The Structure of the Stable Rommate Problem: Efficient Representation and Enumeration of all Stable Assignments

Dan Gusfield

YALE/DCS/TR 482
June 1986

Research funded in part by ONR grant N00014-82-K-0184 and NSF grant MCS-81/05894

The Structure of the Stable Roommate Problem: Efficient Representation and Enumeration of all Stable Assignments

ABSTRACT

The stable *roommates* problem is a well known problem of matching 2n people into disjoint pairs to achieve a certain type of stability. The problem strictly generalizes the better known stable *marriage* problem. It has been previously shown [IL] that the set of stable marriages, for a given instance, can be compactly represented and this representation has been exploited to yield a number of very efficient algorithms concerned with stable marriage [ILG], [G], [GILS]. In this paper we generalize the structure of the stable marriages to obtain two efficiently computed, small, implicit representations of the set of all stable roommate assignments, for any given instance. One representation is a partial order Π on $O(n^2)$ elements such that the stable assignments are in one-one correspondence with certain easily recognized subsets of Π. Partial order Π is a strict generalization of the stable marriage representation. The second representation is an efficiently constructed, undirected graph G with $O(n^2)$ nodes, such that there exists a one-one correspondance between the *maximal* (not maximum) independent sets of G and the stable roommate assignments. In either representation, G or Π, given a set representing a stable assignment, the assignment itself can be constructed in $O(n^2)$ time. We also give an algorithm to generate each stable assignment for any given instance in $O(n^2)$ time per assignment. The efficiency of this method depends heavily on special properties of the stable assignment problem developed in this paper. Finaly, we give a succinct characterization of the set of all "stable pairs", those pairs of people who are roommates in at least one stable assignment, and we give an $O(n^3 \log n)$ time algorithm to find them all.

Note: This report supersedes and replaces Yale Computer Science technical report TR-435 dated November 1985, entitled "Roommate Stability Leads to Marriage: The Structure of the Stable Roommate Problem". The present report shares a large amount of technical material with the earlier report, but has a very different focus, that of representing and enumerating the stable assignments. The focus of the first report was an efficient reduction from the stable roommate problem to the stable marriage problem. However, the proof is in error, and the main consequence of the claimed reduction, the claimed partition theorem, is not true. I thank Sally Floyd for first catching this error, and Robert Irving for also finding the error.

Table of Contents

The Structure of the Stable Roommate Problem: Efficient Representation and Enumeration of all Stable Assignments

Dan Gusfield
Yale University, Department of Computer Science

1. Introduction

The stable *roommates* problem is a well known problem of matching 2n people into disjoint pairs to achieve a certain type of stability. The input to the problem is a set of 2n preference lists, one for each person i, where person i's list is a rank ordering (most preferred first) of the 2n-1 people other than i. A roommate assignment A is a pairing of the 2n people into n disjoint pairs. Assignment A is said to be *unstable* if there are two people who are not paired together in A, but who each prefer the other to their respective mates in A; such a pair is said to *block* assignment A. An assignment which is not unstable is called *stable*. An instance of the stable roommates problem is called *solvable* if there is at least one stable assignment. It is known [GS, L, K, PTW] that there are unsolvable instances of the stable roommate problem; the problem of finding an efficient algorithm to determine if an instance is solvable was proposed by Knuth [K] and only recently solved by R. Irving [I].

The stable roommates problem is closely related to, and is a strict generalization of, another well-known problem, the stable *marriage* problem. In the stable marriage problem, the 2n people consist of n men and n women, and each pair is constrained to consist of a man and a women. Each man ranks only the women and each woman ranks only the men, and an assignment in this problem is called a *marriage* (from here on, the word "assignment" will be used only for roommate assignment). A marriage M is unstable if there is a man and a woman who are not married to each other in M, but who mutually prefer each other to their respective mates in M. It is easy to reduce an instance of the stable marriage problem to an instance of the stable roommates problem. However, in contrast to the stable roommates problem, it is well known [GS] that every instance of the stable marriage problem is solvable, i.e. has at least one stable marriage, and for any instance, one stable marriage is easy to find.

For a given instance of the stable marriage problem, there may be many distinct stable marriages, and as a function of n, the number of stable marriages can grow exponentially [K]. Despite this exponential growth, for any given problem instance, there exists an extremely small, efficiently computed, implicit representation of the set of all stable marriages for the instance,

where any particular stable marriage can be extracted from the representation very quickly. The structure of the stable marriages which gives rise to this implicit representation was first made explicit by Irving and Leather [IL], and, as pointed out in [GILS], can also be seen via a more algebraic approach based on the theory of distributive lattices. This efficiently constructed representation of the set of all stable marriages was exploited in [IL], [ILG], [G], [GILS] to solve a number of problems related to stable marriage. For example, for any given instance of the stable marriage problem, the implicit representation of all the stable marriages can be constructed in $O(n^2)$ time, and thereafter, each stable marriage can be generated from the representation in O(n) time per marriage [G], which is time optimal.

In this paper we generalize the structure of the stable marriages to obtain two efficiently computed, small, implicit representations of the set of all stable roommate assignments, for any given instance. One representation is a partial order Π on $O(n^2)$ elements such that the stable assignments are in one-one correspondence with certain easily recognized subsets of Π. Partial order Π is a strict generalization of the stable marriage representation given in [IL] and [GILS]. The second representation is an efficiently constructed, undirected graph G with $O(n^2)$ nodes, such that there exists a one-one correspondance between the *maximal* (not maximum) independent sets of G and the stable roommate assignments. In either representation, G or Π, given a set representing a stable assignment, the assignment itself can be constructed in $O(n^2)$ time. We then give an algorithm to generate each stable assignment for any given instance in $O(n^2)$ time per assignment. This compares favorabley to the $O(n^3)$ time, per marriage, method given by Knuth [K] to generate all stable *marriages*. The faster method given in this paper doesn't follow immediately from the representations above, since the fastest known time to generate maximal independent sets in a general graph on $O(n^2)$ nodes runs in $O(n^4)$ time per independent set, and methods based on general partial orders appear even less efficient. Finaly, we give a succinct characterization of the set of all "stable pairs", those pairs of people who are roommates in at least one stable assignment, and we give an $O(n^3 \log n)$ time algorithm to find all the stable pairs.

The results and algorithms in this paper are obtained by close examination of Irving's algorithm [I] which finds *one* stable assignment if there is one, and else reports that no stable assignment exists. Hence we will begin by describing algorithm I.

2. Algorithm I and its execution tree D

2.1. Algorithm I

Algorithm I successively deletes entries from preference lists until either each person has only one entry on its list, or until someone has no entries. In the first case, the entries specify a stable roommate assignment, and in the second case, there are no stable assignments; the algorithm runs in time $O(n^2)$. The algorithm is divided into two phases. In phase one, entries are removed from lists, but no stable assignments are affected, i.e. if j is removed from i's list, then (i,j) is a pair in no stable assignment. In phase 2, the removed entries may affect stable assignments, but the invariant is maintained at each iteration, that *if* there is a stable assignment in the lists before the current iteration of removals, *then* there is a stable assignment in the lists resulting from the iteration of removals. Hence if any list becomes empty in either phase, there can be no stable assignment. Before describing the algorithm, we need the following definitions.

Definition: The current set of lists at any point in the algorithm is called a *table*.

Definition: Let e_i denote a person. At any point in the algorithm, h_i will denote the current head of person e_i's list, and s_i will denote the current second entry on e_i's list.

Definition: At any point in algorithm I, a person e_i is said to be *semi-engaged* to h_i if and only if e_i is the bottom entry in h_i's list. A person who is not semi-engaged is called *free*. A person may alternate between being free and semi-engaged.

Note that semi-engagement is not a symmetric relation. However, if everyone is semi-engaged, then the set of list heads is a permutation of the 2n people. We now describe algorithm I.

Phase 1 of algorithm I iterates the following:

> If there is an empty list, then terminate algorithm I; there is no stable assignment.
>
> Else, if everyone is semi-engaged, then go to phase 2.
>
> Else, pick an arbitrary free person e_i, and execute the following operations for each person k who is ranked below e_i on h_i's list in T: remove k from h_i's list, and remove h_i from k's list.

Note that throughout phase 1, person i is on j's list if and only if j is on i's list. Hence in a step where e_i becomes semi-engaged to h_i, if there is a person p who is semi-engaged to h_i just before that step, then p is (automatically) not semi-engaged to h_i after that step; This follows since at the start of the step, p must be below e_i on h_i's list, so during that step, p is removed from h_i's list, and h_i is removed from p's list. After the step, p might be free, or it might have become (automatically) semi-engaged to the new head of its list.

The set of lists at the end of phase 1 is called the *phase 1 table*. It is proved in [I] that if j is

missing from i's list in the phase 1 table, then there are no stable assignments which pair i to j. Hence if some list in the phase 1 table is empty, there are no stable assignments. Otherwise, when phase 1 terminates with everyone semi-engaged, j is the head of i's list if and only if i is the bottom of j's list, and so the set of head entries of the phase 1 table are a permutation of the 2n people.

Figure 1a gives an instance of the stable roommate problem, and 1b shows the three stable assignments for the table. Figure 2a shows the phase 1 table for the example.

Phase 2

Throughout phase 2 all the people remain semi-engaged, although who they are semi-engaged to may change. Hence at any point in phase 2, j is the head of i's list if and only if i is the bottom of j's list. It will also be true that i is on j's list if and only if j is on i's list. Phase 2 starts with the phase 1 table and removes entries from lists in a way similar to phase 1, but the selection of lists is more constrained. We first need some definitions.

Definition: In a table T, an *exposed rotation* R is an ordered subset of people $E = \{e_1, e_2, \ldots, e_r\}$, such that $s_i = h_{i+1}$, for all i from 1 to r, where i+1 is taken modulo r. Note that since the order of E is cyclic, the actual selection of which element in E is named e_1 is arbitrary, but that selection determines the rest of the ordering.

Figure 2b shows two rotations that are exposed in the phase 1 table of the running example.

We will often write "R = (E,H,S)", where H is the set of head entries of E ordered to correspond to the order of E, and S is the set of second entries of E, with corresponding order. Note that, *as sets*, S = H, and that, as *ordered sets*, S is a (backwards) cyclic rotation of H; when that point is central, we will write $S = H^r$. We will sometimes say that "e is in R" to mean that e is the E set of R; we will also say that "(e,h) is a pair in R" to mean that $e = e_i$ and $h = h_i$ for some e_i in E.

Definition: If R = (E,H,S) is an exposed rotation in table T, then the *elimination* of R from T is the following operation: for every s_i in S, remove every entry below e_i in s_i's list in T, i.e. move the bottom of s_i's list up to e_i (from e_{i+1}). Then remove s_i from k's list, for each person k who was just removed from s_i's list.

Notice that if all people are semi-engaged in a table T before a rotation elimination, then all people are semi-engaged after that elimination; hence, one affect of the elimination is to move the head of e_i's list down one place, for each e_i in R, i.e. e_i becomes semi-engaged to the s_i of table T. Figure 2c shows the table resulting from eliminating R_1 from the phase 1 table.

Phase 2 of the algorithm is simply:

```
While some person has more than one entry on his list, and no list is empty,
find and eliminate a rotation.

If every person has exactly one entry on his list, then pairing each
person with their head entry specifies a stable assignment.

If there is an empty list, then there are no stable assignments.
```

Figure 2d completes the execution of phase 2, eliminating rotations R_3 and R_4.

2.2. Correctness of algorithm I

The correctness of algorithm I is proved in [I], and will not be fully repeated here. However, we need the statements of the central lemmas that prove correctness, and we need to extend some of them; we will give proofs of the extended lemmas.

Definition: If T is a table, then roommate assignment A is said to be *contained in*, or *in*, T, if and only if every pair in A is in T, i.e. i is on j's list, and j is on i's list for each pair (i,j) in A.

The following lemmas imply the correctness of algorithm I.

Lemma 2.1 [I]: If T is a table (in phase 2) where no list is empty, and at least one person has more than one entry, then there is a rotation exposed in T.

Lemma 2.1 will be proved and extended in the next section.

Lemma 2.2 [I]: Let R = (E,H,S) be an exposed rotation in T, and let A be any stable assignment contained in T. If $e_1 \in E$ and (e_1, h_1) is a pair in A, then (e_i, h_i) must also be a pair in A, for every e_i in E.

Lemma 2.2 will be proved and extended in the next section.

Lemma 2.3 [I]: If rotation R = (E,H,S) is exposed in T, and there exists a stable assignment in T where $e_1 \in E$ pairs with h_1, then there also exists a stable assignment in T where e_1 does not pair with h_1, and by lemma 2.2, no e_i pairs with h_i, for any e_i in E.

Lemma 2.4 [I]: If the algorithm ends with a single entry on each list, then pairing each person to that entry gives a stable assignment.

2.3. Extensions of the central lemmas

We will prove lemmas 2.1 and 2.2 in order to extend them.

Proof of lemma 2.1: Let e_i be a person who has at least two entries, h_i and s_i, on its list in T. Since the head entries are a permutation of the people, and $s_i \neq h_i$, there must be a person e_j such that $s_i = h_j$. We claim that e_j must have two or more entries on its list. If not, then h_j is

its only entry, and so e_j is the only entry on h_j's list; To see this, note that h_j is both the head and bottom entry on e_j's list, so e_j must also be both the head and bottom of h_j's list. But, $h_j = s_i$ which is on e_i's list, so e_i (which can't be e_j) must also be on h_j's list, and so both h_j and e_j must have at least two entries on their lists. Repeating this argument, we must eventually cycle, in which case a rotation has been found. □

Definition: The proof above gives an (implicit) algorithm for finding a rotation R, starting from any person e who has at least two entries on its list in T. Let e_1 denote the person who is visited twice by the algorithm (i.e. where the cycle is detected). Every person who is visited before the first visit to e_1 is said to be on a *tail* of R, and the other people are in the *body* of R.

Note that in a given table, an exposed rotation may have many tails, and in a different table, the same exposed rotation may have different tails. This is illustrated in the example of figure 2. We will need the following extension of lemma 2.1.

Corollary 2.1: If e is a person with two or more entries on its list in table T, then e is either in a tail or in the body of a rotation exposed in T.

The following lemma extends lemma 2.2.

Lemma 2.5: Let R be a rotation exposed in table T, and (e_i, h_i) a pair in R. If A is a stable assignment contained in T where e_i pairs with h_i, and if (e,h) is ***any*** pair in ***either*** the body of R or a tail of R, then (e,h) must be a pair in assignment A.

Proof: Let e_j be a person either in R or in a tail of R. If e_k is any other person such that $s_k = h_j$, then in A, e_k must pair with h_k if e_j pairs with h_j; If not, then e_k must be paired with a person below s_k on its list, since h_k is already paired with e_j, and A is in T. But s_k is the head of e_j's list in T, so e_j must be the bottom of s_k's list, and since s_k is on e_k's list, e_k is on s_k's list, and is preferred to e_j by s_k. Hence e_k and s_k would block A. It follows that if e_j is in the body of R, and if e_j pairs with h_j in A, then every e_i in R must pair with h_j in A. Now consider any tail of R (relative to T). If (e,h,s) is the last triple of the tail, then $s = h_i$ for some e_i in R, so (e,h) must be a pair in A, and the implication follows backwards along the tail. Hence in A, each person in the tail must also pair with the head person on their list in T. □

2.4. The execution tree D

Algorithm I is guaranteed to produce a stable assignment if there is one. However, in this paper we are concerned with the the structure of the set of all the stable assignments for a particular instance; most of what we will deduce will be by examining the possible executions of algorithm I. Hence we need the following

Theorem 2.1: If A is any stable roommate assignment, then there is an execution of algorithm

I which produces A.

Proof: Let T be any table obtained from a (partial) execution of algorithm I, where stable assignment A is in T. If in T, the head of each persons list is their partner in A, then, as in the proof of lemma 2.1, each list has only a single entry, and so T is the final table of an execution of algorithm I, and A is the resulting stable assignment. So, assume that there is a person p whose partner in A is not the head element of p's list in T. Hence p's list has at least two entries, and, by corollary 2.1, p is either in the body or in a tail of a rotation R = (E,H,S) exposed in T. We claim that no person e_i in the body of R pairs with h_i in A. This follows directly from lemma 2.5, since if (e_i, h_i) is a pair in A, then p's partner in A must also be its head entry in T, contradicting the selection of p.

We will show that when R is eliminated from T, assignment A is still contained in the resulting table. The elimination of R from T can be viewed as a two step process. First, the head of each element e_i in R is moved down one position to h_i. By the argument in the paragraph above, assignment A is in the table after these moves. Hence in A, each e_i in E must be paired with s_i or below in its list, and for the stability of A, it follows that each s_i in S must be paired with e_i or above, in its list. Hence A will be in the remaining table if, for each s_i in S, we remove all the elements below e_i in s_i's list, and remove s_i from the lists of each of these elements. But these are exactly the elements that are removed when R is eliminated from T. Hence rotation R can be eliminated from T, creating a smaller table T′ which still contains the stable assignment A. The theorem follows by repeating this argument until no rotations remain. □

Corollary 2.2: Let R = (E,H,S) be an exposed rotation in table T, and let T′ be the table after eliminating R from T. If e_i is any person in E, then T′ contains all stable assignments that T contains, except for those assignments where e_i mates with h_i.

Definition: We use D to refer to the resulting execution tree, when, for a given phase 1 table, phase 2 of algorithm I is executed in all possible ways. Each node x in D represents the table T(x), which is the current state of the algorithm at node x. Each edge out of x is labelled with a rotation which is exposed in T(x), and which is the next rotation eliminated from T(x) on that execution path out of x. We use D(x) to denote the subtree of D rooted at x.

Note that D is defined only for the the phase 2 executions. In the remainder of the paper, when we talk about algorithm I, we will be referring to phase 2, unless we specifically state otherwise.

2.4.1. Naive enumeration of all stable assignments

Given Theorem 2.1, we could generate all stable assignments by forcing all possible executions of Irving's algorithm. This would be simple to do, but would be terribly inefficient, as it would most often generate the same stable assignment several times. However, we will show in this paper that an efficiently implemented modification of this naive approach generates each stable assignment exactly once, at a cost of $O(n^2)$ time per assignment. Although the modification is simple, its proof of correctness and time is not, and most of this paper centers on developing the needed tools for the proof. We will return to the enumeration problem after examining the structure of D, and the rotations in the the next several sections.

3. Basic Lemmas

In this section we develop the basic (technical) tools and definitions that will be used in the rest of the paper. Before going on, it is useful to examine the execution tree D in a running example (see figure 3). Three initial observations stand out: first, if P and P′ are paths in D that lead to the same stable assignment, then the edges of P and P′ are marked with the same *set* of rotations, although in different order; second, every path in D has the same length; and third, many of the rotations seem to come in dual pairs as defined below.

Definition: If $R = (E,H,S)$ is a rotation in D then we define R^d to be the triple (S,E,E^r), where S and E have the same order in R^d as they have in R. Note that with this definition $(R^d)^d = R$. Note also that R^d has the *form* of a rotation; If R^d is actually a rotation in D (i.e. is a rotation exposed in some table), then we call R and R^d a ***dual pair*** of rotations. Any rotation without a dual is called a ***singleton*** rotation.

In the example, rotations R_1 and R_3 are singletons, and (R_4, R_5) and (R_2, R_6) are each a dual pair of rotations. With this terminology, we can make a more precise observation, which replaces the second and third observations above: Each path from the root to a leaf in D contains every singleton rotation, and exactly one of each pair of dual rotations. We will prove that these observations are facts which hold for any execution tree D of algorithm I, and these facts will then be exploited to reveal the structure of the set of stable roommate assignments. However, we first need several technical definitions and lemmas.

Lemma 3.1: If $R = (E,H,S)$ and $R^d = (S,E,E^r)$ are dual rotations that are both exposed in a table T, then in T each list of $E \cup S$ has exactly two elements.

Proof: This follows simply from definition of duals, and the fact that person i is the head of j's list in any table if and only if person j is the bottom of i's list in that table. □

Definition: For a table T, the *active part* of T is the subtable of T consisting of those lists which contain more than one person.

Lemma 3.2: If R = (E,H,S) and R^d = (S,E,E^r) are both exposed in T, then the active part of the table resulting from eliminating R from T is the same as the active part of the table resulting from eliminating R^d from T. Further, that active part is just the active part of T minus the lists of E ∪ S.

Proof: This follows directly from the definitions dual rotations and rotation elimination, and lemma 3.1 above. □

Definition: Let T be a table and R = (E,H,S) be a rotation. If there is a subset of elements of T which form a table T′, such that R is exposed in T′, then we say that R is *embedded* in T. Note that the definition does not require that some execution of algorithm I actually exposes R.

Lemma 3.3: If R and R^d are dual rotations, then R is embedded in table T if and only if R^d is.

Proof: This follows directly from the definition of duals, and the fact that i is on j's list if and only if j is on i's list. □

Definition: Let R be a rotation exposed in table T, and let T(R) be the table resulting from eliminating R from T. If rotation R′ = (E′, H′, S′) is embedded in T but not in T(R), then we say that R *removes* R′ from T. Note that for R to remove R′ from T all that is required is that h'_i or s'_i not appear on e'_i's list in T(R), for at least one e'_i in R′.

Definition: A path P in D is said to *contain* the rotations that label the edges of P.

Lemma 3.4: If P is a path from the root of D to a node x in D, and P′ is a path from the root to a node x′, and P and P′ *contain* the same rotations in different order, then table T(x) and table T(x′) are identical. Hence a table is determined from the phase 1 table by the set of rotations leading to it, not by their order.

Proof: It is clear from the way that elements are removed in phase 1 and phase 2, that at any point in phase 2, the current table T is determined by the phase 1 table and the bottom elements of each list in T. In phase 2, the bottom element of person i's list is changed only if person i is in the S set of an eliminated rotation. Hence if i is not the second element in any rotation on the path, then i's bottom element in T is its bottom in the phase 1 table; otherwise, the bottom of i's list in T is given by the person p who i most prefers, such that i is the second element in p's list in some rotation on the path to T. □

Definition: The set of rotations that appear on a path from the root to a leaf in D is called a *path set*. We will use this term when the order of the rotations is not important.

Lemma 3.4 shows a mapping from the path sets onto the stable assignments, but does not show the converse, i.e. it is still possible that two path sets generate the same stable assignment.

We will later show that the mapping is in fact one-one. i.e. that any two paths in D which lead to the same table must contain the same set of rotations. This fact, which is more difficult to prove than Lemma 3.4, will be central in the efficient enumeration of the stable assignments.

We are now ready to state and prove the first non-trivial technical lemma.

Lemma 3.5: If R = (E,H,S) and R′ = (E′,H′,S′) are two distinct rotations exposed in a table T, then R removes R′ from T if and only if R′ = R^d. Hence the only way to remove an exposed rotation is to explicitly eliminate it or its dual rotation, if it has one.

Proof: One direction is trivial. If R and R^d are dual rotations, then since one is embedded in T if and only if the other is, the elimination of R must remove R^d. To prove the other direction, suppose that R and R′ are exposed in T, and that R $\neq$ R′ eliminates R′ from T. We will show that R′ must be R^d. When R is eliminated, person $s_i \in S$ is removed from the list of a person p if and only if p is below e_i on s_i's list in table T. Clearly, these removals of individual people from T affect the lists of people in E′ only if s_i is in H′ (hence in S′), or if s_i is in E′. To see that the first case is not possible, recall that in table T the elements in the H column are a permutation of the 2n people, and that in each *rotation* R, the set of S elements and H elements in R are the same. So even though the S column of table T is not necessarily a permutation (i.e. a person can appear more than once in the S column), no person can appear in the S set of more than one rotation exposed in T. Hence $S \cap S' = S \cap H' = \emptyset$, and so the first case is not possible. Hence the elimination of R removes R′ from T only if some s_i is in $S \cap E'$. For ease of discussion, assume wlog that $s_1 \in S \cap E'$, and that $s_1 = e'_j$.

If $e_1 \neq h'_j$, then the change of the bottom of s_1 to e_1 (the consequence of eliminating R) will not affect R′, so we assume also that $e_1 = h'_j$. Let T(R) be the table resulting from eliminating R in T. Since the bottom of s_1's list moves up to e_1, which is the head of s_1's list in T (since $s_1 = e'_j$ and $e_1 = h'_j$), s_1's list in T(R) contains only the single element e_1. We claim that if $e'_i \in E'$, then e'_i's list in T(R) must also contain only a single element. If not, then in T(R), e'_i be on a tail that leads to no rotation. To see this, note first that H′ cannot be affected by the elimination of R, and if e'_i has two elements in its list in T(R), then its first two elements in T(R) are the same as in T. Hence, following the proof of Lemma 2.1, the unique path from e'_i in T(R) is the same as in T, but that path can't form a cycle, since it will encounter a member of R′ (s_1 or earlier) which has only one element on its list. Hence if e'_i has more than one element on its list in T(R) then it will be on a tail leading to no rotation. But this contradicts Corollary 2.1, so in T(R) each e'_i in E′ contains only a single element in its list. Now R′ is exposed in T, so each e'_i in E′ has two or more elements on its list in T, so the affect of eliminating R in T is to move the bottom of each e'_i in R′. But this is possible only if for each $e'_i \in E'$, $e'_i = s_k$ and $h'_i = e_k$, for some e_k in R.

So we now know that if R removes R′ from T, then *as sets*, E′ = H, and H′ = E = S′. This is necessary if R′ = R^d, but in order to actually prove that equality, we need to show that the order inside the sets is correct. We already know that the correspondence between E′ and H′ is correct, i.e. that $e'_i = s_k$ and $h'_i = e_k$, for the same k. So, assuming wlog that $s_1 = e'_1$, we must show that $s'_i = e_{i+1}$ for each i, where i+1 is taken (mod r), and r is the size of R. To do this, we first note that in T the list of every element in R contains exactly two elements; this follows from what we just showed, since in T, each e_i in R is the head of s_i's list, so s_i is the bottom of e_i's list, so each element in R has exactly two people on its list in T. But then each e_i can be only on the list of h_i or s_i in T. Further, e_i appears once in H′ and once in S′. Now e_i is the head of s_i's list in T, so e_i must be the second element in h_i's list in T. So $e'_i = s_i = h_{i+1}$, and the second element on h_{i+1}'s list is e_{i+1}, so $s'_i = e_{i+1}$, as claimed. Hence if R removes R′ from T, then R′ = R^d. □

Later in the paper we will strengthen this theorem to show that if R is exposed in T, and R′ is *embedded* in T, but perhaps not exposed, then R removes R′ from T if and only if R′ = R^d.

4. The structure of D

In this section we examine the structure of D, as this structure will reveal the structure of the set of stable assignments.

4.1. Covering Rotations

Definition: Let x be a node in D and D(x) the subtree of D rooted at x. The *active part* of D(x) is the tree D(x) where at each node y in D(x), T(y) is replaced by the active part of T(y). Note that the edge labels of D(x) do not change.

Definition: If R and R^d are dual rotations, and path P in D contains *either* of them, then we say that P *covers* R *and* R^d. If R is a singleton, and P contains it, then then P covers R.

Lemma 4.1: Let x be a node in D with associated table T(x). Every path from x to a leaf in D(x) covers the same set of rotations.

Proof: Let d(x) denote the maximum number of edges on any path from x to a leaf in D. The theorem will be proved by induction on d(x). For d(x) = 1, if there is only one edge out of x (i.e. only one rotation exposed in T(x)) then the basis is trivially true. If there are two rotations R and R′ exposed in T(x), then, by lemma 3.5, they must be duals of each other, since eliminating either one of them results in a table with no rotations (i.e. each removes the other). Similarly, there cannot be more than two rotations in T(x), since the elimination of any of them removes them all. So the basis is proved.

Assuming that the theorem holds for d(x) $\leq$ k, let x be a node in D where d(x) = k+1, and let

z be a child of x such that d(z) = k; by the induction hypothesis, all paths from x through z to a leaf must cover the same rotations; let P be any such path, and let R be the rotation labelling the edge (x,z). If R is the only rotation exposed in T(x) then there is nothing to prove, so let y be another child of x, and let R′ be the rotation on the edge (x,y). We will show that every path from x through y to a leaf of D(y) covers the same set of rotations as P.

If $R' = R^d$, then, by lemma 3.2 (since both R and R^d are exposed in T(x)), the *active* parts of T(z) and T(y) are identical, and hence the active parts of D(z) and D(y) are identical. Further, d(z) = k, and $d(y) \leq k$, so each path from z covers the same rotations, and each path from y covers the same rotations, so, since the subtrees from z and y are identical, any path from z must cover the same rotations as any path from y. Then every path from x through y covers the same rotations as P.

If $R' \neq R^d$, then, by lemma 3.5, R is still exposed in table T(y), and R′ is still exposed in table T(z). Let z′ be the node associated with the table obtained by eliminating R′ from T(z), and let y′ be the node associated with the table obtained by eliminating R from T(y); and let P(z′) and P(y′) be paths from x to leafs in D that pass through z′ and y′ respectively. Now the set of rotations on the path from the root of D to z′ is exactly the same as the set of rotations on the path to y′, hence by lemma 3.4, T(z′) is identical to T(y′), and so D(z′) = D(y′). It follows, as in the case above, that P(z′) and P(y′) cover the same set of rotations. But, P(z′) covers the same set as P, and since $d(y) \leq k$, any path out of y covers the same set of rotations as P(y′), hence covers the same set as P. Node y was an arbitrary child of x such that $y \neq z$, so the theorem is proved. □

By definition, every rotation is exposed somewhere in D, hence the major consequence of this theorem is the following

Path Theorem

Theorem 4.1: Every path from the root of D to a leaf covers all the rotations. Further, since no path can contain both a rotation and its dual, each path contains every singleton and exactly one of each dual pair of rotations.

Corollary 4.1: Every path in D from the root to a leaf has the same length.

Hence the observations in the example hold in general.

We can now strengthen lemma 3.5.

Corollary 4.2: If R is exposed in table T, and $R' \neq R$ is *embedded* in T, then R removes R′ if and only if $R' = R^d$.

Proof: Clearly, R removes R^d whether R^d is exposed or not. To prove the converse, let x be a

node in D with associated table T(x), and let y be the child of x obtained by eliminating R from T(x). Since R′ is embedded in T(x), neither R′ nor R'^{d} are on the path from the root to x. If R removes R′, it removes R'^{d} also, so neither of these rotations is on any path from x to a leaf. Hence to avoid contradicting theorem 4.1, it follows that $R' = R^{d}$. □

5. The structure of the rotations and stable assignments

In this section we derive two compact representations of the set of all stable assignments. We first need a few more technical observations.

5.1. Unique elimination

Definition: Let e_i, h_i, s_i be a triple in rotation R; hence when R is eliminated from any table it is exposed in, the bottom of s_i's list moves from e_{i+1} to e_i, where i+1 is taken mod r. Let A(R,i) denote the set of people on s_i's original list between e_i and e_{i+1}, including e_i but excluding e_{i+1}. Similarly, let B(R,i) be the people between e_i and e_{i+1}, including e_{i+1} but excluding e_i.

Lemma 5.1: For any e_i in R, R is the only rotation whose elimination moves the bottom of s_i's list *to* a person in A(R,i), and is the only rotation whose elimination moves the bottom of s_i's list *from* a person in B(R,i).

Proof: Let R′ be a different rotation whose elimination moves the bottom of s_i's list to a person p in A(R,i), from a person q. Clearly, since e_i is above q, and p is above e_{i+1}, no path in D can contain both R and R′. Further, R^{d} (if it is a rotation) cannot precede R′ on any path, since R^{d} moves the head of s_i's list to e_{i+1}, which is below p. Similarly, R′ cannot precede R^{d} on any path, since it moves s_i's bottom to p, which is above e_{i+1}. But every path contains either R or R^{d}, so no path contains R′, contradicting the definition of a rotation. The proof for moves from B(R,i) is similar. □

Corollary 5.1: A person p is the H element a person q's list in at most one rotation, and is the S element of q's list in at most one rotation. Hence there is at most one rotation whose elimination moves the head of q's list to p.

Corollary 5.2: If $p \neq e_i$, and $p \in A(R,i)$, then s_i can never be paired with p in any stable roommate assignment.

Proof: Consider any path P where p is paired with s_i. Since s_i prefers p to e_i, p is not the bottom of s_i's list in the phase I table. Hence, somewhere on P, p must become the bottom of s_i's list. But this contradicts lemma 5.1 above. □

5.1.1. Necessary elimination

Let R be a rotation with the triple e_i, h_i, s_i in R. If $p \neq h_i$, and p is above s_i in e_i's list, then R will never be exposed until p is removed from e_i's list.

Lemma 5.2: Let p be a person who must be removed from e_i's list before R is exposed. There exists a unique rotation R′, such that R′ appears before R on every path that contains R, and such that of all the rotations which appear before R on any path in D, R′ is the only one whose elimination removes p from e_i's list.

Proof: From examination of algorithm I, there are only two ways in which p is removed from e_i's list: either p is removed when the bottom of e_i's list moves up above p, or when the bottom of p's list moves up above e_i. If p is removed by the first case event, then R cannot be embedded in the table after e_i's bottom moves above p, since p is above s_i. Hence p is removed by the first case event only on paths that don't contain R. By lemma 5.1, the second case can happen only when a particular unique rotation, R′, is eliminated. Since p most be removed from e_i's list before R can be exposed, R′ must precede R on any path that contains R. □

Definition: If p must be removed from e_i's list before R is exposed, and if R′ is the (unique) rotation discussed in lemma 5.2, then we say that R′ *explicitly precedes* R.

5.2. The partial order Π^* and the structure of the stable assignments

Definition: Let Π^* be the reflexive transitive closure of the above relation of explicit precedence. It is clear that Π^* is a partial order on the rotations.

Figure 4 shows the Hasse diagram of Π^* for the running example.

Definition: In partial order Π^*, a subset C of rotations is called *closed* if and only if it is closed under the predecessor relation, i.e. if R is in C, and R′ precedes R in Π^*, then R′ is in C.

Lemma 5.3: There is a one-one correspondence between the path sets in D and the set of those closed subsets of Π^* which contain all the singleton rotations, and contain exactly one of each dual pair of rotations.

Proof: One direction is trivial. Let C be a path set in D, and let P be any of the paths in D containing path set C. We claim that C forms a closed subset in Π^* of the required type. We know that each path in D contains all the singleton rotations and exactly one of each dual pair of rotations, hence we only need to show that C is closed in Π^*. But, by lemma 5.3 above, any rotation that precedes $R \in C$ must be contained on P, hence C is closed. Conversely, let C be a closed set of the required type; we will show that there is a path P in D which contains C exactly. First, the maximal elements $C_0 \subseteq C$ (those with no predecessors in Π^*) must be exposed rotations in the phase I table, and since only one of any dual pair is in C, there is a subpath from

the root of D consisting of the rotations C_0. After the C_0 rotations are eliminated, the elements $C_1 \subset C$ whose only predecessors are in C_0 must now be exposed, and (since only one of each dual pair is in C) each remains exposed until eliminated, and hence there is a path from the root consisting of C_0 followed by C_1. Continuing in this way, there is a path from the root to a leaf in D consisting of the rotations in C. □

The proof of the following is essentially the same as the second part of the above proof of Lemma 5.3.

Corollary 5.3: If C is a closed set in Π^*, and is contained in some path P in D, then there exists a path P′ in D containing the same rotations as P, but where all rotation in C appear before any rotations not in C. Further, the internal order of the rotations in C is the same on both paths, as is the internal order of the rotations not in C, i.e. in P′ the rotations in C simply move above the non-C rotations, but keep their same internal order.

Since each path set maps to a unique stable assignment, Lemma 5.3 implies that each closed subset in Π^* of the required type maps to a unique stable assignment. We now show that distinct closed subsets map to distinct stable assignments.

Lemma 5.4: Let C and C′ be two distinct closed subsets of Π^* which both contain all the singletons and exactly one of each dual pair of rotations. Then, as path sets, C and C′ yield distinct stable assignments.

Proof: Suppose to the contrary that both the path sets C and C′ produce the same stable assignment *A*. We claim that the minimal rotations in C must be in C′. Suppose not, and let R = (E,H,S) be a minimal rotation in C, and let e_i be in E. Then in *A*, e must be mated to s_i for every rotation that moves e_i below s_i (and moves s_i above e_i) must be preceded by R. But R is minimal in C, so e_i is mated to s_i in *A*. But by Corollary 5.1, there is only one rotation that moves the head of e_i to s_i. Hence R, and all minimal rotations in C, must also be in C′. But since C and C′ are closed, all the predecessors of the minimal rotations in C must also be in C, and hence in C′. But any closed subset is precisely the set of its minimal elements, plus the predecessors of those minimal elements. Hence $C \subseteq C'$, but since they both have the same cardinality, $C = C'$. □

The following two theorems connect the preceding lemmas and summarize what we now know about the structure of the stable assignments.

Theorem 5.1: There is a one-one correspondence between stable assignments and those closed sets in Π^* which contain every singleton rotation and exactly one of each dual pair.

Hence Π^* is a small, $O(n^2)$ nodes, representation of the set of all stable assignments. We will later discuss how to efficiently construct Π^*, and how to construct a stable assignment from a

closed subset of the correct type.

Theorem 5.2: There is a one-one correspondence between path sets in D and stable assignments.

Theorem 5.2 is Lemma 3.4 and its converse, and is one of the keys to efficient enumeration of the stable assignments. Before discussing enumeration, we derive an alternative representation for the stable assignments.

5.3. Independent set representation of stable assignments

In this subsection we present a second compact representation of the set of all stable assignments. We first need some additional observations.

5.3.1. Refining Π^*

Lemma 5.5: Let (R, R^d) and (R_1, R_1^d) be two dual pairs of rotations, and R' a singleton rotation. Then:

1) Neither R nor R^d can precede R' in Π^*, i.e. only a singleton rotation can precede a singleton.

2) R precedes R_1 in Π^* if and only if R_1^d precedes R^d in Π^*.

Proof: Since each singleton rotation is on every path in D, any rotation which precedes a singleton rotation must be on every path in D. So if R precedes R', R is on every path, and R^d is on no paths in D, contradicting the assumption that R^d is a rotation; so the first fact is proved. For the second fact, observe first that since R precedes R_1, no path can contain both R_1 and R^d, so any path containing R^d contains R_1^d. We must show that in any such path, R_1^d appears before R^d. Let P be a path containing R^d, and consider the point x where R^d is eliminated. Since no path can contain both R_1 and R^d, R_1 cannot be exposed in D(x). But Corollary 6.2, which will be proven later, states that if any non-singleton rotation is embedded in T(x), then both it and its dual are exposed somewhere in D(x), the subtree of D below x. Hence it must be that R_1^d is not embedded in T(x), so R_1^d must appear on P before R^d, and fact 2 is proved. □

5.3.2. Graph G

We define an undirected graph G as follows. There exists one node in G for each non-singleton rotation, and two rotations R_1 and R_2 are connected by an edge in G if and only if there exists a rotation R (possibly R_1 or R_2) such that in Π^*, R precedes R_1 and R^d precedes R_2. It follows that R and R^d are connected for each dual pair (R,R^d) and if two rotations are adjacent in G, then they cannot appear together on any path in D.

Lemma 5.6: Every maximal independent set in G contains exactly one node from each dual

pair of rotations.

Proof: First, no independent set can contain both nodes of a dual pair. For the other side, let S be an independent set in G which does not contain either R or R^d. If neither R nor R^d can be added to S, then there must be a node R_1 in S such that (R,R_1) is an edge in G, and there must be a node R_2 in S such that (R^d,R_2) is an edge in G. But then there exists a rotation R_3 such that R_3 precedes R, and R_3^d precedes R_1, so R_1^d precedes R_3 (by Lemma 5.5 part 2), and it follows that R_1^d precedes R. Also, there exists a rotation R_4 such that R_4 precedes R^d and R_4^d precedes R_2, so R precedes R_4^d precedes R_2. So R_1^d precedes R precedes R_2. But then R_1 and R_2 would be connected in G. Hence it can't happen that neither R nor R^d could be added to S to create a larger independent set, and continuing in this way, the lemma is proved. □

Definition: Let Π be the partial order Π^* with the singletons removed.

The following is directly implied by Theorem 5.1 and Lemma 5.5 part 1.

Lemma 5.7: Let Σ be the set of all singleton rotations in Π^*. A set of rotations C is closed in Π^* if and only if C - Σ is closed in Π. Hence there is a one-one correspondence between the stable assignments and the closed subsets of Π that contain exactly one of each dual pair.

We can now show a one-one correspondence between the maximal independent sets in G and the stable assignments.

Lemma 5.8: Any closed subset C in Π that contains exactly one of each dual pair, is a maximal independent set in G.

Proof: No path in D can contain two rotations which are connected in G. But since C is closed in Π, C $\cup$ Σ corresponds to a path set from D, hence to at least one path in D. Hence no rotations in C can be connected in G, and C is an independent set in G. It is maximal, because it has one rotation from each dual pair, and no independent set in G has more. □

Lemma 5.9: Any maximal independent set in G is a closed set in Π.

Proof: Let S be a maximal independent set in G, and let R be any rotation in S. Suppose R' precedes R in Π. Then, since R'^d precedes itself, R and R'^d are connected in G, so R'^d is not in S. But S must contain one of each dual pair, so S contains R'. It follows that S is closed in Π. □

In summary,

Theorem 5.2: There is a one-one correspondence between the maximal independent sets in G and the set of stable assignments.

Hence, G is another small implicit representation of the set of all stable assignments. We can simplify the definition of G with the following

Lemma 5.10: Rotations R and R′ are connected in G if and only if R^d precedes R′ in Π.

Proof: First, if R^d precedes R′, R and R′ are connected in G, since R precedes R′. Conversely, if R and R′ are connected in G, then there is a rotation R^* such that R^* precedes R and R^{*d} precedes R′. But by Lemma 5.5, R^* precedes R implies that R^d precedes R^{*d} so R^d precedes R′. □

The above lemma allows a more efficient method to construct G: we can determine whether to connect two nodes by looking directly at a single entry in the precedence relation, rather than looking for a third rotation to satisfy the initial requirement for connecting two nodes in G. It turns out that G can be constructed in time $O(n^4)$; details are omitted, but are very similar to the construction of a related partial order Π given in [G].

5.4. Equivalent sets and representations

We now know that the path sets of D, the closed subsets of Π^* that contain each singleton and exactly one of each dual pair, and the maximal independent sets of G augmented with the singletons, are all exactly the same sets of rotations, and these sets are in one-one correspondence with the stable assignments. We will see later that each such set can be used to generate the associated stable assignment in $O(n^2)$ time.

6. Efficient enumeration of all stable assignments

We now discuss an efficient method to generate all stable assignments. The method is a modification of a very general, naive method that is often suggested as a way to enumerate constrained sets, but in general, this naive method is rarely efficient. The efficiency here is a consequence of very special properties of the stable assignment problem, and the way the enumeration is implemented.

By Theorem 2.1, the stable assignments can be generated by forcing all execution behaviors of algorithm I. This would not be guaranteed to be efficient, since the same assignment would likely be generated many times. However, by Theorem 5.2, we need not generate each path in D, but only each path *set*. One approach is to use graph G to generate each maximal independent set (path set) in G, and, as we will show below, then use the path set to generate the associated stable assignment. The fastest known methods to generate all maximal independent sets in a general graph appear in [LLR], but applying those methods to G yields a time bound of $O(n^4)$ per independent set. Alternatively, we could try to generate all the closed subset of Π^* which contain all singletons and exactly one of each dual pair. However, general techniques for such constrained enumeration in general partial orders seem very inefficient. Here, we will use a different approach, heavily exploiting results about the structure of D and the rotations, to obtain a method which, after Π^* is constructed, will generate each path set, and each stable assignment,

in $O(n^2)$ time per assignment. This compares favorably with the $O(n^3)$ methods given in [K] and [MW] to construct each stable *marriage*[1]. In the first subsection we will present the method and prove it correct, and in the second subsection we will discuss the time it requires. We first need the following definitions.

Definition: Let SR be a set of rotations. For each person p, let SR(p) be the highest person on p's list such that $SR(p) = e_i$ and $p = s_i$ in some rotation $R = (E,H,S)$ in SR.

Definition: Let T be a table and SR be a set of rotations. The *elimination of SR from T* is the following: For each person p, delete all people in p's list below SR(p), if there are any, and delete p from the lists of all of those deleted people, if p exists in those lists.

Note that the definition does not require that all the rotations in SR be embedded in T.

Definition: For a rotation R, define $\Pi^*(R)$ as the set of rotations consisting of R and all the predecessors of R in Π^*.

6.1. The dual enumeration method

The idea of the method is to simulate algorithm I forcing it to generate each path set, and associated stable assignment, exactly once. We will represent the simulation by a binary tree B. As in tree D, each node x in B will represent a table; when node x is a non-leaf, one edge out of x will be labelled by a single rotation which is exposed in T(x), but the other edge, if it exists, will be labelled by a set of rotations, not necessarily embedded in T(x). We will call the first edge the *left* edge and the second edge the *right* edge. If the left edge (x,y) in B is labelled by R, then the table T(y) at node y is obtained from T(x) by eliminating R from T(x). When R is a singleton, then there will be no right edge out of x, but when R is a non-singleton, then the right edge (x,z) will exist and will be labelled with $\Pi^*(R^d)$, and the resulting table T(z) will be the table obtained by the elimination of the set $\Pi^*(R^d)$ from T(x). Each leaf of B will contain a table with no exposed rotations. The simulation begins with a node representing the phase one table, and each node x in B is expanded by arbitrarily choosing an exposed rotation R in T(x), eliminating R from T(x) on the left edge out of x, and, if R is a non-singleton, eliminating $\Pi^*(R^d)$ from T(x) on the right edge out of x. A table with no exposed rotations is not expandable; we will show that such a table must specify a stable assignment. The simulation ends when no unexpanded nodes in B are expandable. We call this method the *dual enumeration* method. Figure 5 illustrates this method on the running example.

[1]It is reported in [K] that the time is $O(n^2)$, per marriage, but this is incorrect. Constructions appear in the appendix of [G1] showing that the algorithm can take $\Omega(n^3k/\lfloor\log k^2\rfloor)$ time for k stable marriages.

6.1.1. Correctness of the dual enumeration method

Lemma 6.1: The dual enumeration method never generates a path set (stable assignment) more than once.

Proof: Let x be a non-leaf node in B, and let R be the rotation on the left edge out of x. Consider the leaves in the subtree rooted at x, and the path sets associated with those leaves. Those path set are divided into those sets containing R and those containing R^d (none, if R is a singleton). Since no path set contains both a rotation and its dual, there is no intersection between these two set of path sets. Applying this fact inductively upward from the leaves to the root in B, it follows that no path set is generated more than once. Theorem 5.2 then implies that no stable assignment is generated more than once. □

Lemma 6.2: Every stable assignment (path set) is generated at least once by the above method.

Proof: Let x be a non-leaf node in B which is also a node in D (the root of B is such a node, although we will later see that all nodes of B are in D). Let A be a stable assignment contained in T(x). Let R be the exposed rotation eliminated from T(x) on the left edge out of x in B, and let T(y) be the resulting table. Suppose first that the path set for A contains R. Then by Corollary 2.2, and the fact that T(x) is in D as well as in B, it follows that node y is in both D and B, and assignment A is contained in table T(y).

Now suppose that the path set for A contains R^d, and let w be any node in D(x), the subtree of x in D, such that T(w) contains A and such that the edge into w is labelled with R^d. Clearly, every path from x which leads to a leaf labelled with A contains such a node w. Also, every rotation in $\Pi^*(R^d)$ must be on the path in D from the root to w. Now, as in the proof of Corollary 5.3, a rotation R^d is exposed if all of its predecessors in Π^* have been eliminated and R has not been removed, so there must also be a path in D from x to a node z, containing exactly those rotations in $\Pi^*(R^d)$ that are on the path from x to w (i.e. this second path is obtained by deleting all the rotations on the first path from x that are not in $\Pi^*(R^d)$). Further, the elements of T(w) are all contained in T(z), since the rotations leading to w are a superset of those leading to z. Hence A must be contained in T(z) as well as in T(w). But T(z) is exactly the table obtained in B by the elimination of $\Pi^*(R^d)$ from T(x). Hence, if the path set for A contains R^d, then the right child of node x in B is the above node z in D, and T(z) in B contains assignment A.

Now the phase one table is in both D and B, and so by iterating the above arguments downward from the root, we see that there is a path in B consisting of nodes in both D and B, such that assignment A is contained in each of the tables on the nodes in the path. Since the nodes on this path are in D, and the table sizes decrease with each edge on the path, the path

ends with a table containing assignment A exactly. Hence assignment A appears at some leaf of tree B. □

The above lemmas show that every stable assignment is generated exactly once, but this does not prove the correctness of the dual method, since we have not proved that the method never generates tables that are not in D. If it did, then it could generate assignments that are not stable, or it could generate tables that have "rotations" that are not in D. Lemma 6.2 does not show that every table in B is also in D, although it is immediate that if x is in B and D, then y, the left child of x in B, is also in D. What remains to show is that the right child of x in B is necessarily in D. The point is that even though there is an assignment in T(x) whose path set contains the non-singleton rotation R, which is exposed in T(x), we don't know for sure that there is an assignment in T(x) whose path set contains R^d. If there is no such assignment, then the dual enumeration method will either generate a table which doesn't specify an assignment but which has no exposed rotations (in which case it will have done excess work), or it will generate an assignment which is not stable. Neither of these things can happen if each table in B is also in D. To prove that each table of B is in D, we show that for any node x in D, if R is exposed in T(x), then R^d is exposed in some table in D(x), the subtree of D rooted at x. This implies that T(x) contains a stable assignment whose path set contains R^d in D(x), if R is exposed in T(x). Of course, since R is exposed in T(x), R^d appears below x on any of these paths.

Theorem 6.1: Let R and R^d be dual rotations, and x a point in D. If R is exposed in T(x), then R^d is exposed in D(x). Hence if R is exposed in T(x), then there is a stable assignment in T(x) whose path set contains R^d as well as one which contains R.

To prove this Theorem we first need the following

Lemma 6.3: If P is a path from the root of D to a node x in D, and P′ is a path from the root to a node x′, and P and P′ *cover* the same set of rotations, then the *active parts* of table T(x) and table T(x′) are identical, and hence the active parts of D(x) and D(x′) are also identical.

Proof: Note first that since no path can contain both rotations in a dual pair, the length of P and P′ are the same. Let dP and dP′ be the parts of P and P′ respectively, after the point v in D where P and P′ diverge. The proof of the lemma is by induction of the length of dP (which is, of course, also the length of dP′). For length of one, dP must contain R while dP′ contains R^d, for some dual pair of rotations. Then in T(x), both R and R^d are exposed and the basis follows from lemma 3.2. Now assuming the theorem holds for dP of length k, consider dP of length k+1, and let R and R′ be the first rotations on dP and dP′ respectively, and let v_R and $v_{R'}$ be the first nodes below v on these paths (see figure 6a). If $R' = R^d$ then the active tables are the same after eliminating either rotation, and hence there must be a path from v_R that is identical to the part of dP′ starting at $v_{R'}$. Hence the table T at the end of that path is T(x′). But, by

the induction hypothesis, the active part of T(x) is the same as the active part of T, hence the theorem follows in this case.

Now suppose that $R' \neq R^d$. There are two cases to consider: either R is on dP′, or R^d is on dP′.

Let w be the point on dP′ where R (in the first case) or R^d (in the second case) is eliminated. Since both R and R′ are exposed at v, R must be exposed at every table on dP′ down to w.

In the first case, consider the edge on dP′ into w, and let R^* be the rotation eliminated there (see figure 6b). If instead of eliminating R^*, R is eliminated at that point, R^* will not be removed (since $R^* \neq R^d$), hence the path which is identical to dP′ except that the order of R^* and R is reversed, is in fact a path from v; call that path dP*. Now dP′ and dP* contain exactly the same rotations, so the table at the end of dP* is T(x′). Repeating this argument up the length of dP′, moving R up at each step and leaving the rest of the path the same, it follows that there is a path from v through v_R which contains the same rotations as dP′, and hence ends with table T = T(x′). But, by the inductive hypothesis, as above, the active parts of T(x) are identical to the active parts of T, and hence of T(x′).

In the second case, there must be a path, P(R), from v through v_R, which is identical to dP′, except that R replaces R^d (see figure 6c). This follows from lemma 3.2, and the fact that R is exposed at w. But now dP′ and P(R) diverge below v, hence by the induction hypothesis, the active part of the table, T, at the end of P(R) is identical to the active part of T(x′). Now we can repeat the step argument of the first case, moving R up P(R) to v, and conclude that there is a path from v through v_R which contains exactly the same rotations as P(R). Then, by the inductive hypothesis, the active part of T(x) is the same as the active part of T, which is the same as the active part of T(x′). □

Proof of theorem 6.1: Let z be the closest ancestor of x such that R^d is exposed in D(z), and let y (possibly x) be the child of z on the path from z to x; Let R_1 be the rotation on the (z,y) edge. Let P be a path from z to a leaf, where P contains R^d, and let R_2 be the first rotation on P (see figure 7a). If $R_1 = R_2{}^d$, then the active tables after eliminating either rotation are the same, so the subtrees below those two points must be the same, hence D(y) must contain R^d. So, assume that $R_1 \neq R_2{}^d$. Neither R_1 nor $R_1{}^d$ is on the path from the root to z, so either R_1 or $R_1{}^d$ must be on P, say at a point w. Note that in either case, R_1 is exposed at w, as in the above proof of lemma 4.2. Hence if R^d is before w on P, then we can assume P contains R_1. If P contains $R_1{}^d$ and R^d is after w (see figure 7b), then consider the affect of eliminating R_1 at w; the resulting active table is the same as after eliminating $R_1{}^d$, so there is a path from w which contains R^d. So we can always assume that P contains R_1 and R^d. But now, we can move R_1 up a step at a time, as the preceding proof, concluding that there is a path from z through y that

contains R^d. This contradicts the selection of z, and proves the theorem. □

Theorem 6.1 completes the proof of correctness of the dual enumeration method. In summary,

Theorem 6.2: The dual enumeration method generates each stable assignment exactly once, and the table at each leaf in B specifies a stable assignment.

There are several useful corollaries of theorem 6.1.

Corollary 6.1: For any table T(x) in D, if R is the only rotation exposed in T(x), then R is a singleton.

Corollary 6.2: If R and R^d are duals embedded in T(x), then both are exposed somewhere in D(x).

Recall that the proof of Lemma 5.5 part 2 depended on Corollary 6.2, so Lemma 5.5 is now fully proved.

Corollary 6.3: If R and R^d are duals, then there is a point x in D where both R and R^d are exposed in the table T(x).

Corollary 6.4: If R is a non-singleton rotation, and $e_i \in R$, then there is a stable roommate assignment where each e_i is paired with h_i, and also one where each e_i is paired with s_i, where e_i is in R.

Proof: At point x where both R and R^d are exposed, eliminating R^d makes h_i the only element on e_i's list, and eliminating R makes s_i the only element on e_i's list. With either elimination, the algorithm is guaranteed to find a stable assignment in the resulting table. □

6.2. Complexity Analysis

In this subsection we show that the dual enumeration method works in time $O(n^3 \log n + kn^2)$ to enumerate k stable assignments. The $O(n^3 \log n)$ term is the time needed to find all the rotations in D and to construct Π^*. Thereafter, each stable assignment is generated in $O(n^2)$ time, per assignment. To prove the above bound, we first examine the computational steps needed before and in the dual method: how to find all the rotations and recognize the singletons, how to eliminate a set of rotations, how to generate "enough" of Π^*, and how to find $\Pi^*(R)$ for a rotation R. We then show how to charge the work during the dual algorithm, and prove the above time bound.

6.2.1. Finding all the rotations in $O(n^3 \log n)$ time

Even though the number of stable assignments, hence the size of D, grows exponentially in n, the Path Theorem, and the fact that algorithm I runs in $O(n^2)$ time, imply that there can be at most $O(n^2)$ rotations in D. In fact, Corollary 5.1 shows that there can be at most n(n-1) rotations. We show here that all the rotations in D can be found in $O(n^3 \log n)$ time. To do this, we run algorithm I once, following a path P in D, finding the rotations on P. By the Path Theorem, P covers all the rotations on D, but we still have to determine which are singletons and which have duals. If R is a rotation on P, then we can test if R^d is a rotation by simply returning to the point on P where R is eliminated and successively choosing and eliminating any rotation other than R. By Corollary 4.2, we will either expose and eliminate R^d, or we will have a table where only R is exposed; in the latter case, R must be a singleton, by Corollary 6.1. There are at most n(n-1) rotations on P, and each run of algorithm I costs $O(n^2)$ time, so although the size of D can be exponential in n, all rotations in D can be found in $O(n^4)$ time. Of course, in practice this procedure can be sped up by noting at each step which other rotations are exposed.

We can speed up the above method to run in time $O(n^3 \log n)$. The idea is that we can find 2n chains in Π^* that contain all the rotations on path P; there is one chain for each person p, and it simply consists of the ordered list of the rotations on P that move the head of p's list. The rotations in the chain are ordered by the relative order that they appear on P. It is easy to see that if rotation R moves p's head before rotation R′ does, on P, then R precedes R′ in the partial order Π^*. Now Lemma 5.5 part 1 says that in Π^* only singleton rotations can precede a singleton, so this implies that there is a point on each chain where all the rotations above the point are singletons, and all below it are non-singletons; we search for that point using binary search. Each query in the search costs $O(n^2)$ time, as above, and since there are only n(n-1) rotations per chain, the breakpoint for each chain is found in $O(n^2 \log n)$ time. There are only 2n chains, so at most $O(n^3 \log n)$ total time is required. Since the form of the dual rotations (not on P) are known, and the total size of their description is $O(n^2)$, once the singletons have been identified, the rotations not on P can be generated in $O(n^2)$ total time.

6.2.2. Time needed for the elimination of a set of rotations

Any set of rotations SR can be eliminated from any table T in $O(n^2)$ time. The key is that the total size of the description of SR is $O(n^2)$, so a simple scan through the rotations in SR finds SR(p) for each person p, and this takes $O(n^2)$ time in total. Further, any table is of size $O(n^2)$, and when SR is eliminated, each element to be removed can be found and removed in constant time, so the elimination of the elements only requires $O(n^2)$ time. This also shows that given a path set, the associated stable assignment can be generated in $O(n^2)$ time, by simply eliminating the path set from the phase one table.

6.2.3. Time needed to "construct" Π^* and to find $\Pi^*(R)$

The partial order Π^* has $O(n^2)$ elements, and if we represent Π^* as a directed graph, DG, where each node is an element in Π^* and each edge corresponds to a pair in the relation, then there might be as many as $\Omega(n^4)$ edges in the graph. However, in the dual enumeration method, we only need to know Π^* in order to find the predecessors of a rotation. Hence it will suffice to know any subgraph of DG whose transitive closure is Π^*. By definition, the Hasse diagram of Π^* is the smallest such subgraph. It turns out that the Hasse diagram has at most $O(n^2)$ edges, and there is a supergraph, DG*, of the Hasse diagram, which also contains only $O(n^2)$ edges, and DG* can be found from the set of rotations in $O(n^2)$ time. Then since DG* has only $O(n^2)$ nodes and edges, given any rotation R, we can find $\Pi^*(R)$ in $O(n^2)$ time by backwards search from R in the obvious way.

Summarizing, we have

Lemma 6.4: There exists a directed acyclic subgraph, DG*, of DG containing all the nodes of DG, such that R leads to R′ by a path in DG* if and only if R is connected by a directed edge to R′ in DG. Further, DG* has $O(n^2)$ edges, and it can be constructed from the set of rotations in $O(n^2)$ time.

Proof: Let e_i, h_i, s_i be a triple in rotation R. Recall that when R is eliminated from any table it is exposed in, the bottom of s_i's list moves from e_{i+1} to e_i, where i+1 is taken mod |R|. Recall also that A(R,i) is the set of people on s_i's original list between e_i and e_{i+1}, including e_i but excluding e_{i+1}. In order to "construct" Π^*, we first examine each rotation R = (E,H,S), and for each person s_i in S, we find A(R,i) and then for each person p in A(R,i), we mark s_i in p's list with rotation R. All of these markings can be done in $O(n^2)$ time, since each A(R,i) is a contiguous list of elements in s_i's original list, and by Lemma 5.1, no two A(R,i) sets intersect.

The rest of the construction method, and the proof of its correctness and time, essentially appear in [G] section 4.2 where enumeration for the stable *marriage* problem is discussed. In the stable marriage problem, the set of stable marriages are also represented by a supergraph of a the Hasse diagram of a particular partial order. That supergraph has $O(n^2)$ nodes and edges and is constructed from a marked table similar to the one above. What is important is that the only properties of the marked table that are needed in the construction of the supergraph of [G], and to prove the size and time bounds, also hold for the above marked table used here. The reader is refered to [G] for the complete details. □

6.2.4. Time needed for the dual enumeration method

Theorem 6.3: Given DG* as above, each stable assignment can be generated in $O(n^2)$ time, per assignment. In fact, the assignments can be generated on-line in this time.

Proof: Consider a node x in B, and define x as a left node if it is the root of B, or if the edge

into it from its parent is a left edge. From each leaf which is also a left node there is a unique maximal path upward from the leaf consisting only of left nodes (see figure 8). For example, the path from the leftmost leaf in B runs to the root of B. These paths are edge disjoint, and cover every left edge in B. Now consider the top node x of one of these paths. Starting from this top node x, the work of the dual enumeration method along the path down to the associated leaf, consists of an execution of algorithm I starting from T(x), which is a subtable of the phase one table. Hence the total time for the work along this path is $O(n^2)$. But these paths are disjoint, each ends at a distinct leaf of B, hence at a distinct stable assignment, and these paths cover all the left edges in B. Hence the total work of the dual enumeration method along the left edges of B is $O(n^2)$, per assignment.

The work on any right edge out of any node x in B consists of finding $\Pi^*(R^d)$ for a given R^d, and eliminating $\Pi^*(R^d)$ from T(x). As shown above, each of these operations can be done in $O(n^2)$ time. If (x,y) is a right edge, and y is a leaf, then we can charge the $O(n^2)$ time for edge (x,y) to the assignment at node y. If (x,y) is a right edge and y is not a leaf, then y is the top node of one of the maximal paths discussed above. We charge the work on (x,y) to the assignment at the end of that path. Clearly, no assignment gets charged twice for a right edge, so the work for all the right edges in B is also $O(n^2)$, per assignment.

In order to make the time bound on-line, simply expand B in a depth first manner, going left whenever possible. In such a traversal, no more than $O(n^2)$ time can pass between the generation of new stable assignments. □

In the above analysis, the time for the left edges can be accounted for more closely, to get an O(n) time bound per assignment for the left edges. However, the work for the right edges remains $O(n^2)$ per assignment, and it is an open question whether this can be substantially reduced.

7. Characterizing the Stable Pairs

Definition: If persons i and j are paired together in some stable assignment, then they are called a *stable pair*. If they are paired together in all assignments, then they are called a *fixed pair*.

Knuth mentions the utility of knowing the stable pairs in the stable marriage problem, and in [G] it is shown how to find all the stable pairs in $O(n^2)$ time. Here we examine the equivalent question for the stable roommate problem.

Definition: Let Σ be the set of singleton rotations, and let P be a subpath from the root in D containing all and only the singleton rotations. Let x^* be the end of path P.

By lemmas 5.3 and 5.5, it is clear that such a P exists, and that all stable assignments are in

$D(x^*)$. In other words, if all the singleton rotations are eliminated as a set from the phase one table, $T(x^*)$ results, and it contains all stable assignments.

Lemma 7.1: Person i is in a fixed pair if and only if i's list in $T(x^*)$ has only a single entry.

Proof: First, since $D(x^*)$ contains all the stable assignments, if i's list in $T(x^*)$ contains only one entry, j, then (i,j) is a pair in every stable assignment. If i's list in $T(x^*)$ is not a single entry, then i must be in the E set of a rotation R, and in the H = S sets of rotation R^d. Hence by corollary 6.4, i is in at least two stable pairs, and so is not in a fixed pair. □

Given the working of algorithm I, and the fact that $T(x^*)$ is generated by the singletons, it is clear that if i is not in a fixed pair, then i can mate with person p only if p is in the S set for i in some non-singleton rotation R, if and only if i is in the S set for p in R^d. This observation combined with Corollary 6.4 gives

Theorem 7.1: If (e_i,j) is not a fixed pair, then it is a stable pair if and only if $j = h_i$ in some non-singleton rotation R = (E,H,S).

Hence the non-fixed stable pairs can be found immediately from the non-singleton rotations, and the fixed pairs can be found in $O(n^2)$ time from the singletons, by eliminating the singletons from the phase 1 table. Hence the set of all stable pairs can be found in $O(n^3 \log n)$, and any speed up, down to $O(n^2)$ time, in finding the rotations will speed up finding the stable pairs. It is not difficult, using Theorem 7.1, to find all the fixed pairs in $O(n^3)$ time, but it is open whether this leads to a faster way to find all the stable pairs, or to find the singletons.

8. Specializing the roommate structure to marriage

Definition: Let MP be an instance of the stable marriage problem on n men M, and n women W. Let RMP be an instance of the stable roommate problem on $M \cup W$, obtained from MP by adding to the end of each man (woman) p's list all the men (women) other than p.

It is easy to prove that each stable assignment in RMP is a stable marriage in MP and conversely. Hence the structure of the stable roommate problem applies to the marriage problem, and it is of interest to see how the general structure specializes in the case of stable marriage. In particular, it is interesting to compare the results here to those of [IL], where a structure of the set of stable marriages was first obtained. The results in this paper specialize to those in [IL] for the marriage problem. However, the structure of stable marriages is somewhat simpler than the general roommate structure, and this allows faster algorithms to construct it, and a simpler view of how to construct stable marriages from the partial order(s). Hence the structure resulting from specializing the roommate structure to stable marriage at first appears different than that in [IL]. In the next paragraphs we sketch additional observations that connect the structures and the expositions.

The following facts are stated without proof:

1. In Π^* resulting from RMP, there are no singleton rotations.

2. If R = (E,H,S) is a rotation in Π^* resulting from RMP, then all people in E have the same sex, and all people in H have the opposite sex. Hence the E set in R is male if and only if the E set in R^d is female, and the rotations partition naturaly into two equal sized sets of rotations, one where each rotation has a male E set and one where each rotation has a female E set. We call the first type the male rotations, and the second type the female rotations.

3. In every pair in relation Π^* resulting from RMP, the two rotations have the same sex. Hence, the partial order Π^* in RMP is composed of two disjoint partial orders: MΠ^* containing the male rotations, and FΠ^* containing the female rotations.

4. Let C be a closed set in MΠ^*, and let C′ be the rotations in MΠ^* - C, and let C'^d be the duals, in FΠ^*, of the rotations in C′. If C is closed in MΠ^*, then C'^d is closed in FΠ^*, and hence their union is closed in Π^*, and since the union contains exactly one of each dual pair, it represents a stable assignment in RMP (stable marriage in MP). The converse is obviously also true. Hence, there is a one-one correspondance between the closed sets in MΠ^* (without further constraints) and the stable marriages of MP.

This paper introduced the concepts of dual rotation and singleton rotation. Facts 1 and 4 explain why these concepts where not needed in the structure of stable marriage. Although the exposition is quite different here, the representation of the set of stable marriages given in [IL] can be expressed as MΠ^*. Facts 3 and 4 help "explain" why the representation of stable marriages is simpler than the representation of stable assignments: each closed subset in MΠ^* represents a stable marriage, and conversely, while in the general case of stable assignment, only particular, highly constrained closed subsets in Π^* represent stable assignments. This also partly "explains" why each stable marriage can be constructed in O(n) time in [G], while in this paper, $O(n^2)$ time for each stable assignment is the best bound obtained. Hence what makes the roommate problem more involved is the existance of singleton rotations, and the fact that the non-singletons don't partition in a nice way, as they do in the marriage problem. As a final comment, note that given fact 1, all rotations in MΠ^* can be obtained from a single pass through algorithm I in $O(n^2)$. This bound was first obtained for the stable marriage problem by a very different method and argument in [G].

9. Open Problems

First, given the O(n) time method in [G] to enumerate each stable marriage, compared to the $O(n^2)$ method for the more general problem of stable assignment, a natural problem is to bring down the time for enumerating assignments, or to more fully explain why the stable assignment

problem is more complex than the marriage problem. Second, the $O(n^3 \log n)$ bound for finding all the rotations, and hence the stable pairs, seems too large. We conjecture that $O(n^2)$ is the correct bound, and that the singletons should be recognizable after a single execution of algorithm I. It seems likely that there are additional structural observations about rotations that will lead to this time bound. For example, it is an easy corollary of Theorem 6.1 that if R = (E,H,S) is a rotation such that $E \cap H \neq \emptyset$, then R is a singleton rotation. In the example presented in this paper, that fact identifies the singletons, but it is not true that for every singleton rotation $E \cap H \neq \emptyset$. Finaly, there is the more general question of what algebraic structure is generated by the set of stable assignments, under some reasonable relation. It is known that the stable marriages generate a distributive lattice under the relation of "domination" [K], and as pointed out in [GILS], this is the essential key to the efficient representation of the set of stable marriages. Further, there is a good sized class of interesting combinatorial problems each of whose solution sets generate a distributive lattice over some natural relation. It then follows (see [IR] or [N]) that for each of these problems, the solution sets can be represented by a compact partial order where the solutions are in one-one correspondence with the closed subsets of the partial order, although the time needed to find the partial order and to extract the solutions differs in each case. This partial order representation has many algorithmic uses; applications in stable marriage appear in [IL], [ILG], and [GILS], but there are many other applications in other combinatorial problems (see [IR] for a good bibliography). The stable assignments are not known to be representable as closed subsets in a partial order, but their representation in Π (closed subsets which contain exactly one of each dual pair) is certainly related, and it raises the question of whether an algebraic study of this structure would be fruitful. In particular: What sort of algebraic structure do these closed subsets of Π generate, and is there a natural relation that ties them together? Is there an interesting general class of problems with this structure? Can such problems be reduced to problems of the first (seemingly simpler) type? Does this more general algebraic approach lead to more efficient or generalizable algorithms?

10. References

[GS] D. Gale, and L. Shapley. College Admissions and the Stability of Marriage. Am. Math. Monthly 69 (1962).

[GS84] D. Gale, and M. Sotomayor. Some Remarks on the Stable Matching Problem. Discrete Applied Math (to appear).

[G] D. Gusfield, Three Fast Algorithms for Four Problems in Stable Marriage. SIAM Journal on Computing, to appear.

[G1] D. Gusfield, Three Fast Algorithms for Four Problems in Stable Marriage. Department of

Computer Science, Yale University Technical Report #TR-407, July 1985.

[GILS] D. Gusfield, R. Irving, P. Leather, M. Saks. Every Finite Distributive Lattice is a Set of Stable Matchings for a *Small* Stable Marriage Instance. Journal of Combinatorial Theory, A. to appear.

[I] R. Irving. An Efficient Algorithm for the Stable Room-mates Problem. J. of Algorithms 6 (1985) 577-595.

[IL] R. Irving and P. Leather. The complexity of counting stable marriages. SIAM J. on Computing, to appear.

[ILG] R. Irving, P. Leather and D. Gusfield. An Efficient Algorithm for the "Optimal" Stable Marriage. July 1985.

[IR] M. Iri. Structural Theory for the Combinatorial Systems Characterized by Submodular Functions, in *Progress in Combinatorial Optimization*, Academic Press, Canada, 1984.

[K] D. E. Knuth. *Mariages Stables* (in French). Les Presses de L'Universite de Montreal, 1976.

[L] E. Lawler. *Combinatorial Optimization: networks and matroids.* Holt, Rhinehard and Winston, 1976.

[LLR] E. Lawler, J.K. Lenstra, and A.H.G. Rinnooy-Kan. Generating all Maximal Independent Sets: NP-hardness and Polynomial-Time Algorithms. Preprint.

[N] M. Nakamura. Boolean sublattices connected with minimization problems on matroids. Mathematical Programming, Vol. 22 (1982), pp. 117-120.

[PTW] G. Polya, R.E. Tarjan, D.R. Woods. *Notes on Introductory Combinatorics*, Birkhauser Verlag, Boston, 1983.

11. Acknowledgements

There are many people I want to thank. First, I thank the students in the fall 1985 Yale computer science 260 class, who generated many useful execution trees at the beginning of this research; the trees of A. Zoler and E. Winters were particularly helpful, the latter demonstrating a counter-example to an early conjecture of mine. I thank David Warren who acted as a sounding-board, and who contributed the version of phase 1 of algorithm I that appears in this paper. I also thank all the people in the Yale theory group, particularly Dana Angluin, who listened to many parts of this work as it developed. Finally, I thank Carrie Shepard, my stable roommate and partner in stable marriage.

1	7	2	6	8	5	3	4
2	4	6	5	3	8	1	7
3	5	2	1	7	4	6	8
4	1	7	3	6	5	8	2
5	7	1	8	4	6	2	3
6	7	3	8	4	5	1	2
7	2	8	4	3	5	6	1
8	4	2	3	5	6	7	1

1a. 8 person preference lists.

(1,6)	(1,5)	(1,4)
(2,3)	(2,3)	(2,3)
(7,4)	(7,4)	(7,5)
(8,5)	(8,6)	(8,6)

1b. Three stable roommate assignments.

1	2	6	5	3	4		
2	6	5	3	8	1		
3	5	2	1	7	4	6	
4	1	7	3	6	5	8	
5	7	1	8	4	6	2	3
6	3	8	4	5	1	2	
7	8	4	3	5			
8	4	2	5	6	7		

2a. Phase 1 table.

E_1	H_1	S_1
1	2	6
2	6	5
3	5	2

E_2	H_2	S_2
4	1	7
5	7	1

2b. Rotation $R_1 = (E_1, H_1, S_1)$ Rotation $R_2 = (E_2, H_2, S_2)$ both exposed in the phase 1 table. In the phase 1 table 6, 7, 8 is a tail of R_1, and R_2 has no tail.

1	6 5 3 4
2	5 3
3	2 1 7 4 6
4	1 7 3 6 5 8
5	7 1 8 4 6 2
6	3 8 4 5 1
7	8 4 3 5
8	4 5 6 7

2c. Table after eliminating R_1 from the phase 1 table. Note that R_2 is still exposed, and now has a tail of 3. Rotation $R_3 = (E_3, H_3, S_3)$ where $E_3 = \{2,6,7,8\}$ is now also exposed.

1	6 5
2	5 3
3	2 4 6
4	7 3 6 5 8
5	1 8 4 6 2
6	3 8 4 5 1
7	8 4
8	4 5 6 7

2d. Table after eliminating R_2. R_3 is the only exposed rotation.

1	6 5
2	3
3	2
4	7
5	1 8
6	8 1
7	4
8	5 6

Table after elimination of R_3. Now R_4, R_5 are exposed, where $E_4 = \{1,8\}$ $E_5 = \{5,8\}$.

1	5
2	3
3	2
4	7
5	1
6	8
7	4
8	6

Table after elimination of R_4.

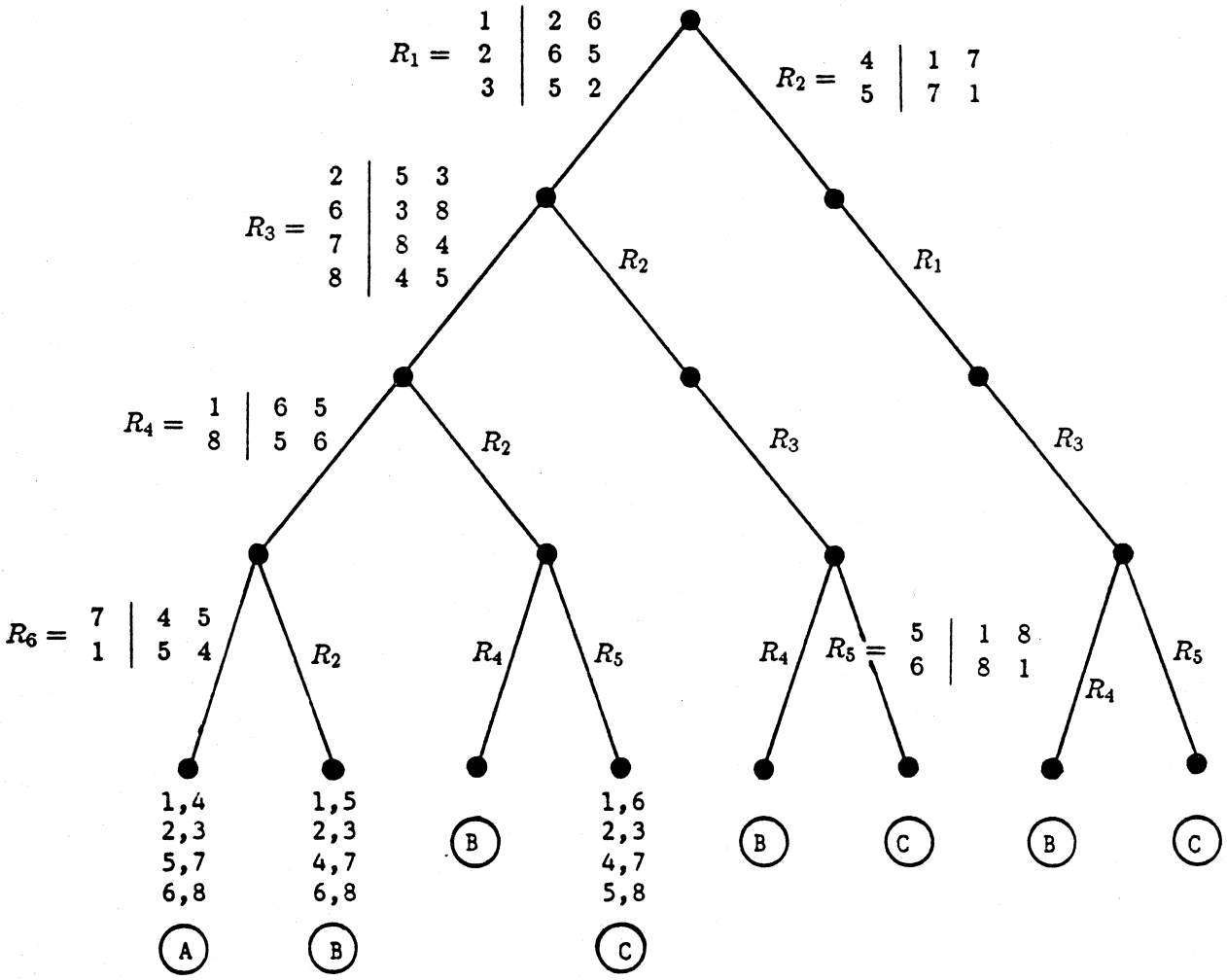
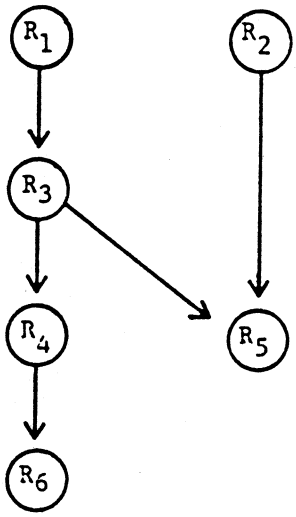
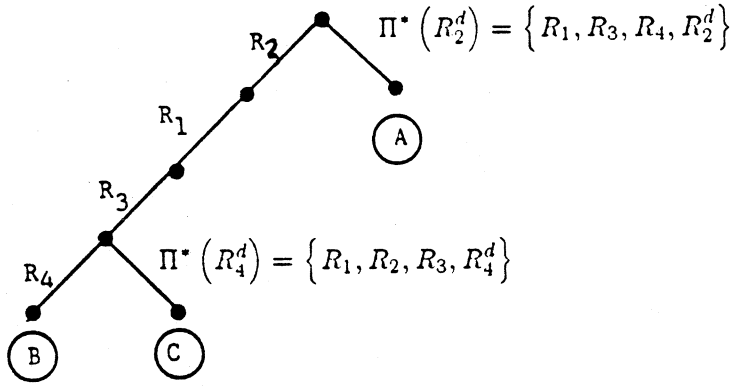


Figure 3. Tree D for the example. The tables at the nodes are not shown. All paths have length 4. (R_4, R_5) and (R_2, R_6) are each a dual pair of rotations. R_1 and R_3 have no duals.

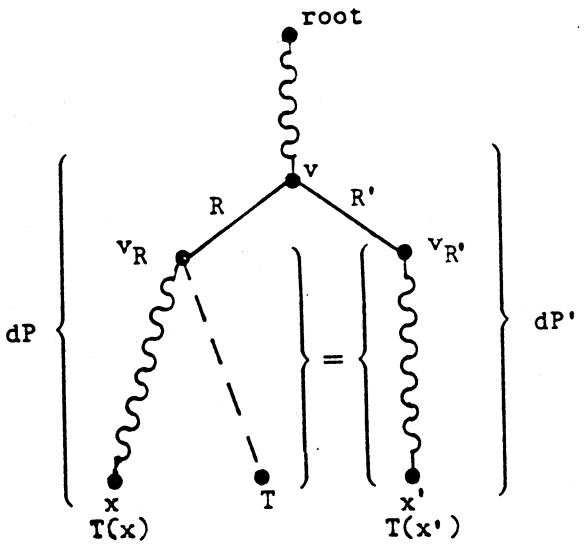


π^*

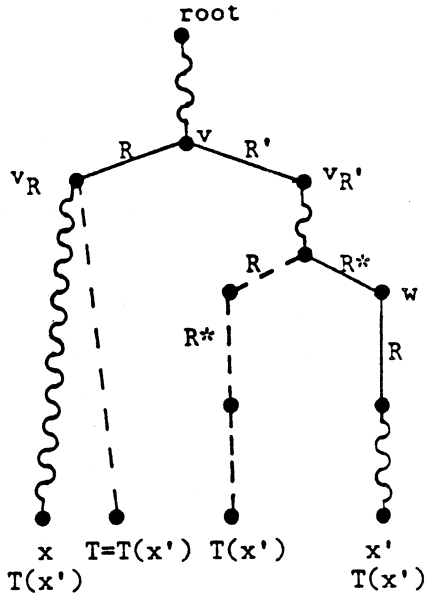
4.



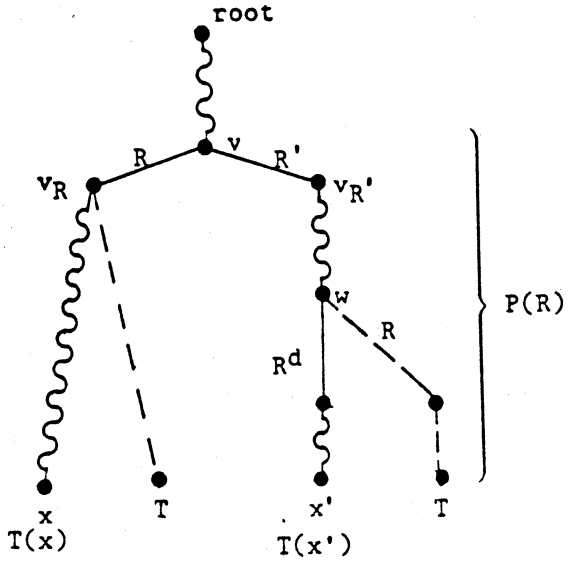
5. Tree B given by the dual enumeration method. The labels at the leaves refer to assignments given in Figure 3.



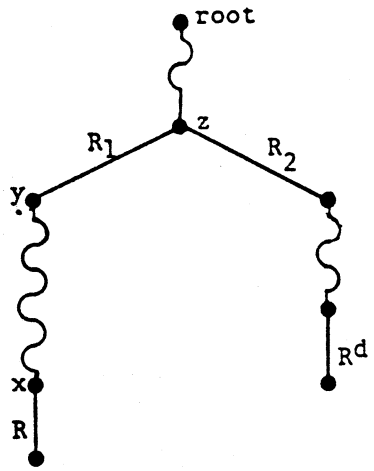
6a. Proof of Lemma 6.3, when $R' = R^d$. Wavy lines are known paths in D, solid edges are known single edges in D, labelled dashed lines are inferred edges, and unlabelled dashed lines are inferred paths in D.



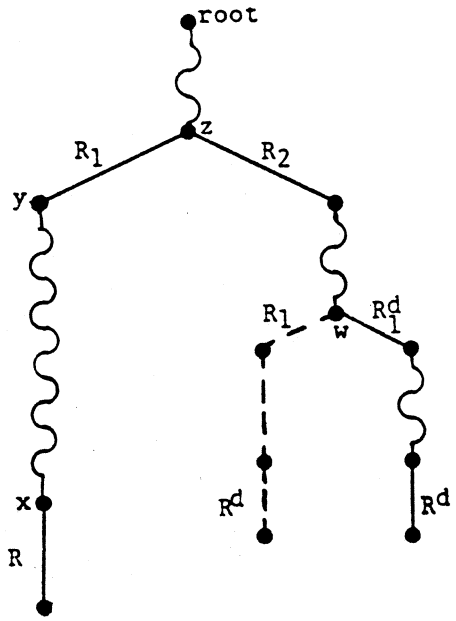
6b. $R' \neq R^d$ and R^d is on dP'



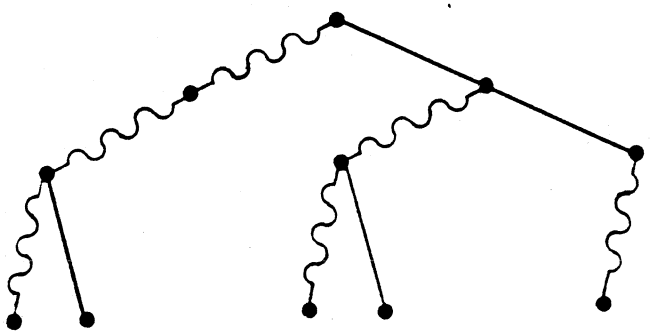
6c. $R' \neq R^d$ and R^d is on dP'



7a.



7b. R^d is on P after w



8. Schematic tree B. The maximal paths of left edges are drawn with wavy lines.