposals are made.

The algorithms for the stable marriage and hospitals/residents problems can be modified in the same way, too.

5 Exercises

5.1 Lemma 3.10. Prove Lemma 3.10. Hint: for (ii), first consider the case where $a_i = b_j$ for some i, j and use (i).

5.2 Geometric Stable Roommates. Consider the stable roommate problem where the individuals are elements of the real numbers, $\mathbb{R}$ and x prefers y to z iff $|x - y| < |x - z|$, i.e. the norm replaces the preference lists, and assume there are no ties. Describe an algorithm to find a stable matching in $\Theta(nlog(n))$-time. Hint: dynamic binary search trees.

What about the vector space $\mathbb{R}^d$ for $d \in \mathbb{Z}^+$, where $x = (x_1, x_2, \ldots, x_d)$ and $|x| = \sqrt{(x_1)^2 + \cdots + (x_d)^2}$?

5.3 Median Matching. Given a stable roommates instance with a solution, let $\mathcal{S}$ be a subset of all the stable matchings such that $|\mathcal{S}|$ is odd. Show that $\mathcal{M}$, defined by $\mathcal{M}(x) :=$ the x's median partner in all matchings in $\mathcal{S}$, is a stable matching. Hint: first consider the case where $\mathcal{S}$ contains only 3 stable matchings.

6 References

1. D. Gale and L. S. Shapley: "College Admissions and the Stability of Marriage", American Mathematical Monthly 69, 9-14, 1962.

2. J. Kleinberg, E. Tardos. Algorithm Design. Addison Wesley, 2005.

3. Irving, Robert W. (1985), "An efficient algorithm for the "stable roommates" problem", Journal of Algorithms 6 (4): 577595.

4. D. Gusfield and R. W. Irving, The Stable Marriage Problem: Structure and Algorithms, p. 54; MIT Press, 1989.

7 Bibliographic Notes

1. Robert W. Irving, Stable marriage and indifference, Discrete Applied Mathematics, Volume 48, Issue 3, 15 February 1994, Pages 261-272

2. Dan Gusfield. 1988. The structure of the stable roomate problem: efficient representation and enumeration of all stable assignments. SIAM J. Comput. 17, 4 (August 1988), 742-769.

3. Irving, Robert W.; Manlove, David F. (2002), "The Stable Roommates Problem with Ties", Journal of Algorithms 43 (1): 85105.

4. Tams Fleiner, Robert W. Irving, David F. Manlove, Efficient algorithms for generalized Stable Marriage and Roommates problems, Theoretical Computer Science, Volume 381, Issues 13, 22 August 2007, Pages 162-176.

5. D. E. Knuth. Mariages Stables et leurs relations avec d'autres problmes combinatoires. 1976.

6. Robert W Irving and Paul Leather. 1986. The complexity of counting stable marriages. SIAM J. Comput. 15, 3 (August 1986), 655-667.

7. Iwama, K.; Miyazaki, S., "A Survey of the Stable Marriage Problem and Its Variants," Informatics Education and Research for Knowledge-Circulating Society, 2008. ICKS 2008. International Conference on , vol., no., pp.131,136, 17-17 Jan. 2008

8. Roth, Alvin E, 1984. "The Evolution of the Labor Market for Medical Interns and Residents: A Case Study in Game Theory," Journal of Political Economy, University of Chicago Press, vol. 92(6), pages 991-1016, December.

9. Roth, A.E., "On the Allocation of Residents to Rural Hospitals: A General Property of Two-Sided Matching Markets," Econometrica, 54, 1986, 425-427.

10. E. Ronn, NP-complete stable matching problems, J. Algorithms, 11 (1990), pp. 285304

11. A. Kato, Complexity of the sex-equal stable marriage problem, Japan Journal of Industrial and Applied Mathematics (JJIAM), Vol. 10, pp. 119, 1993.

12. David F Manlove, Robert W Irving, Kazuo Iwama, Shuichi Miyazaki, Yasufumi Morita, Hard variants of stable marriage, Theoretical Computer Science, Volume 276, Issues 12, 6 April 2002, Pages 261-279.

13. Magns M. Halldrsson, Kazuo Iwama, Shuichi Miyazaki, and Yasufumi Morita. 2002. Inapproximability Results on Stable Marriage Problems. In Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN '02), Sergio Rajsbaum (Ed.). Springer-Verlag, London, UK, UK, 554-568.

14. Chung-Piaw Teo and Jay Sethuraman. 1998. The Geometry of Fractional Stable Matchings and its Applications. Math. Oper. Res. 23, 4 (November 1998), 874-891.

15. Jay Sethuraman, Chung-Piaw Teo, and Liwen Qian. 2006. Many-to-One Stable Matching: Geometry and Fairness. Math. Oper. Res. 31, 3 (August 2006), 581-596.

16. K. Iwama, S. Miyazaki, and K. Okamoto. Stable roommates problem with triple rooms, 2007.

17. Katarna Cechlrov, On the complexity of exchange-stable roommates, Discrete Applied Mathematics, Volume 116, Issue 3, 15 February 2002, Pages 279-287.

18. Cechlarova, Katarina and Manlove, David (2005) The Exchange-Stable Marriage Problem. Discrete Applied Mathematics 152(1-3):109-122.

19. Eric Mc Dermid, Christine Cheng, Ichiro Suzuki, Hardness results on the man-exchange stable marriage problem with short preference lists, Information Processing Letters, Volume 101, Issue 1, 16 January 2007, Pages 13-19.

20. Esther M. Arkin, Sang Won Bae, Alon Efrat, Kazuya Okamoto, Joseph S.B. Mitchell, Valentin Polishchuk, Geometric stable roommates, Information Processing Letters, Volume 109, Issue 4, 31 January 2009, Pages 219-224.

21. Prasad Chebolu, Leslie Ann Goldberg, Russell Martin, The complexity of approximately counting stable matchings, Theoretical Computer Science, Volume 437, 15 June 2012, Pages 35-68

22. K. Cechlarova and T. Fleiner. On a generalization of the stable roommates problem. ACM Trans. Algorithms, 1(1):143156, 2005.

23. K. Cechlrov, V. Valov, The stable multiple activities problem, IM Preprint, Series A, No. 1/2005, 2005.

24. Viera Borbelov, Katarna Cechlrov, On the stable b-matching problem in multigraphs, Discrete Applied Mathematics, Volume 156, Issue 5, 1 March 2008, Pages 673-684.

25. Robert W. Irving, Sandy Scott, The stable fixtures problemA many-to-many extension of stable roommates, Discrete Applied Mathematics, Volume 155, Issue 16, 1 October 2007, Pages 2118-2129.

26. David J. Abraham, Robert W. Irving, David F. Manlove, Two algorithms for the Student-Project Allocation problem, Journal of Discrete Algorithms, Volume 5, Issue 1, March 2007, Pages 73-90.