August1 989

Report No. STAN-CS-89-1275

A New Approach to Stable Matching Problems

by

Ashok Subramanian

Department of Computer Science

Stanford University

Stanford, California 94305

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	Unrestricted: Distribution Unlimited
4 PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-89-1275	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Computer Science Department 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Stanford University Stanford, CA 94305	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION ONR 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-88-K-0166
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

A New Approach to Stable Matching Problems

12 PERSONAL AUTHOR(S)
Ashok Subramanian

13a TYPE OF REPORT	13b TIME COVERED FROM ____ TO ____	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
		August 1989	34

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Abstract. We show that Stable Matching problems are the same as problems about stable configurations of X-networks. Consequences include easy proofs of old theorems, a new simple algorithm for finding a stable matching, an understanding of the difference between Stable Marriage and Stable Roommates, $\mathcal{NP}$-completeness of Three-party Stable Marriage, $\mathcal{CC}$-completeness of several Stable Matching problems, and a fast parallel reduction from the Stable Marriage problem to the Assignment problem.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	
22a NAME OF RESPONSIBLE INDIVIDUAL Ernst Mayr	22b TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete*

SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

A New Approach to Stable Matching Problems *

Ashok Subramanian
Department of Computer Science
St anford University
Stanford, CA 94305-2140

Abstract. We show that Stable Matching problems are the same as problems about stable configurations of X-networks. Consequences include easy proofs of old theorems, a new simple algorithm for finding a stable matching, an understanding of the difference between Stable Marriage and Stable Roommates, $\mathcal{NP}$-completeness of Three-party Stable Marriage, CC-completeness of several Stable Matching problems, and a fast parallel reduction from the Stable Marriage problem to the Assignment problem.

Key words. stable matching, stable marriage, stable roommates, stable configuration, circuit value, network stability, parallel complexity, comparator, CC, scatter, assignment problem, weighted matching

AMS(MOS) subject classifications. 05A05, 68Q10, 68Q15, 68Q20, 68R05, 90C27, 94C99

Abbreviated title. New Approach to Stable Mat ching

1 Introduction

In a Stable Matching problem, the task is to match a number of persons in pairs, subject to certain preference information. Briefly, each person regards some of the others as acceptable mates, and ranks them in order of preference. A matching is unstable if there are two persons, not matched to each other, who would rather be together. The task is to find a stable matching, i.e., one that is not unstable.

This problem **was first studied by** Gale and Shapley [7]. They showed that a stable matching always exists if the problem is a Marriage problem, i.e., if the participants can be divided into two sexes, the men and the women, in such a way that the acceptable mates of each person are all of the opposite sex; in fact, they gave a linear-time algorithm to find such a matching. Irving, in [14], gave a linear-time algorithm for the general problem. An introductory treatment of Stable Matching appears in [23]; a comprehensive treatment may be found in [12].

This paper explores the relationship between Stable Matching and Network Stability. For our purposes, a network is a boolean circuit with feedback. The task in Network Stability is, given a

*This work was supported by ONR contract N00014-88-K-0166.

network and its inputs, to find an assignment of boolean values to the edges of the network that respects the input constraints and the gate equations. The Network Stability problem is $\mathcal{NP}$-complete in general [21], but when every gate in the network is a special gate called the X-gate, the problem becomes equivalent to Stable Matching. The X-gate has a certain nice property: it is scatter-free. We exploit this property to solve X-Network Stability efficiently.

The appropriateness of the new framework is illustrated by new easy proofs of old theorems and simple algorithms. We give simple linear-time algorithms for finding a stable matching and for finding a 'minimum-regret' stable matching. For both of these problems, optimal algorithms were already known. Also, we propose a novel method of finding a 'random' stable marriage. We give an easy proof of the theorem of [8] that every stable matching of a given instance matches the same set of persons. We also prove a structural theorem that might help us understand why certain instances of Stable Matching have no solutions. We give a new simple proof of the #P-completeness of Stable Matching; this result, in a stronger form, was first proven in [16]. It turns out that just as Stable Matching is equivalent to Network Stability for X-networks, the 'three-party' version of Stable Matching is equivalent to Network Stability for Y-networks. We use properties of the Y-gate to conclude that Three-party Stable Marriage is $\mathcal{NP}$-complete, thus answering a question posed in [20].

It has been observed repeatedly that the Stable Marriage problem seems easier and has more structure than Stable Matching. We explain this difference by observing that the networks corresponding to Stable Marriage are comparator networks. The comparator gate is a simpler gate than the X-gate. The monotonicity of the comparator is responsible for the fact that Stable Marriage instances always have solutions. Also, Conway's lattice theorem for stable marriages can be viewed as a direct consequence of the lattice theorem for comparator networks.

We also study the parallel complexity of Stable Matching. While we do not give any fast parallel algorithms for the problem, we do point out that several problems related to Stable Matching are complete for the class CC of problems defined in [21]. We use these ideas to give a fast parallel reduction from the Stable Marriage problem to the Assignment problem, thus partially solving a problem of [20].

Our approach does not seem to apply when the instances of Stable Matching have ties, or when issues of deceit are involved. It does not explore the structure of all stable matchings of an instance, as does the work of [10, 11, 17], or the more recent work of [5].

2 Preliminaries

2.1 Stable Matching

An instance of Stable Matching (also called Stable Roommates) consists of a set of persons, each of whom regards some of the others as acceptable mates, and ranks them in decreasing order of preference. For our purposes we may assume that acceptability is mutual: x is acceptable to y if and only if y is acceptable to x. A matching is a pairing of some or all of the persons. A matching may be *unstable* in three ways: two unmatched persons may find each other acceptable; a matched person may prefer an unmatched person to his current mate; or two matched persons may prefer each other to their current mates. A matching that is not unstable is said to be *stable.*

An instance of Stable Matching is an instance of Stable Marriage if the persons can be divided into two sets, the men and the women, so that the acceptable mates of each person are all of

the opposite sex. An instance of Stable Matching is an instance of Complete Stable Matching if there are an even number of persons, and each person is acceptable to everyone else. Similarly, an instance of Stable Marriage is an instance of Complete Stable Marriage if there are an equal number of men and women, and each person is acceptable to every person of the opposite sex.

The size of an instance of Stable Matching is the sum, over all persons x, of the number of persons acceptable to x. The most common tasks associated with an instance of Stable Matching are to determine whether a stable matching exists, and to construct one if possible. Other tasks might include counting and enumerating all stable matchings of a given instance.

Remark: Historically, most of the interest in the literature has been in Complete Stable Marriage and Complete Stable Matching; these problems are commonly called the Stable Marriage and Stable Roommates problems respectively. Also, it has been traditional to measure the running time of algorithms in terms of the number of participants, and not in terms of the 'length of the input'. Our departure from this tradition might lead to one cause of confusion: algorithms that we describe as linear-time (in terms of the size of the instance) might be described in the literature as taking quadratic time (in terms of the number of participants).

2.2 Gates, circuits and networks

The definitions in this section and in Section 2.3 are taken almost entirely from [21].

A X-input, p-output *gate* is a function $g : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{\mu}$ from X-bit input words to μ-bit output words. In this paper, all gates have a fixed number of inputs and outputs. Further, a gate does not have useless inputs or outputs: each input bit affects some output bit, and each output bit depends non-trivially on some input bit. A *basis* is a set of gates. The gates that appear in this paper are listed in Table 1.

Gate	Inputs	Outputs
ID	i_1	$o_1 = i_1$
COPY	i_1	$o_1 = i_1, o_2 = i_1$
NOT	i_1	$o_1 = \overline{i_1}$
AND	i_1, i_2	$o_1 = i_1\, i_2$
OR	i_1, i_2	$o_1 = i_1 + i_2$
NAND	i_1, i_2	$o_1 = \overline{i_1} + \overline{i_2}$
C	i_1, i_2	$o_1 = i_1\, i_2, o_2 = i_1 + i_2$
X	i_1, i_2	$o_1 = i_1\, \overline{i_2}, o_2 = \overline{i_1}\, i_2$
Y	i_1, i_2, i_3	$o_1 = i_1\,(\overline{i_2} + \overline{i_3}), o_2 = i_2\,(\overline{i_1} + \overline{i_3}), o_3 = i_3\,(\overline{i_1} + \overline{i_2})$

Table 1: Some gates

A *network* is a finite labelled directed graph. Source (in-degree zero) nodes of the directed graph have out-degree one and are called input nodes; sink (out-degree zero) nodes have in-degree one and are called output nodes. Each internal node is labelled with a gate and an ordering of its predecessors and successors. If an internal node has in-degree λ and out-degree μ, its gate has λ inputs and μ outputs. If the underlying directed graph of a network is acyclic, the network is called a *circuit.* A circuit computes a set of functions in the standard manner. A network (circuit)

is said to be *over* a basis Ω if every gate in it is from Ω. The size of a network is the number of edges in it.

Our definition of circuit differs from the standard definition in two respects: all fanout is explicit and must occur within a gate, and multiple-output gates are allowed. Our definition of 'network' is non-standard. A network is essentially a combinational circuit with feedback; it is not intended to compute anything.

A gate g_1 is a *restriction* of gate g_2 if it can be obtained as follows: Fix a subset (possibly empty) of the inputs of g_2 to constant values. This operation fixes a (possibly empty) set of outputs of g_2 to constant values. Now discard all the inputs and outputs of g_2 that are fixed, and perhaps some outputs that are not fixed. For example, the NOT gate is a restriction of the X-gate, obtained by fixing any one input of the X-gate to 1.

Let Ω be any basis. Define Ω^*, the closure of Ω, to be the basis consisting of all gates in Ω, together with all their restrictions. For example, X^*, the closure of the basis {X}, consists of the X-gate, the NOT gate, and the *ID* gate.

Gate g can be *simulated* by basis Ω if g is a restriction of the function computed by some circuit over Ω. We say, for convenience, that gate g_1 can be simulated by gate g_2 when we mean g_1 can be simulated by the basis $\{g_2\}$; similarly, we say a basis can be simulated (by another basis) if every gate in it can be simulated. For instance, a gate (or basis) is monotone if it cannot simulate $\{NOT\}$.

A gate has *scatter* if it has a restriction that has more outputs than inputs; otherwise, it is *scatter-free.* A basis is said to be scatter-free if every gate in it is scatter-free. All the gates in Table 1 are scatter-free, except the *COPY* gate and the Y-gate.

The X-gate and the C-gate (comparator) play crucial roles in this paper. These gates are scatter-free; in addition, the comparator is monotone. Many of the results of this paper rely on these elementary facts.

The following easy lemma is crucial.

Lemma 1 *Let N be any network over a scatter-free basis Ω. Then N has at most us many outputs us inputs. In particular, if N has no inputs, it has no outputs.*

Proof: Let *Gates* be the set of gates of N. Then

$$\#(\text{inputs of N}) - \#(\text{outputs of N}) = \sum_{g \in Gates} (\#(\text{inputs of } g) - \#(\text{outputs of g}))$$

Since every gate in *Gates* is scatter-free, each summand on the right hand side is non-negative; hence the sum is non-negative. II

2.2.1 X-networks, and snakes in X-networks

An X-network is a network over {X}. We introduce some terminology to describe X-networks. Recall that the X-gate has inputs i_1, i_2 and outputs $o_1 = i_1 \overline{i_2}$, $o_2 = \overline{i_1} i_2$. We *associate* input i_j and output o_j. This association allows us to describe an X-network in terms of *snakes.* A snake is a sequence of edges; the sequence may be cyclic or acyclic. If an input edge of an X-gate is in snake q, the associated output edge is the next element of q; if an output edge of an X-gate is in q, the associated input edge is the previous element of q. The first element of an acyclic snake is thus

an input edge of the network; the last element of an acyclic snake is an output edge of the network. The snakes of a network constitute a partition of its edges.

We often describe an X-network by naming its snakes and telling how they meet each other; in such a description, the gates are implicit: there is a gate wherever two snakes meet. We say that a snake enters a gate with value v if the edge of the snake that is an input to the gate has the value v. (Some care is needed, because a snake could enter the same gate twice, once through each input.)

2.3 The Circuit Value and Network Stability problems

The Circuit Value problem (CV) is the task of computing the value assigned to a specified output of a circuit, when the circuit is evaluated on a given input assignment. More formally, given a circuit with λ inputs, and an input assignment $s_{in} \in \{0, 1\}^{\lambda}$ to the circuit, determine whether the specified output takes the value 1.

The Network Stability problem is a question about the existence of *configurations* of a network consistent with a given input assignment. Let N be a network with $\lambda \geq 0$ inputs. Let $s_{in} \in \{0, 1\}^{\lambda}$ be an input assignment to N. A *configuration* Q of N is an assignment of boolean values (0s and 1s) to the edges of N. It is a *stable configuration* of $[N, s_{in}]$ if it satisfies the gate equations at each internal node, and is consistent with the input assignment s_{in}. The *network stability problem (NS)* is this: Given a network N and an input assignment s_{in}, determine whether [N, s_{in}] has a stable configuration. Often we leave the input assignment implicit, and say "N has a stable configuration" when we mean "[N, s_{in}] has a stable configuration."

Given a basis Ω, we define Ω-CV and Ω-NS to be the special cases of CV and NS respectively in which the network is required to be over Ω. For instance, C-CV is the circuit value problem for comparator circuits, and X-NS is the stability problem for X-networks.

The input assignment that assigns every input of a network the value 1 is called **1;** the empty input assignment is called e.

3 The Stable Matching problem is an X-Network Stability problem

We show how to transform an instance I of Stable Matching into an X-network N in such a way that the stable matchings of *I* are in one-to-one correspondence with the stable configurations of [N, **1**]. Here is the transformation: Network N has one acyclic snake q_x for each person x of instance *I;* there are no other snakes. Snakes q_x and q_y meet if and only if x and y find each other acceptable; snake q_x meets the snakes of other persons in the order that these persons appear on the preference list of person x, i.e., in decreasing order of their popularity with x. Any two snakes meet at most once; no snake meets itself. This description completely specifies N. We present a method to construct N given the description. For each pair x, y of persons who find each other acceptable, introduce an X-gate labelled with the pair {x, y}. For each person, introduce a snake that visits all the gates whose labels contain his name, in the order determined by his preference list. This completes the construction. An example is shown in Figure 1. (In this figure, the snake of each person is labelled with his name for clarity.)

In Section 3.1 below, we prove a lemma about stable configurations of X-networks. We then prove the correspondence between stable matchings and stable configurations in Section 3.2.

Person	Preferences		
A	D	B	
B	D	C	A
C	B	D	
D	A	B	C

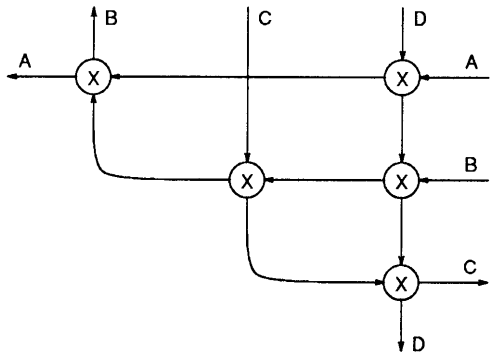


Figure 1: Reducing Stable Matching to X-Network Stability

3.1 A preliminary lemma about stable configurations of X-networks

Let N be any X-network without cyclic snakes. Let Q be a stable configuration of [N, 1]. The stability of Q places two conditions upon it: it must satisfy the input assignment, and it must satisfy all the gate equations. Let us investigate what these conditions really mean. Let $q = \langle e_1, e_2 \ldots \rangle$ be any snake of N; for each j, let v_j be the value on edge e_j. Every input edge has the value 1, so $v_1 = 1$. The behaviour of the X-gate requires that $v_j = 0 \Rightarrow v_{j+1} = 0$. Thus, the sequence $\langle v_1, v_2 \ldots \rangle$ is a sequence of 1s, followed perhaps by a sequence of 0s. Define the *l-segment* of snake q in configuration Q to be the subsequence of edges that have value 1.

We say that snake q *drops* at gate g if the l-segment of q ends at gate g; we say that g is the endpoint of the l-segment of q. (A snake that has the value 1 along its entire length is identical with its l-segment, and has no endpoint.) It is clear that each snake drops at most once.

A snake that enters a gate with value 1 will drop if and only if the other snake entering the gate also enters with value 1. Thus, snakes must drop in pairs. Also, the l-segments of different snakes cannot share vertices, except possibly a common endpoint. We describe this situation informally by saying that l-segments avoid one another.

It turns out that the conditions we have listed above are necessary and sufficient for Q to be a stable configuration. In other words,

Lemma 2 *Let N be an X-network without cyclic snakes; let Q be a configuration of [N, 1]. Then Q is stable if and only if*

a) *the values on any given snake of N form a non-empty sequence of 1s, followed by a possibly empty sequence of Os,*

b) snakes drop in pairs, and

c) the l-segments of different snakes avoid one another.

Proof: We have already seen that the above three conditions will be met if Q is stable. Suppose now that Q is a configuration that meets the three conditions; we show that Q must be stable. The first value on each snake is a 1, so the input conditions are satisfied. Let g be an arbitrary gate of N; we must show that the gate equations are satisfied at g. If both snakes entering g enter it with value 0, condition a) requires that both outputs of g be 0 as well. If one snake is 0 and the

other is 1 while entering g, conditions a) and b) demand that both snakes leave g with the same values they entered it with. If both snakes are 1 while entering g, condition c) demands that both drop. In all three cases, the gate equations are satisfied. □

3.2 The correspondence between stable matchings and stable configurations

Theorem **3** *Let I be an instance of Stable Matching; let N be the corresponding X-network. Then there is a one-to-one correspondence between the stable matchings* ***of*** *I and the stable configurations of* $[N, 1]$.

Proof: The rule to read off a matching of *I* from a stable configuration of [N, **1]** is this: two persons are matched to each other if their snakes drop together; a person is unmatched if his snake does not drop. To produce the configuration of [N, 1] corresponding to a given matching of *I, we* invert the above rule: If person x is matched to person y, snake q_x takes the value 1 until it meets snake q_y, and the value 0 thereafter; if x is unmatched, snake q_x is 1 along its entire length. We now show that stable configurations yield stable matchings and vice-versa.

Suppose that stable configuration Q yields matching M. Assume that M is unstable. Hence there must be two persons x and y, each of whom prefers the other to his mate in M, or to the state of being unmatched. Since x and y are acceptable to each other, snakes q_x and q_y meet. We claim that q_x is 1 after meeting q_y . To prove the claim, we consider two cases. If x is unmatched, then q_x must be 1 along its entire length, and we are done. In the other case, let z be the mate of x. Then q_x and q_z must meet with value 1; hence q_x must be 1 all the way until it meets q_z. Since x prefers y to z (by hypothesis), q_x meets q_y *before* it meets q_z. It follows that q_x is 1 after meeting q_y; this proves the claim. By an entirely analogous argument, q_y is 1 after meeting q_x. This means that the l-segments of q_x and q_y do not avoid one another. Lemma 2 tells us that Q must be unstable, which is a contradiction. Hence M must be stable.

Now suppose that stable matching M yields configuration Q. It is clear that Q meets conditions a) and b) of Lemma 2. To prove that Q is stable, it suffices to show that condition c) is met as well. Assume, for the sake of contradiction, that there are two persons whose snakes violate condition c). Then it follows that each of these persons prefers the other to his mate (if any) in M, contradicting the stability of M. □

Remark: We reiterate the meaning of the value assigned to an edge by a stable configuration of [N, 1]. Let e be the edge of snake q_x just after q_x meets q_y. Then e is assigned the value 0 in a stable configuration of [N, 1] if and only if x has a mate no worse than y in the corresponding stable matching of *I.*

4 Solving Network Stability for Scatter-free Networks

4.1 The algorithm

In this section, we show how to construct a stable configuration of a network over any scatter-free basis. This algorithm was presented in [21]. The X-gate is scatter-free, so the results of this section apply to Stable Matching. In particular, we get a simple linear-time algorithm for constructing a Stable h/latching. A different linear-time algorithm for this problem was already known [14].

Let N be any network over a scatter-free basis Ω; let s_{in} be an input assignment to N. We show how to find a stable configuration of $[N, s_{in}]$. The interesting case is when N has no inputs, i.e., when s_{in} is the null assignment e. Otherwise, we could use the input assignment to simplify the network. After all, if we know the value of an input to a gate g, we can replace g by a restriction with fewer inputs, and perhaps deduce the value of some of the outputs of g. Iterating this process of 'input elimination' yields a partial assignment *P* of values to edges, and an input-free network N' on the remaining edges, with the property that the stable configurations of [N, s_{in}] are obtained by augmenting the stable configurations of [N',e] with *P*. In particular, $[N, s_{in}]$ has a stable configuration if and only if [N',e] does. Notice that the time taken to perform input elimination is linear in the amount of simplification performed, i.e., in the number of edges that are assigned values by the partial assignment *P*. The order in which the inputs are eliminated is immaterial; in fact, the process is properly regarded as a concurrent process, although we do not use this fact here. Lemma 1 assures us that N' will have no outputs. Henceforth we will simply assume that N has no inputs or outputs.

Consider what happens if we break an edge e of N, thus creating an input $e(in)$ and an output $e(out)$, and then place the value v $\in \{0,1\}$ on $e(in)$. We get a network with one input and one output; we may apply input elimination to this network. We call this the process of *propagating* the pair [e, v]. The following observation is crucial: the elimination of the sole input of the network must assign a value, say $v(out)$, to the output edge $e(out)$. To see this, apply Lemma 1 to the network that remains after the propagation. If *v(out)* = v, we say that the propagation is *successful,* and that $[e, v]$ is *viable* in N. The result of a successful propagation is a partial assignment *P* and an input-free network N' on the remaining edges, with the property that the stable configurations of [N, e] that assign edge e the value v are obtained by augmenting the stable configurations of [N',e] with *P*. If, on the other hand, $v(out) \neq$ v, no stable configuration of $[N, \mathbf{e}]$ may assign edge e the value v; in this case, we say the propagation is *unsuccessful.* Again, the time taken by a successful propagation is linearly related to the amount of simplification achieved.

The *type* of an edge in an input-free network is the set of viable values for it, i.e., $type_N(e)$ = $\{v | [e, v]$ is viable in N$\}$. If $type_N(e) = \emptyset$, we say that *e* is a *contradicting* edge. Clearly, if the edge we select is contradicting, there can be no stable configurations. Also, if its *type* is $\{0\}$ or {1}, there is only one possible value that this edge could take in any stable configuration; so we can assign this edge its value, and simplify the network accordingly. The difficulty arises when both values seem possible, i.e., when $type_N(e) = \{0,1\}$.

Our strategy in such a situation is to pick *either* value, and proceed. Clearly, if we are lucky, we will make all the right choices, and will find a stable configuration. But what if we are unlucky? Might not a set of bad choices early on in the algorithm lead us to a situation where we can no longer complete the partial assignment we have to a stable configuration? Lemma 4 below comes to our rescue, because it says that the *type* of an edge is preserved by successful propagations. This means that if we find a contradicting edge after performing a sequence of successful propagations, then the same edge must have been contradicting in the original network as well, meaning that there were no stable configurations to start with. In other words, there are no bad choices.

Lemma 4 *Let N be an input-free network over a scatter-free basis* Ω. *Suppose that* $[e_1, v_1]$ *is viable in N. Let* N' *be the network* ***left*** *after propagating* $[e_1, v_1]$ *in N; let* e_2 *be an edge* of *N'. Then* $type_{N'}(e_2) = type_N(e_2)$.

Proof: This proof is due to Feder [6]; our original proof was much more complicated. We make

the following definition. A trace of the propagation of $[e_0, v_0]$ in N is a sequence of statements that records all the assignments that are made during this propagation, alongwith the dependencies between the assignments. Each statement is of the form e ← v, meaning 'edge e gets value v'. The first statement is $e_0 \leftarrow v_0$ and constitutes the *preamble* of the trace; the rest of the trace is called its *body*. A statement in the body of the trace is said to be *terminal* if it mentions edge e_0. All other statements of the trace are *non-terminal*. Every statement in the body is derived from a subset of the non-terminal statements preceding it in the trace by invoking the equation of a single gate. The trace is complete, in the sense that all statements that are derivable in the above manner are included. A statement may occur at most once in the body of the trace; any subsequent occurrences would be redundant.

Each statement of a trace may be regarded as a step in the propagation process. A propagation succeeds if the statement $e_0 \leftarrow v_0$ appears in the body of the trace; it fails if the statement $e_0 \leftarrow \overline{v_0}$ appears in the body. Clearly both statements cannot appear simultaneously in the body of any trace; on the other hand, we know that at least one of these statements will appear in the body, because the network is input-free and scatter-free.

Let T_1 be a trace of $[e_1, v_1]$ in N; let T_2 be a trace of $[e_2, v_2]$ in N. The effect of propagating $[e_1, v_1]$ is to assign values to some edges of N. We would like to claim that, even after these assignments are made, each statement in the body of T_2 can still be inferred from the preamble of T_2. This would mean that propagating $[e_2, v_2]$ in N' will establish every assignment that would have been made by the propagation of $[e_2, v_2]$ in N that has not already been made by the propagation of $[e_1, v_1]$ in N. It would then follow that $[e_2, v_2]$ is viable in N if and only if it is viable in N', completing the proof of Lemma 4.

To prove the claim, it suffices to show that there is no 'conflict' between the two traces. A conflict is a situation where the statement e ← v appears in one trace while the statement e ← $\overline{v}$ appears in the other. We assume a conflict exists, and derive a contradiction. Let S_2 be the earliest statement in T_2 that is involved in a conflict; assume it says $e_c \leftarrow v_c$, and that it conflicts with statement S_1 of T_1. Statement S_2 cannot be in the preamble of T_2, because, by hypothesis, edge e_2 does not appear in T_1. Hence, edge e_c is the output of some gate; let this gate be g,. The gate equations used to derive S_1 and S_2 assert that g_c has two partial input assignments P_1 and P_2, such that on one of these assignments, output e_c takes the value v_c, whereas on the other assignment, output e_c takes the value $\overline{v_c}$. Now, by the choice of S_2, the partial input assignments P_1 and P_2 do not conflict. Let P be the partial input assignment that is the union of P_1 and P_2. Consider the behaviour of gate g_c on P. Clearly, output e_c must be both 0 and 1 in this situation, which is a contradiction. II

An immediate consequence of Lemma 4 is the correctness of the following simple polynomial-time algorithm for constructing stable configurations of networks over any scatter-free basis:

Algorithm 1

1. Eliminate all inputs.
2. Pick any edge e. (If there are no edges, there is nothing to do.)
3. Determine that e is contradicting (in which case there are no stable configurations), or find a value v such that [e, v] is viable.
4. Suppose [e, v] is viable. Propagate $[e, v]$, obtaining a partial assignment P and a simplified network N'.

5. Find a stable configuration of N' and augment it with *P* to yield the desired stable configuration. If N' has no stable configuration, neither does N.

Theorem *5 Let N be a network over a scatter-free basis Ω; let s_{in} be any input assignment to N. There is a linear-time algorithm that will construct a stable configuration of $[N, s_{in}]$ if there is one, or conclude that none exists.*

Proof: Algorithm 1 can be implemented to run in time that is linear in the size (number of edges) of the network N. Notice that steps 1 and 4 take time linear in the amount of simplification they perform, and step 2 takes constant time. Also, if step 3 determines that e is contradicting, the time needed to do so is linear in the size of the network. The only obstacle to a linear run-time is that step 3 may take a long time to return a viable value v. To avoid this, we use the following standard trick. Implement step 3 as follows: Propagate [e, 0] and [e, 1] 'in parallel'; whichever propagation succeeds first determines the value v to be used in step 4. This guarantees that step 3 runs in time linear in the amount of simplification performed in step 4. The linear time bound for the algorithm follows from the observation that the size of N' in step 5 is less than the size of N by the amount of simplification achieved in step 4. □

The following corollary is useful in the context of parallel computation:

Theorem 6 *Let N be an input-free network over a scatter-free basis Ω. Then N has a stable configuration if and only if it has no contradicting edge.*

Proof: If N has a contradicting edge, this edge can take neither the value 0 nor the value 1 in any stable configuration. Hence there cannot be any stable configurations. If, on the other hand, there are no contradicting edges, Algorithm 1 will find a stable configuration. cl

The *type* of an edge has a nice interpretation:

Theorem 7 *Let N be an input-free network over a scatter-free basis Ω; suppose that N has no contradicting edges. Let e be an edge of N. Then the type of e in N equals the set of values assigned to e in the (various) stable configurations of N.*

Proof: Clearly, if [e, v] is not viable in N, then no stable configuration of N may assign e the value v. If $[e, v]$ is viable, on the other hand, we can construct a stable configuration that assigns e the value v, by letting [e, v] be the first edge-value pair picked by Algorithm 1. cl

The following theorem is proven for Stable Marriages in [8]; a proof for the Stable Matching case can be found in [12]. We give an easy alternative proof:

Theorem *8 The set of unmatched persons in any instance of Stable Matching is the same in every stable matching.*

Proof: Let *I* be an instance of Stable Matching; let N be the corresponding X-network. The process of eliminating the inputs of N assigns a value to every output edge of N. Hence every stable configuration of [N, 1] has the same output assignment. The theorem follows by observing that a person is unmatched in a stable matching of instance *I* if and only if the output edge of his snake takes the value 1 in the corresponding stable configuration of [N, 1]. cl

4.2 Application to Minimum-regret Stable Matching

Algorithm 1 has two kinds of flexibility: edge e may be chosen to be any edge of the network, and there is some flexibility in the choice of value v. We have already demonstrated how to use the second kind of flexibility to make the algorithm run in linear time. Here we explore a different possibility.

Let I be any instance of Stable Matching. The *regret* of a participant in a stable matching of I is the position of his mate on his preference list. Thus, a participant who gets his first choice of mate has regret 1. Clearly, each participant would like to minimize his regret. In the Minimum-regret Stable Matching problem, we seek a stable matching that minimizes the largest regret among all the participants. A linear-time algorithm is given in [10]. However, this algorithm relies on non-trivial properties of the structure of all stable matchings of an instance of Stable Marriage; moreover, it works only for instances of Stable Marriage. Irving [15] gives a linear-time algorithm that works for all instances of Stable Matching; see [12] for an exposition. We give a different linear-time solution based on the network formulation.

We assume that I has a stable matching, and that no person is unmatched in the stable matchings of I. The second assumption is legal because, by Theorem 8, the unmatched persons are the same in every stable matching; hence we can find these unmatched persons by input-eliminati on in linear time, delete them, and find a minimum-regret matching of the remaining participants.

Define *regret(l)* to be the value of the largest regret in a minimum-regret stable matching of I. The crux of the problem is the computation of *regret(I)*. We first show how to proceed if *regret(I)* is known, and then show how to compute *regret(I)*.

Suppose *regret(I)* = r. Delete from each person's list all but the first r persons, and then make the acceptability relation symmetric: if person x does not appear on the list of person y, delete y from the list of person x. Let us call the resulting instance I_r. It is easily checked that the stable matchings of I_r are precisely the minimum-regret stable matchings of I. A stable matching of I_r may be found in linear time by using the strategy outlined in the proof of Theorem 5.

We show how a modified version of Algorithm 1 can compute *regret(I)* in linear time. Let N be the X-network for I. Number each edge of N by its position on its snake: input edges get the number 1; output edges get big numbers. The value of the largest regret in any stable matching is given by the largest number of an edge, among all the edges with value 1 in the corresponding stable configuration. This means that the way to get a small regret is to assign the value 0 to the edges with big numbers.

Run Algorithm 1 with the following modifications. Whenever we have to pick an edge-value pair to propagate, we choose an edge e that has the largest number, and assign it the value 0. Suppose edge e is the edge numbered k on the snake of person x. Then assigning e the value 0 amounts to guaranteeing that person x gets one of his first $(k - 1)$ choices. If the propagation of [e, 0] fails, there is no stable matching, consistent with the choices already made, that assigns x one of his first $(k - 1)$ choices; hence *regret(I)* must be k. (The reason it cannot be larger than k is because at the time we assign edge e the value 0, we have already succeeded in assigning every edge with number larger than k the value 0.) Also, if the propagation assigns the value 1 to another edge with number k, then again the value of the minimum regret is k. We continue assigning the value 0 to more and more edges with large numbers, until ultimately one of the two exit conditions applies. Thus this algorithm will compute *regret(I)*. To see that it takes linear time, observe that the time taken by all the successful propagations taken together is linear in the number of edges

of **N;** also, there is at most one unsuccessful propagation, which can take at most linear time.

5 The structure of the network of contradicting edges

Theorem 6 says that if an instance of X-network stability has no stable configurations, the associated input-free network contains at least one contradicting edge. In this section, we prove that the contradicting edges form disjoint cycles, each containing an odd number of *NOT* gates. We believe this is a step towards understanding why certain instances of Stable Matching do not have stable matchings.

Lemma 9 *Let N be any input-free network over X^*. Suppose that $[e_1, v_1]$ is viable in N, and that edge e_2 is assigned the value v_2 during the propagation **of** $[e_1, v_1]$. Then $[e_2, v_2]$ is viable in N.*

Proof: As in the proof of Lemma 4, let T_1 be a trace of $[e_1$, vi] in N, and let T_2 be a trace of $[e_2, v_2]$ in N. If $e_1 = e_2$, then $v_1 = v_2$ and there is nothing to prove; so assume that $e_1 \neq e_2$. Notice that the preamble of T_2 occurs in T_1, and every statement of T_1 occurs as a non-terminal statement in T_1. Hence we may show inductively that each statement of T_2 occurs in T_1. It follows that the value assigned to edge e_2 by the propagation of $[e_2, v_2]$ is v_2. cl

Lemma 9 says that successful propagations do not assign values to contradicting edges. By repeatedly performing successful propagations, we can assign values to all non-contradicting edges. What is left is an input-free network $N_\emptyset$ over X* consisting of exactly the contradicting edges.

Lemma 10 *The network $N_\emptyset$ **of** contradicting edges does not contain any X-gates.*

Proof: Assume the opposite, that $N_\emptyset$ contains an X-gate g, with input edges i_1, i_2 and output edges o_1, o_2. We derive a contradiction.

Let T_1 be a trace of $[o_1, 0]$ in $N_\emptyset$. Since o_1 is contradicting, the statement $o_1 \leftarrow 1$ must appear in the body of T_1. In order for this to happen, statements $i_1 \leftarrow 1$ and $i_2 \leftarrow 0$ must also appear in T_1; hence so must $o_2 \leftarrow 0$.

Now let T_2 be a trace of $[o_2, 0]$ in $N_\emptyset$. Edge o_2 must be assigned a value by the propagation of $[o_2, 0]$; hence at least one of the edges i_1 or i_2 must be assigned a value during this propagation. Let S be the first statement in the body of T_2 that assigns a value to any of the edges incident upon gate g. Clearly, statement S assigns a value to either i_1 or i_2. We claim that S appears in T_1. To prove the claim, observe that the preamble of T_2 occurs in T_1, and the derivation of S in T_2 does not involve edge o_1 in any intermediate step. Hence we may prove inductively that every statement involved in the derivation of S in T_2 occurs as a non-terminal statement in T_1, and the claim follows. Thus S must be either $i_1 \leftarrow 1$ or $i_2 \leftarrow 0$. In either case, the statement $o_2 \leftarrow 0$ will appear in the body of T_2. This indicates that $[o_2, 0]$ is viable, which is a contradiction. □

Theorem 11 *Each connected component **of** $N_\emptyset$ is a directed cycle over $\{ID, NOT\}$ with an odd number **of** NOT gates.*

Proof: Follows from the fact that $N_\emptyset$ is input-free and contains no X-gates. Each cycle must have an odd number of *NOT* gates in order for the edges on it to be contradicting. □

Remark: The results of this section extend naturally to the case where N is over any scatter-free basis, but the proof of Theorem 11 is more involved. (See [27] for details.)

6 The case of Stable Marriage

In this section, we examine the X-networks arising from instances of Stable Marriage. We show that these X-networks may be replaced by comparator networks. This observation, together with the monotonicity of the comparator, yield easy algorithms for certain problems associated with St able Marriage.

6.1 Bipartitionable X-networks and comparator networks

An X-network is *bipartitionable* if its snakes can be two-colored so that snakes of the same color never meet; such a coloring is called a bipartition. In any bipartition, the color of an edge is defined to be the color of the snake to which it belongs. In this paper, whenever we say that a network is bipartitionable, we shall implicitly mean that a bipartition is given. The X-networks corresponding to instances of Stable Marriage are bipartitionable: color the snakes of the men red and the snakes of the women blue. This shows that instances of Stable Marriage may be solved by solving X-Network Stability for bipartitionable X-networks. The converse is also true; see Section 7.3.

We adopt the following conventions: in all diagrams containing X-gates, an input and its associated output appear opposite each other; in diagrams of bipartitionable X-networks, the red edges are drawn vertically and the blue edges are drawn horizontally; in all bipartitionable networks derived from instances of St able Marriage, the red snakes belong to men and the blue snakes belong to women.

Given a bipartitionable X-network N and an input assignment s_{in}, we show how to transform them into a comparator network N' and an input assignment s'_{in} to N', in such a way that the stable configurations of [N, s_{in}] and [N', s'_{in}] are isomorphic. Bipartition the snakes of N into red and blue snakes. Replace every blue edge of the network by two *NOT* gates in series. Regard every X-gate g to be part of a triplet of gates: g, the *NOT* gate on the blue input of g, and the *NOT* gate on the blue output of g. Each triplet is functionally equivalent to, and hence may be replaced by, a comparator; see Figure 2(a). After these replacements, the only non-comparator gates in the network are the *NOT* gates on the blue inputs and outputs. These *NOT* gates just mean that the input and output assignments of the two networks do not match exactly: the blue bits have to be complemented. Delete the *NOT* gates to get N'. To get s'_{in}, complement the blue input bits of s_{in}.

Essentially the same transformation may be used to replace a comparator network by a bipartitionable X-network on a different input assignment. Proceed as follows: Replace each comparator by three X-gates as shown in Figure 2(b). A bipartition of the resulting X-network is obtained by coloring all the 'horizontal snakes blue and all the 'vertical' snakes red.

These transformations show that finding stable configurations in bipartitionable X-networks is just the same as finding stable configurations in comparator networks. Thus, we may find stable matchings for instances of Stable Marriage by finding stable configurations of comparator networks.

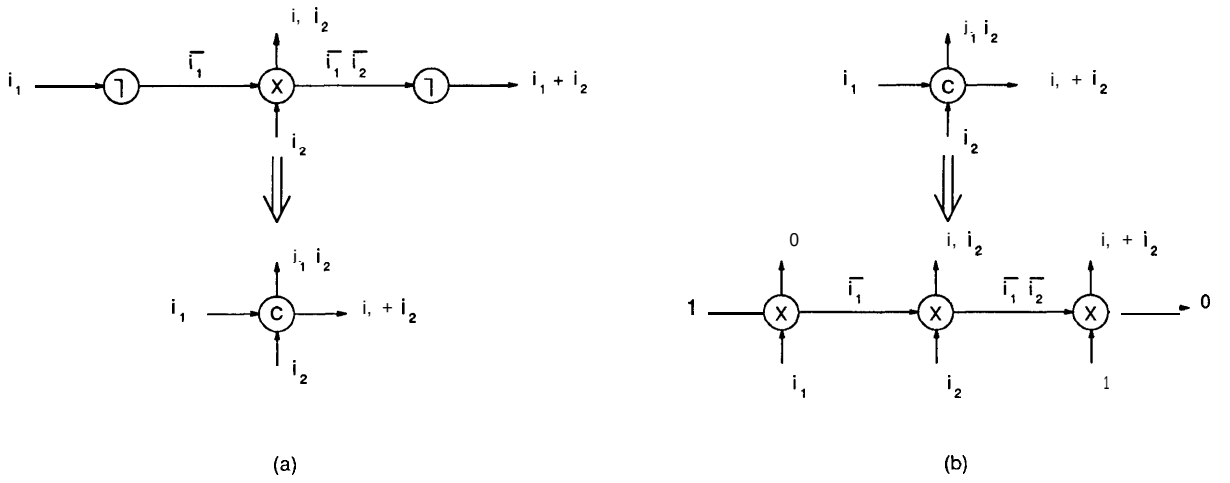


Figure 2: Interchanging Comparators and X-gates

6.2 Applications

In this section, we give linear-time algorithms for some problems related to Stable Marriage. For convenience of presentation, we phrase the following discussion partly in terms of bipartitionable X-networks and partly in terms of comparators.

6.2.1 Existence of stable matchings

The standard way to show that every instance of Stable Marriage has a stable matching is to give an algorithm, the proposal algorithm of [7], that finds the *man-optimal* stable matching in linear-time. It is nice to be able to give a proof of the existence of stable matchings that does not involve an algorithm. Here is such a proof: An instance of Stable Marriage has a stable matching if and only if the associated comparator network has a stable configuration (on the appropriate input assignment). Comparator gates are monotone; monotone networks always have stable configurations.

6.2.2 Finding a stable matching

Finding a stable configuration of a monotone network is easy: eliminate the inputs, then assign all remaining edges the value 0. This stable configuration is the 'most-O' stable configuration Q^0 of the network (on the given input assignment), because every edge that is assigned the value 1 in this stable configuration must have the value 1 in every stable configuration. We show that this strategy yields the man-optimal stable matching found by the proposal algorithm of [7].

Theorem 12 *Let I be an instance of Stable Marriage; let [N, s_{in}] be the associated instance of Comparator Network Stability. Then the 'most-0' stable configuration Q^0 of $[N, s_{in}]$ yields the man-optimal stable matching of I.*

Proof: Let $\tilde{N}$ be the X-network corresponding to *I.* (Thus, N is obtained from $\tilde{N}$ by applying the first transformation of Section 6.1.) The configuration of $[\tilde{N}, 1]$ that corresponds to Q^0 has the

following property: it assigns each edge on a red snake the value 1 only if it must be 1 in every stable configuration of $[\tilde{N}, \mathbf{1}]$; likewise, each edge on a blue snake is 0 only if it must be 0 in every stable configuration of $[\tilde{N}, \mathbf{1}]$. The implication for the Stable Marriage instance is that the stable matching found by the above procedure assigns every man the best possible mate he could get in any stable matching; similarly, every woman gets the worst possible mate she could get in any stable matching. cl

It is easy to check that eliminating the inputs of N and then assigning the remaining edges the value 1 will yield the woman-optimal stable matching.

We may think of any network stability problem as a fixpoint problem: find a configuration that is a fixpoint of the network. We believe Theorem 12 makes it easier to understand why the various possible serializations of the proposal algorithm of [7] yield the same final result: all the serializations compute the least fixpoint of the same monotone system of equations, though they may reach this fixpoint in different ways.

6.2.3 Finding other stable matchings

The following lemma is useful in giving a randomized algorithm for finding a stable matching:

Lemma 13 *Let N be an input-free network over a monotone scatter-free basis; let e be an edge of N. Then* $type_N(e) = \{0, 1\}$.

Proof: The 'all-o' and 'all-l' configurations of $[N, \mathbf{e}]$ are stable; this means that e takes both values in stable configurations of [N, e]. The result follows from Theorem 7. cl

Let N be the comparator network corresponding to an instance of Stable Marriage; let N' be the network left after input elimination. By Lemma 13, each edge of N' has type $\{0, 1\}$; thus, every edge-value pair leads to a successful propagation. This means that we may, in steps **2** and 3 of Algorithm 1, pick the next edge-value pair arbitrarily, and still retain the guarantee of a successful propagation; moreover, the algorithm still runs in linear time. This is an interesting way to pick an arbitrary stable matching out of all the different possible stable matchings. Of course, this randomized algorithm is likely to find certain stable matchings more often than others; there is **no** *a priori* reason to expect the distribution of stable matchings produced by this algorithm to be uniform or nice in other respects.

6.3 The lattice theorem for stable configurations of comparator networks

We show that the stable configurations of comparator networks form a distributive lattice. Conway's lattice theorem for Stable Marriage follows directly. For an introduction to the theory of lattices, see [1].

Let N be any network. The set of all configurations of N has the structure of the boolean lattice $\mathbf{2}^m$ under the natural ordering. (Here, *m* is the number of edges of N.) It is an easy exercise in lattice theory to show that any subset of $\mathbf{2}^m$ that is closed under the operations of bitwise AND and bitwise *OR* forms a distributive sublattice of $\mathbf{2}^m$ under the following definition of *meet* and *join:* the *meet* of two elements is their bitwise *AND* and the *join* of two elements is their bitwise

OR. We show that the set of stable configurations of a comparator network is closed under bitwise *AND* and *OR;* it follows that they form a distributive sublattice of the boolean lattice of all stable configurations.

We need the following lemma:

Lemma 14 *Let s_{in} be an input assignment to a comparator network N. Let e_1 and e_2 be two edges that are inputs to the same gate g* ***of*** *N. Then there cannot be two stable configurations Q_α and Q_β* ***of*** *$[N, s_{in}]$, such that $e_1 \leftarrow 0$ and $e_2 \leftarrow 1$ in Q_α, and $e_1 \leftarrow 1$ and $e_2 \leftarrow 0$ in Qp.*

Proof: Assume the contrary, i.e., suppose $e_1 \leftarrow 0$ and $e_2 \leftarrow 1$ in Q_α, and $e_1 \leftarrow 1$ and $e_2 \leftarrow 0$ in Q_β. Eliminate the inputs of N. This process will not assign a value to either e_1 or e_2, because, by hypothesis, these edges take on both possible values in the stable configurations of [N, s_{in}]. Hence the network left after input elimination will also have two stable configurations Q'_α and Q'_β such that $e_1 \leftarrow 0$ and $e_2 \leftarrow 1$ in Q'_α, and $e_1 \leftarrow 1$ and $e_2 \leftarrow 0$ in Q'_β. Hence it suffices to prove Lemma 14 for input-free networks. From now on, we assume s_{in} = e.

Let es be the 'min' output edge of g. Since es $\leftarrow$ 0 in Q_α and Q_β, [es, 0] is viable in N. Hence the propagation of [es, 0] must assign the value 0 to at least one of e_1 and e_2. Assume that e_1 is assigned the value 0. This means that e_1 is assigned the value 0 in every stable configuration of $[N, s_{in}]$ in which e_3 is assigned the value 0. This contradicts the stability of Q_β. Similarly, if e_2 is assigned the value 0, the stability of Q_α is contradicted. □

Theorem 15 *Let N be a comparator network, and let s_{in} be an input assignment to N. Then the stable configurations* ***of*** **[N,** *s_{in}]* ***form*** *a distributive sublattice* ***of*** *the lattice* ***of*** *all configurations, under the following definition* ***of*** *meet and join: the meet* ***of*** *two configurations is their bitwise AND; the join* ***of*** *two configurations is their bitwise OR.*

Proof: By previous remarks, it suffices to prove that the set of stable configurations is closed under bitwise *AND* and bitwise *OR.*

Let Q_α and Q_β be two stable configurations of $[N, s_{in}]$. Let Q be the bitwise *AND* of Q_α and Q_β. The values assigned by Q_α, Q_β, and Q to any particular input edge of N are the same, so Q trivially satisfies the input conditions. Hence it suffices to show that Q satisfies the gate equations at every gate of N. Let g be an arbitrary gate of N; let e_1 and e_2 be its two input edges. The following simple case analysis shows that the gate equations are satisfied at gate g.

The first case is where one of Q_α and Q_β assigns both input edges the value 0. If this happens, the input and output edges of g are all assigned the value 0 in Q, and so the gate equations are satisfied. The second case is where one of Q_α and Q_β, say Q_α, assigns both input edges the value 1. In this case, the input and output edges of g are assigned the same values in Q and in Q_β, so the gate equations are satisfied. In the third case, Q_α and Q_β both assign the same values to the two input edges. In this case, the input and output edges of g are assigned the same values in Q, in Q_α, and in Q_β, so again the gate equations are satisfied. The only remaining case is when one of Q_α and Q_β assigns the input edges the values $e_1 \leftarrow 0$, $e_2 \leftarrow 1$ and the other assigns the input edges the values $e_1 \leftarrow 1$, $e_2 \leftarrow 0$. But Lemma 14 assures us that this is not possible.

The case of the bitwise *OR* of Q_α and Q_β is analogous. cl

Remarks: Tarski's theorem [28] says that the set of fixpoints of any monotone function from a complete lattice to itself forms a complete lattice. This theorem, together with Lemma 1, is sufficient to prove that the stable configurations (on a given input assignment) of any monotone scatter-free network form a lattice. However this approach does not seem to yield the sublattice property. Nevertheless, and even though our proof is specific to the comparator, Theorem 15 does extend to any monotone scatter-free network. An easy application of the ideas in [5] yields the desired proof.

Interpreting Lemma 14 in the context of Stable Marriage yields the following theorem proven in [16]:

Lemma 16 *Let I be an instance of Stable Marriage. Suppose man m is matched to woman w in some stable matching M* of *I. Then there is no stable matching of I in which both m and w get better mates than in M.*

Proof: Consider the comparator network corresponding to instance I and apply Lemma 14. □

An easy corollary of Theorem 15 is the Lattice Theorem for Stable Marriage, proven and attributed to Conway in [20]:

Theorem 17 *The stable matchings of a given instance of Stable Marriage form a distributive lattice,* under *the following definition of 'meet' and 'join': The meet of two matchings M_1 and M_2 is a matching in which every man is married to the better of his two mates in M_1 and M_2; the join of M_1 and M_2 is a matching in which every man is married to the worse of his two mates in M_1 and M_2. (A man who is unmatched in M_1 and M_2 stays unmatched in $meet(M_1, M_2)$ and $join(M_1, M_2)$.)*

Remark: By Theorem 8, it is not possible for a man to be matched in exactly one of the two mat chings.

7 Implications for parallel complexity

7.1 Complexity theory background

For an introduction to complexity theory, see [26, 13]; for an introduction to parallel computation, see [3, 19].

The reductions in this paper are the many-one logspace reductions of [18]. We believe our reductions are simple enough that they will continue to hold under reasonable alternative definitions of reducibility. A problem is complete for a class if it is in the class and every problem in the class reduces to it. The class CC is the class of decision problems that are equivalent to Comparator Circuit Value; see [21].

7.2 Overview of reductions

In the following sections, we show that several problems related to Stable Matching are CC-complete. Section 7.3 shows that Stable Matching and Network Stability for X-networks are essentially the same problem, and that Stable Marriage and Network Stability for Comparators are essentially the

same problem. Thus, determining whether an X-network has a stable configuration is just as hard as determining whether an instance of Stable Roommates has a stable assignment; constructing a stable configuration of an X-network is just as hard as constructing a stable roommate assignment for an instance of Stable Roommates. Similar remarks apply to Stable Marriage and Network Stability for Comparators. Of course, in this case the existence questions are trivial, because the answer is always 'Yes'.

Section 7.5 shows that the existence of stable configurations in X-networks is equivalent to Comparator Circuit Value. In Section 7.6, it is shown that constructing a stable marriage for an instance of Stable Marriage is equivalent to Comparator Circuit Value: given an oracle for C-CV, we can compute the man-optimal stable matching; in the other direction, C-CV reduces to the problem of telling whether man m is married to woman w in an instance of Stable Marriage that has exactly one stable matching. This leaves the problem of constructing a stable configuration of an X-network in the non-bipartitionable case; our methods do not tell us whether this problem reduces to C-CV.

Section 7.6 also shows the following problems to be equivalent to C-CV: the Fixed Pair and Stable Pair problems, and the decision version of the Minimum-regret problem. For these problems, it does not matter whether the instance is one of Stable Matching or Stable Marriage. For the Minimum Regret problem, there is a difference between three and two: determining whether the largest regret is at most three is equivalent to C-CV, but telling whether the largest regret is two seems easier: at least it is no harder than a 2-SAT computation. (Of course, at this point, we have no proof that C-CV is harder than 2-SAT.)

Section 7.6 also shows that constructing a minimum regret stable matching for an instance of Stable Marriage is just as hard as C-CV. The proof indicates that constructing a minimum regret stable matching for instances of Stable Matching reduces to C-CV if and only if constructing a stable configuration of an X-network reduces to C-CV. However, we do not know how to directly reduce the construction of a minimum-regret stable matching to the construction of a stable configuration of an X-network.

Feder [5] has recently shown that (the decision problem corresponding to) constructing a stable configuration of an X-network is no harder (for parallel computation) than determining its existence. Given this theorem, it follows that all the problems considered above (except regret-2 stable matching and 2-SAT) are equivalent to Comparator Circuit Value, or in other words, are CC-complete.

Finally, in Section 7.7, we give a reduction from C-CV to the Assignment problem (also called Bipartite Weighted Matching). This implies a fast parallel transformation of instances of Stable Marriage into instances of the Assignment problem in such a way that the solution of the Assignment problem yields the man-optimal stable matching of the Stable Marriage instance; this partially answers a question posed in [20]. However, our reduction is not entirely satisfactory, because it 'loses' all the stable matchings except the man-optimal one. It is an open problem to give a fast parallel transformation from Stable Marriage to the Assignment problem in a manner that preserves the structure of all solutions.

7.3 Stable Matching is the same as X-Network Stability

In this section, we show that Stable Matching and X-Network Stability are essentially the same problem. There are easy reductions between the two problems that map instances of one problem to

instances of the other in a manner that preserves the structure of solutions. These reductions take Stable Marriage instances into instances of Network Stability for bipartitionable X-networks. We have already seen that bipartitionable X-networks and comparator networks are interchangeable. It follows that Stable Marriage and Network Stability for Comparators are essentially the same problem.

Theorem *18 Stable Matching is equivalent to X-Network Stability.*

Proof: **The** reduction from Stable Matching to X-Network Stability follows immediately from the reduction of Section 3. We show how to reduce X-Network Stability to Stable Matching. Given an X-network N and its input assignment s_{in}, we construct an auxiliary X-network $\tilde{N}$, in such a way that the stable configurations of [N, s_{in}] are in one-to-one correspondence with the stable configurations of $[\tilde{N}, 1]$. The new network $\tilde{N}$ has the following properties: every snake is acyclic; no two snakes meet more than once; no snake meets itself.

The first step is to delete snakes whose input edges are assigned the value 0 by s_{in}. Observe that if an input edge of a snake is 0, every edge on this snake will take the value 0 in every stable configuration; furthermore, every snake that intersects this snake will have the same value after the intersection as it did before it. Hence, such O-snakes may be identified and removed without altering the set of stable configurations.

The next step is to insert two NOT gates in series on every non-input edge of N. We then replace each NOT gate by an X-gate with one input fixed at the value 1. The resulting network is the desired network $\tilde{N}$. It is easy to check that it has the properties claimed above.

Finally, we can read off the desired instance *I* of Stable Matching from the network $\tilde{N}$ by 'inverting' the reduction of Section 3, in the following manner. Introduce a person for each snake of $\tilde{N}$. Two persons find each other acceptable if and only if their snakes meet. Finally, the order in which a person's snake meets the other snakes gives his preference ordering. It follows from Theorem 3 that the stable configurations of $[N, s_{in}]$ are in one-to-one correspondence with the stable matchings of instance *I.* *cl*

Remarks: The above proof actually proves a little more. First, if N **is** bipartitionable, so is $\tilde{N}$, and then instance *I* will be an instance of Stable Marriage. Second, the sizes of $\tilde{N}$ and *I* are linear in the size of N. Finally, note that in $\tilde{N}$, each snake meets at most three other snakes, and furthermore, if a snake meets more than one other snake, the value on its output edge is 0. This means that in instance *I,* each person has at most three persons on his preference list. Also, in every stable matching of I, the persons who list more than one other person are guaranteed to be matched to one of their first three choices. We shall use this later, to prove Theorem 32.

We give an easy proof, based on the network formulation of Stable Matching, of the following well- known lemma:

Lemma 19 *Complete Stable Matching is no easier than Stable Matching. Complete Stable Marriage is no easier than Stable Marriage.*

Proof: Given an instance of Stable Matching, we convert it to an instance of Complete Stable Matching, without affecting the set of stable matchings. The first step is to introduce the appropriate number of extra persons, and decree that they have no acceptable mates. This does not affect

the set of stable matchings. In terms of X-networks, introducing these extra persons amounts to adding new snakes with input 1 as separate connected components.

It is instructive to do the rest of the proof with networks instead of preference lists. Let N_1 be the X-network corresponding to the Stable Matching instance we have so far. Network N_1 has one deficiency: each snake of N_1 might not meet every other snake that it should meet. We remedy this by a 'cascade' construction, i.e., by appending a new network N_2 at the 'back' of N_1. The outputs of N_1 become the inputs of N_2. Our final network N will be the composition of these two networks.

Let us examine what properties N_2 should have. First, each snake of N_2 must meet a prescribed set of the other snakes, each exactly once. Second, the sets of stable configurations of [N, **1**] and $[N_1, \mathbf{1}]$ should be isomorphic. The second condition will be met if the following condition is met: whenever s is the output word of some stable configuration of $[N_1, \mathbf{1}]$, $[N_2, s]$ has exactly one stable configuration. A simple solution is to make N_2 a circuit. This works because every circuit has exactly one stable configuration on any given input assignment.

We construct N_2 as follows: Pick any fixed total order O of the snakes, and let each snake meet all the snakes it ought to in the order induced by 0. This defines N_2, and hence N. The desired instance of Stable Matching can be read off from N. This completes the reduction. The proof for Marriage instances is similar.

Notice that the cascade construction actually does something very simple with the preference lists: it enlarges the preference list of each person x, by appending to it all the mates originally unacceptable to x, in the order given by 0. cl

7.4 On reducing problems to Comparator Circuit Value

Following [18], we say that a problem Z is *many-one* reducible to C-CV if we can exhibit a reduction from Z to C-CV that takes each instance of Z to a comparator circuit plus an input assignment to it, so that the instance has a 'Yes' answer if and only if the circuit has output 1 when evaluated on the given input assignment. In proofs, it is more convenient to think in terms of Turing reductions to C-CV. The following lemma helps convert Turing reductions to many-one reductions:

Lemma 20 *If Z_1 and Z_2 are many-one reducible to C-CV, so are $OR(Z_1,Z_2)$, $AND(Z_1,Z_2)$, and $NOT(Z_1)$.*

Proof: The many-one reducibility of *OR*(Z_1,Z_2) and $AND(Z_1,Z_2)$ to C-CV follows from the fact that the comparator can simulate the *OR* gate and the *AND* gate. The many-one reducibility of $NOT(Z_1)$ to c-cv follows because the X-gate can simulate $\{C,NOT\}$, and because X-CV reduces to C-CV by Lemma 21 below. II

Lemma 21 *X-CV is equivalent to C-CV.*

Proof: The basis {X} can simulate {C}; hence C-CV reduces to X-CV. The reduction in the other direction utilizes Goldschlager's idea of using 'double-rail' logic [9]. In other words, encode the values 0 and 1 by the pairs (1,0) and $(0,1)$ respectively. Replace every edge e in the original circuit by a pair of edges (e^-, e^+); the rest of the construction will guarantee that the two edges in a pair will always have complementary values. If e is an input edge that is assigned the value v, edges e^- and e^+ are assigned the values $\overline{v}$ and v respectively. Finally, we must replace every X-gate

by an appropriate gate. The X-gate has two inputs ii, i_2 and two outputs $o_1 = i_1\ \overline{i_2}$, $o_2 = \overline{i_1}\, i_2$. The new gate will have four inputs i_1^-, i_1^+, i_2^-, i_2^+ and four outputs o_1^-, o_1^+, o_2^-, o_2^+. The new gate must simulate the behaviour of the X-gate in the following sense: when presented with an input word that is a valid encoding of the input word to an X-gate, it must produce an output word that is the valid encoding of the corresponding output word of the X-gate. The action of the gate on input words like 0010 may be arbitrarily chosen. One solution is to use the gate defined by the equations $o_1^- = i_1^- + i_2^{*+}, o_1^+ = i_1^+ i_2^-, o_2^- = i_1^+ + i_2^-, o_2^+ = i_1^-\ i_2^+$. It is easily seen that this gate can be simulated by the comparator.

An easy inductive proof shows that if edge e in the original circuit has value v, then the pair of edges (e^-, e^+) in the new circuit will be assigned the value-pair $(\overline{v}, v)$. Hence, in particular, if e_0 is the output edge of the original circuit, the edge e_0^+ will serve as the output edge of the new circuit. Finally, since the new gate can be simulated by the comparator, the new circuit can be converted into a comparator circuit. □

7.5 X-Network Stability is equivalent to Comparator Circuit Value

Let Ω be a scatter-free basis that can simulate *{NAND}*. The objective of this section is to show that Ω-NS and Ω-CV are equivalent. (This result was presented in [21]; here we give a complete proof.) In particular, X-NS is equivalent to X-CV; hence, using Lemma 21, X-NS is equivalent to c-cv.

7.5.1 Reducing Ω-CV to Ω-NS

The proof uses the idea of a *forcer*. A w-forcer is a one-input, zero-output network which has a stable configuration if and only if the input takes the value v. A O-forcer may be built by tying the output of a *NAND* gate to one of its inputs, and a l-forcer may be obtained by adding a NOT gate at the input of a O-forcer.

Lemma 22 *Let Ω be any basis that can simulate (NAND). Then Ω-CV reduces to Ω-NS.*

Proof: To determine the output of a circuit over Ω, feed its output into a l-forcer. The resulting network has a stable configuration if and only if the circuit has output 1. Since basis Ω can simulate *{NAND},* we can build forcers over Ω. II

7.5.2 Reducing Ω-NS to Ω-CV

Given a network N over a scatter-free basis Ω and an input assignment s_{in} to N, we wish to construct a circuit over Ω so that the output of the circuit is 1 if and only if [N, s_{in}] has a stable configuration.

Let N' be the network left after the inputs of N are eliminated. We say that an edge of N is *bad* if it is a contradicting edge of N'. By Theorem 6, network N has no stable configuration if and only if it has a bad edge. Thus the key step is to construct a circuit that will tell us if a given edge is bad. Once this is done (independently) for every edge of N, we may combine the results and answer the stability question by using some simple additional circuitry. The rest of the discussion is divided into three stages. In the first stage, we show how to tell if an edge is bad; in the second stage, we define a network process that will compute whether or not an edge is bad; in the final stage, we convert the network process into a circuit computation.

Stage 1: How do we **tell** if edge e is bad? Break edge e, thus creating an extra input *e(in)* and an extra output *e(out)*. Call the resulting network N_e. Let s_e be the input assignment to N_e obtained by augmenting s_{in} with the assignment *e(in)* $\leftarrow v$. Consider the following experiment *Expt(e, v)*: Assign each input of N_e the value it gets under the input assignment s_e, and perform input elimination. Since the gates are scatter-free, each output of N_e will be assigned a value. We say that experiment *Expt(e, v)* succeeds if the value assigned to edge *e(out)* is v; otherwise it fails.

Lemma **23** *Edge e is bad if and only if both experiments Expt(e, 0) and Expt(e,* 1) *fail.*

Proof: The key observation is that the result of input elimination does not depend on the order in which the inputs are eliminated. Hence we may pretend that experiment $Expt(e, v)$ is conducted in two phases: in the first phase, all the inputs except *e(in)* are eliminated; in the second phase, input $e(in)$ is eliminated. There are two cases, depending on whether or not edge *e(out)* is assigned a value in the first phase.

Case 1: Suppose edge *e(out)* is assigned a value v' in the first phase. This means that edge e would be assigned the value v' when the inputs of N are eliminated. It follows that e is not an edge of N'; hence e is not a bad edge of N. Notice also that *Expt(e,* v') will succeed. Thus Lemma 23 is true in this case.

Case *2:* Suppose edge *e(out)* is not assigned a value in the first phase. This means two things: first, the effect of the first phase is identical to that obtained by eliminating the inputs of N; second, edge e is an edge of N'. Notice that the second phase of $Expt(e, v)$ will assign value v to edge *e(out)* if and only if [e, v] is viable in N'. It follows that both *Expt(e,O)* and *Expt(e,* **1)** fail if and only if edge e is contradicting in N'. Hence Lemma 23 is true in this case as well. cl

Stage 2: We now define a network process that will compute the value assigned to edge *e(out)* by experiment *Expt(e, v)*. Consider the following network process: Initialize network N_e to any configuration consistent with the input assignment s_e. Then let the network 'run'. In other words, perform the following primitive operation at each gate of the network, in parallel: apply the gate-function to the values on the input edges, and use the result to update the values on the output edges of the gate. The crucial observation is that the values on certain edges of the network will stabilize, i.e., will not change after a certain number of parallel steps. In particular, the value on edge *e(out)* will stabilize at the value it is assigned in experiment $Expt(e, v)$.

Lemma **24** *Each edge that is assigned a value in experiment Expt(e, v) will stabilize at the same value in the network process.*

Proof: Notice that the value on each input edge of network N_e is held constant. Hence Lemma 24 is true for these edges. The process of eliminating the inputs of N_e in experiment $Expt(e, v)$ gradually assigns values to some edges of N_e. An assignment to an output edge of gate g follows from an assignment to a subset of the inputs of g by invoking the equation of g. This process is faithfully mimicked in the network process: first each edge in the relevant subset of the inputs stabilizes to the correct value; then the output edge stabilizes. cl

The proof of Lemma *24* may be extended to bound the number of parallel steps needed for edge *e(out)* to stabilize. Let m be the number of edges in network N. Then the longest chain of influence in the input-elimination process of experiment *Expt(e,* v) involves at most $m + 1$ edges. Hence edge *e(out)* will reach its correct value after at most m parallel steps.

Stage 3: The final step is to use the regularity of the network process to 'unravel' it into a circuit computation. In other words, we construct a circuit, over Ω that will simulate the network process. The idea is to make *m* copies of network N_e (we call each copy a 'layer'), and connect these layers up into **a** circuit C in such a way that the values that leave the t-th layer are the values that appear in network N_e after parallel step t. As a result, the correct value of edge *e(out)* may be read off from the top layer of the circuit.

More accurately, we make *m* copies of network N_e, and change the output edges of all the gates so that they enter the next layer, instead of the same layer. The gates of C are $\{(g, t) \mid g$ is a gate of N_e; $1 \leq t \leq m\}$; the edges are $\{(\tilde{e},t) \mid \tilde{e}$ is an edge of N_e; $0 \leq t \leq m\}$. (Not quite! The edges of the form $(\tilde{e},0)$ for $\tilde{e}$ an output edge of N_e, or of the form $(\tilde{e}, m)$ for $\tilde{e}$ an input edge of N_e, are not actually present.) The gates $\{(g, t) \mid 1 \leq t \leq m\}$ all have the same gate type as g. If g has inputs $i_1 \ldots i_\lambda$ and outputs $o_1 \ldots o_\mu$, then (g, t) has inputs $(i_1, t-1). \ldots (i_\lambda, t-1)$ and outputs $(o_1,t)\ldots(o_\mu,t)$.

If $\tilde{e}$ is an input edge of N_e with value $\tilde{v}$, all the input edges $\{(\tilde{e}, t) \mid 0 \leq t < m\}$ get the value $\tilde{v}$ in C; this has the effect of presenting the real inputs of N_e to each layer of the circuit. The edges $(\tilde{e}, 0)$, for $\tilde{e}$ an internal edge of N,, are assigned arbitrary values; this reflects the arbitrariness in the initial configuration of N_e. Finally, if $\tilde{e}$ is a non-input edge of N_e, edge $(\tilde{e}, m)$ is an output edge of C. This completes the description of C.

The following lemma may be proven by an easy induction on t:

Lemma 25 *The value on edge $(\tilde{e},t)$ of C equals the value on edge $\tilde{e}$ of N_e after parallel step t.*

In particular, the value assigned to edge *e(out)* in experiment $Expt(e,v)$ is given by the value assigned to edge (e(*out), m*)in C. Once we have the values assigned to edge *e(out)* in experiments *Expt(e,* 0) and *Expt(e,* 1), we may determine whether or not edge e is bad by adding an *AND* gate and a NOT gate. We may then combine the results for each edge and answer the stability question by adding *OR* gates and a NOT gate. Thus we get, a circuit over $\Omega \cup \{NAND\}$ that computes the answer to the question 'Does [N, s_{in}] have a stable configuration?' This circuit has $O(m^3)$ gates.

Theorem 26 *Let Ω be any scatter-free basis that can simulate (NAND). Then Ω-NS is equivalent to Ω-CV.*

Proof: The reduction from Ω-NS to Ω-CV follows from the construction above; the reduction from Ω-CV to Ω-NS follows from Lemma 22. □

7.6 More Stable Matching problems equivalent to Comparator Circuit Value

In this section, we prove that several Stable Matching problems are CC-complete. We begin by showing (Lemma 27 below) that X-network stability problems remain just as hard even if we place certain constraints on the values that certain edges can take.

Lemma 27 *The following problems are equivalent (and hence equivalent to C-CV):*

1. Given an X-network N, an input assignment s_{in} , and a partial assignment P to the non-input edges of N, can P be completed to a stable configuration of the network?

2. Given an X-network N, and an assignment s_{in} to the input edges of N, is there a stable configuration of [N, s_{in}]?

Proof: Problem 2 is clearly a special case of problem 1, so it suffices to reduce problem 1 to problem 2. This can be done by using forcers. Suppose P requires that edge e get the value v. Break edge e, creating input $e(in)$ and output $e(out)$. Assign edge $e(in)$ the value v, and feed $e(out)$ into a v-forcer. cl

Theorem 28 *The following problems reduce to C-CV:*

1. Fixed pair: Is the given pair of persons a fixed pair of the given instance of Stable Matching, i.e., is it true that these two persons are matched to each other in every stable matching?
2. Stable pair: Is the given pair of persons a stable pair, i.e., is it true that there is a stable matching that matches these two people to each other?
3. Minimum-regret: Is it true that there is a stable matching of the given instance in which every person has regret at most k?

Proof: To check that two persons form a fixed pair, it suffices to check that the snakes corresponding to these two persons always meet each other with the value 1 in the corresponding instance of X-Network Stability. Let g be the gate where the two snakes meet; we have to check that there is no stable configuration that assigns either edge entering g the value 0. This reduces to C-CV by Lemma 27 and Lemma 20.

To check that two persons form a stable pair, it suffices to check that there is a stable configuration that matches these two persons to each other, i.e., a stable configuration in which the snakes corresponding to these two people meet each other with the value 1. This reduces to C-CV by Lemma 27.

To check that there is a stable matching in which every person has regret at most k, just check whether there is a stable configuration that assigns each snake the value 0 after it has traversed k gates. (If snake q has fewer than k gates on it, the desired stable configuration must assign the output edge of q the value 0.) Again, we invoke Lemma 27 to reduce this problem to C-CV. □

Theorem 29 *The Man-optimal Stable Marriage problem "Given an instance I of Stable Marriage, a man m, and a woman w, determine whether m is matched to w in the man-optimal stable matching M^0 of I" reduces to C-CV.*

Proof: Let [N, s;,] be the instance of Comparator Network Stability corresponding to instance *I;* let Q^0 be the stable configuration of $[N, s_{in}]$ corresponding to the man-optimal stable matching of *I.* We show how to construct a comparator circuit to compute Q^0. The proof of Theorem 29 then follows by applying Lemma 20.

Theorem 12 tells us that Q^0 is the 'most-O' stable configuration of [N, s_{in}]. We claim the following network process finds Q^0. Initialize N to the 'all-O' stable configuration. Then make all the red inputs 1. These changes create disturbances that propagate through the network. But the network will eventually enter the stable configuration Q^0.

We justify the claim. Notice that all the input disturbances 'increase' the value assigned to an input edge. The comparator is a monotone gate: increasing the values assigned to the inputs

can never cause an output to decrease. Hence, the effect of the disturbances as they move through the network will be to increase the value of some edges of N, but no edge will ever move from 1 to 0. It follows that the disturbances will die out in at most *m* 'steps'. Now observe that an edge is assigned the value **1** during the network process only if it must have the value 1 in every stable configuration of $[N, s_{in}]$. It follows that the stable configuration that the network reaches is Q^0.

The network computation on N may be unravelled into a circuit computation over the same basis in the standard fashion. (See, for example, the discussion in Section 7.5.) □

We can also reduce the problem of constructing a minimum-regret stable matching for an instance of Stable Marriage to C-CV. Let *I* be any instance of Stable Marriage; let *regret(I)* be the value of the largest regret in a minimum-regret stable matching of *I.* The discussion in Section **4.2** tells us that there is a single instance $I_{regret(I)}$ of Stable Marriage whose stable matchings are all the minimum-regret stable matchings of *I.* Define the man-optimal minimum-regret stable matching of *I* to be the man-optimal stable matching of $I_{regret(I)}$. The following theorem shows that the construction of this special stable matching reduces to C-CV:

Theorem 30 *The Man-optimal Minimum-regret Stable Marriage problem "Given an instance I of Stable Marriage, a man m and a woman w, determine whether m is matched to w in the man-optimal minimum-regret stable matching of I'" reduces to C-CV.*

Proof: Assume without loss of generality that there are at least as many men as women in instance *I;* let *n* be the number of men. Then *regret(I)* is one of the integers between 1 and *(n* + 1). If *regret(I)* = *(n* + 1), this is taken to mean that there is some person who is unmatched in every stable matching of *I.* The idea is to compute the man-optimal stable matching separately in each of the *(n* + 1) cases, and then select the correct stable matching using the value of *regret(I).*

Suppose we wish to compute whether m is matched to w in the matching that is best for the men from among all the stable matchings in which each person has regret at most k. First compute the appropriate instance I_k of Stable Marriage as in Section 4.2. The proof of Theorem 29 can be made to yield a comparator circuit C_k whose output is 1 if and **only** if man *m* is matched to woman w in the man-optimal stable matching of instance I_k.

The reduction from Minimum-regret to C-CV (see Theorem 28 above) can be made to yield, for each $k \in [1, (n$ t $1)]$, a comparator circuit $Atmost_k$ whose output is 1 if and only if *regret(I)* $\leq$ *k.* (The circuit $Atmost_{n+1}$ is trivial: its output is always 1.) Now build a comparator circuit $Exact_k$, where the output of $Exact_k$ is **1** if and only if *regret(I)* = k. For *k* > **1,** *regret(I)* = *k* if and only if $Atmost_k$ has output 1 but $Atmost_{k-1}$ has output 0; Lemma 20 then yields the comparator circuit $Exact_k$. Also, $Exact_1$ is just $Atmost_1$. (Of course, for these constructions, we need two copies of the circuits *Atmostk.)*

The circuit $\overset{OR}{k}(AND(Exact_k, C_k))$ has output 1 if and only if man *m* is matched to woman w in the man-optimal minimum-regret stable matching of I. This circuit can be built with comparators: just simulate the AND gate and the OR gate with comparators. cl

The problems reduced to C-CV in the previous three theorems are in fact equivalent to C-CV:

Theorem 31 *C-CV reduces to the Fixed Pair problem, the Stable Pair problem, the Man-optimal Stable Marriage problem, and the Man-optimal Minimum-regret Stable Marriage problem.*

Proof: Let C be the given comparator circuit. Use the transformation of Figure 2(b) to replace it by a bipartitionable X-circuit. Delete snakes whose input edges have the value 0, as in the proof of Theorem 18. Circuit C has output 1 (on a given input assignment s_{in}) if and only if the output edge of a certain vertical snake q of the X-circuit is assigned the value 1. Add a new horizontal snake q' with input 1 to the X-circuit. Let q' meet q and no other snake. Let q meet q' last. Then C has output 1 on s_{in} if and only if q and q' meet each other with value 1 in the new X-circuit. The method used in the proof of Lemma 19 can be used to ensure that the number of horizontal and vertical snakes are equal, and that every horizontal snake meets every vertical snake exactly once. Now, read off an instance of Complete Stable Marriage from the X-gate circuit. This instance will have exactly one stable matching. Also, the output of C is 1 on s_{in} if and only if the persons corresponding to snakes q and q' are married to each other in the unique stable matching. The theorem follows. □

Theorem 32 *C-CV reduces to the question 'Does the given instance of Complete Stable Marriage have a stable matching in which every person has regret at most 3 ?'*

Proof: Given an instance of C-CV, we produce an instance of Complete Stable Marriage with the following properties: there is exactly one stable matching; the regret of every person except woman w in this stable matching is at most 3; woman w has regret 3 if and only if the instance of C-CV has output 1, otherwise she has regret 4.

As in the proof of Theorem 31, replace the given comparator circuit C by a bipartitionable X-circuit without O-snakes, so that C has output 1 on input assignment s_{in} if and only if the output edge of a certain vertical snake q_1 of the X-circuit is assigned the value 1.

The next step is to prepare the snake of woman w appropriately. Introduce the following new snakes: $q_0, q_2, q_3, q_4, q_5, q_7$. The input edge of each of these snakes is assigned the value 1. The snakes with odd subscripts will be vertical snakes and the snakes with even subscripts will be horizontal snakes. Snake q_0 will ultimately become the snake of woman w. Snake q_0 meets q_3, q_5, q_1, q_7 in that order; snake q_2 meets q_3 only; snake q_3 meets q_2 and q_0 in that order; snake q_4 meets q_5 only; snake q_5 meets q_4 and q_0 in that order; snake q_7 meets q_0 only. Snake q_1 meets q_0 last. The effect of this construction is to guarantee that woman w gets either her third choice or her fourth choice; she gets her third choice if and only if circuit C on input s_{in} has value 1.

The rest of the proof contains a general construction that caa be used to simultaneously reduce the regret of some or all of the participants to three and make the instance 'complete', without changing the structure of the set of all stable matchings. It introduces new persons, but each of them has regret at most three. The same construction works, with the appropriate changes, for the non-bipartitionable case. The size of the output instance produced by the construction is $O(n^4)$, where n is the size of the comparator circuit; it is possible to produce an instance of size $O(n^2)$ if the output instance is not required to be bipartitionable, but such considerations are largely irrelevant from the viewpoint of parallel reductions.

The method used in the proof of Lemma 19 converts our bipartitionable X-network into the X-network corresponding to an instance of Complete Stable Marriage. There is one difficulty, however: some of the participants other than w might have regret more than 3, or in other words, some snakes other than q_0 might have the value 1 even after meeting three other snakes. This can be rectified by performing some minor surgery on the network. The trick, much as in the proof of Theorem 18, is to insert two NOT gates in series into every non-input edge of the network except

the edges on snake q_0, and then replace each NOT gate by an X-gate with input 1. The result is a bipartitionable X-circuit with the following properties: each snake except q_0 meets at most three other snakes; the value on the input edge of every snake is 1; the value on the output edge of every snake is 0; there are an equal number of man-snakes and woman-snakes. Now convert this to an instance of Complete Stable Marriage by completing each person's list in any manner. The resulting instance will have all the desired properties. □

It follows from the following theorem and Theorem 8 that the question 'Does the given instance of Stable Matching have a stable matching in which every person has regret at most 2?' is in NC. This suggests that the number 3 in Theorem 32 cannot be reduced.

Theorem 33 *Stable Matching is in* NC *if each person lists at most two other participants as acceptable mates.*

Proof: Introduce for each person x two boolean variables $gets(x, 1)$ and $gets(x, 2)$. The intended meaning is that $gets(x, j)$ is made true by a stable matching if and only if x **is** matched to his j-th choice. Then the stable matchings are precisely the solutions of the following equations:

1. $\overline{gets(x, 1) \wedge gets(x, 2)}$.
2. If x is the j-th choice of y, and y is the k-th choice of x, then $gets(y, j) \equiv gets(x, k)$.
3. If x is the first choice of y, and y is the first choice of x, then $gets(x, 1)$.
4. If x is the first choice of y, and y is the second choice of x, then $gets(x, 1) \vee gets(x, 2)$.
5. If x is the second choice of y, and y is the second choice of x, then $gets(x, 1) \vee gets(y, 1) \vee gets(x, 2)$.

Notice that if x and y are second choices of each other, then all the equations involving $gets(x, 2)$ and $gets(y, 2)$ may be replaced by the single defining equation $gets(x, 2) \equiv gets(y, 2) \equiv \overline{gets(x, 1) \vee gets(y, 1)}$. We say that such variables $gets(x, 2)$ and $gets(y, 2)$ are *inessential. Solve* the original system of equations as follows: delete all the equations involving inessential variables; solve the resulting system of equations; then solve for the inessential variables by using their defining equations. The key observation is that eliminating the inessential variables renders the system of equations simpler: each equation in the new system mentions at most two variables. Hence the new system may be converted into an instance of 2-SAT, and can be solved in NC [4]. □

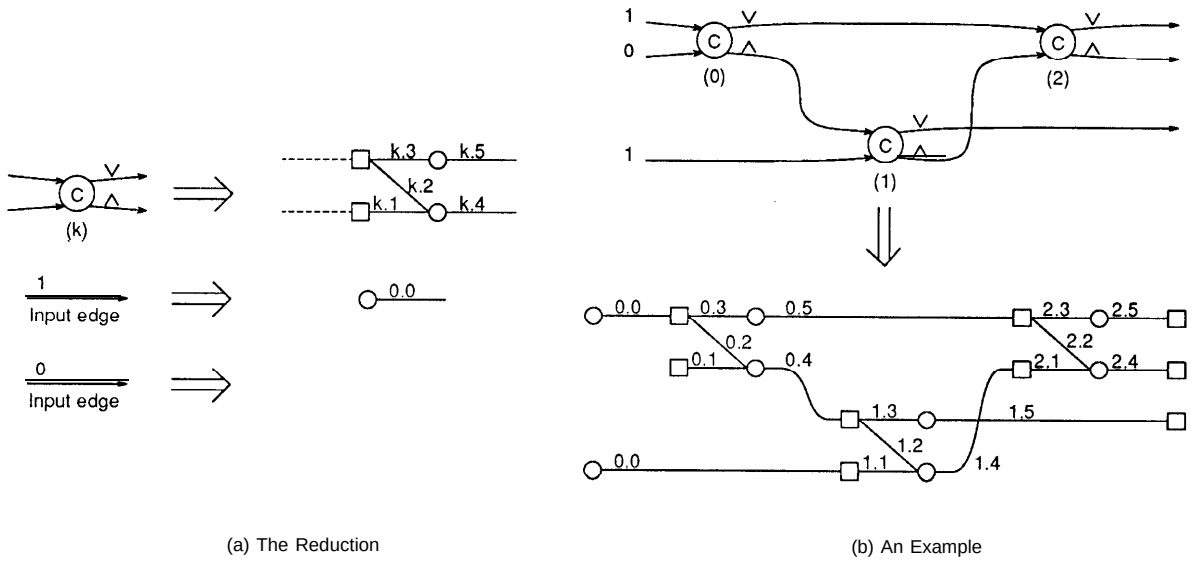
Remark: The difference between preference lists of lengths two and three has also been observed by Soroker [25]. He considers a special case of Stable Marriage, called Priority Marriage. He shows that Priority Marriage is just as hard even if each preference list is at most three long; however, if each list has at most two persons on it, the problem is in NC.

7.7 A reduction from Comparator Circuit Value to the Assignment problem

In this section, we exhibit a fast parallel reduction from C-CV to Bipartite Weighted Matching. Cook [2] initially showed that X-CV is equivalent to a problem called Lex-first Maximal Matching; also, there are easy reductions from C-CV to X-CV and from Lex-first Maximal Matching to

Weighted Matching. The actual reduction used in this section incorporates these ideas; the specific gadgets used are due to Soroker [25].

Let C be the given comparator circuit, with n gates; let s_{in} be the given input assignment to C. We shall assume that C is topologically sorted, that is, we are given a function *num* that maps the gates of C into distinct integers in $[0, \ldots, (n-1)]$, in such a way that if an output edge of g_1 is an input edge of g_2, then $num(g_1) < num(g_2)$. This assumption is justified (see [24]) because there is a fast parallel transformation that takes as input any circuit C and an input assignment s_{in}, and returns a (larger) circuit C' over the same basis, a topological-sorting function *num* for C', and an input assignment s'_{in}, so that [C', s'_{in}] has output 1 if and only if [C, s_{in}] has output 1. We shall also assume, without loss of generality, that the output whose value we wish to compute is the 'OR' output of gate $(n-1)$. We shall construct a bipartite graph G and a function *weight* from the edges of G into the positive integers, so that the maximum-weight matching of G is unique; furthermore, this matching has odd weight if and only if the output of C on the input assignment s_{in} is 1.



(a) The Reduction

(b) An Example

Figure 3: Reducing Comparator Circuit Value to Bipartite Weighted Matching

The construction is illustrated in Figure 3(a). Each comparator gate g is locally transformed into a graph gadget with four vertices and *five* edges. If $num(g) = k$, the edges of the gadget are labelled from $k.1$ to $k.5$ as shown. The weight of an edge labelled $k.l$ will be $2^{5n-(5k+l)}$. Think of the gadget as taking in two input edges (shown dotted) and producing two output edges, the edges labelled *k.4* and *k.5*. The gadgets for different gates are connected to one another in exactly the same way as the gates are connected in C.

Input edges of C are treated as follows. Suppose e is an input edge of C. If e is assigned the value 1 by s_{in}, it becomes an edge with label 0.0 (and weight 2^{5n}) in G. On the other hand, if e is assigned the value 0, it does not appear in G. An example of the transformation is shown in Figure 3(b).

It is easy to prove that G is bipartite: the shape of the vertices (square or round) in Figure 3 gives a bipartition.

Theorem 34 *The maximum-weight matching of G is unique, and has odd weight if and only* if *the value assigned to the output edge of C on input assignment s_{in} is 1.*

Proof: Say an edge is *heavy* if its weight is 2^{5n}; it is *Eight* otherwise. Every heavy edge will appear in every maximum-weight matching of G, because these edges do not share vertices, and the combined weight of all the light edges is $2^{5n} - 1$, which is less than 2^{5n}. The weights of the light edges are decreasing powers of 2, and no two light edges have the same weight. This means that the heaviest light edge weighs more than all the other light edges combined, and hence it is always advantageous to add the heaviest light edge to the current matching. Reasoning in this fashion, we conclude that the maximum-weight matching is unique, and can be found by the following greedy algorithm: Start with a matching consisting of the heavy edges, then inspect each light edge in decreasing order of weight, adding it to the current matching if at all possible.

We prove the following inductive assertion:

$(*)$ Edge e of C is assigned the value 1 in the unique stable configuration of [C, s_{in}] if and only if the corresponding edge in G is matched in the unique maximum weight matching M of G.

We first prove $(*)$ when e is an input edge. Then we consider each gate of G, in increasing order of $num(g)$; and prove that if $(*)$ is true for the edges that are inputs to g, it must also be true for the edges that are outputs of g. This will complete the proof of $(*)$

The base case is when e is an input edge of C. If e is assigned the value 0 by s_{in}, $(*)$ is vacuously true. If e is assigned the value **1** by s_{in}, the corresponding edge in G is heavy, and will be included in M.

Now assume that $(*)$ is true for the edges that are inputs to the gate g with $num(g) = k$. We wish to prove $(*)$ for the outputs of g. To do this, examine the graph gadget corresponding to gate g. See Figure 3(a). Define the boolean variables a_1 and a_2 as follows: a_1 is true if and only if the top input to g is 1; a_2 is true if and only if the bottom input to g is 1. By assumption, these variables also capture whether or not the input edges to the graph gadget are matched in M. Now notice that in the vicinity of the graph gadget, the weights decrease in the following order: the input edges (in some order); then the edges labelled $k.1$, $k.2$, $k.3$, $k.4$, $k.5$, in this order; then the other edges, if any, that are incident to the right end-points of the edges labelled *k.4* and *k.5.* This allows us to infer when (under what conditions on the variables a_1 and a_2) the greedy algorithm will match the edges labelled $k.l$. The computations in Table 2 verify that $(*)$ holds for the output edges of g.

Given $(*)$, we complete the proof of Theorem 34. All the edges of G have even weights except the edge labelled $(n-1).5$, which has weight 1. Hence M has odd weight if and only if this edge is in M, which happens if and only if the output edge of C is assigned the value 1 when C is evaluated on input assignment Sin. cl

Remark: By combining the reduction of this section with the reduction in the proof of Theorem 29, we get a reduction from the Man-optimal Stable Marriage problem to Bipartite Weighted Matching. This reduction takes an instance of Stable Marriage with *n* men and *n* women to an instance of Bipartite Weighted Matching that has $(2 + o(1))n^4$ vertices on either side of the graph, and edge weights that have $(5 + o(1))n^4$ bits each. Given the optimal matching M of the Weighted

Edge	When matched
Top input edge	a_1
Bottom input edge	a_2
Edge labelled $k.1$	$\overline{a_2}$
Edge labelled $k.2$	$\overline{a_1}\, a_2$
Edge labelled $k.3$	$a_1\, \overline{a_2}$
Edge labelled $k.4$	$a_1\, a_2$
Edge labelled $k.5$	$a_1 + a_2$

Table 2: When the edges of the graph gadget are matched

Matching instance, we can read off the man-optimal stable matching of the Stable Marriage instance.

8 Other reductions

8.1 Counting the number of stable matchings is #P-complete

We show how to use the network formulation to give a simple proof that the problem of **counting** the number of stable matchings in an instance of Stable Matching is #P-complete. A stronger result is known [16], namely that the theorem holds even if the instance is one of Stable Marriage, but the proof of the stronger result uses a non-trivial theorem about the structure of all stable matchings of an instance of Stable Marriage.

Theorem 35 *The problem of counting the number of stable configurations in an instance of X-Network Stability is #P-complete.*

Proof: Testing whether a configuration of an X-network is stable is clearly in $\mathcal{P}$; this puts the counting problem in $\#\mathcal{P}$. To show completeness for $\#\mathcal{P}$, we give a parsimonious reduction from Monotone 2-SAT. Monotone 2-SAT is shown to be #P-complete in [29].

Let I be the instance of Monotone 2-SAT. Construct X-network N as follows: it has a cyclic snake q_a for each variable a of *I;* snakes q_a and q_b intersect if and only if the clause $a \vee b$ appears in *I.* The instance of X-Network Stability is $[N, \mathrm{e}]$.

Notice that all the edges of any snake q_a are assigned the same value in any stable configuration of [N, e]. If this value is 0, this is taken to mean that the variable *a* is 'true'. It is easy to check that this gives a one-to-one correspondence between the stable configurations of [N, e] and the satisfying assignments of *I.* *cl*

8.2 Three-party Stable Marriage is $\mathcal{NP}$-complete

Knuth [20] asks about the complexity of Stable Marriage if the participants are of three kinds: men, women, and dogs. We show that one possible formulation of this problem is $\mathcal{NP}$-complete. It has been independently shown in [22] that this problem is $\mathcal{NP}$-complete.

We consider the following formulation: An instance of Three-party Stable Marriage has equal numbers of three kinds of participants, say men, women, and dogs. A matching M is a set of triplets:

each triplet contains exactly one participant of each kind, and each participant is in exactly one triplet of M. The participants have preference lists. Each participant lists all the triplets containing his name in order of preference. Matching M is unstable if there is a man *m,* a woman w, and a dog *d,* each of whom prefers the triplet *(m,* w, *d)* to the triplet he is assigned to in M. The problem is: Given an instance of Three-party Stable Marriage, determine whether there is a stable matching.

Lemma 36 *Three-party Stable Marriage is equivalent to Y-Network Stability.*

Proof: Section 3 shows that every instance of Stable Matching is an instance of X-Network Stability. In an analogous fashion, it can be shown that every instance *I* of Three-party Stable Marriage is an instance [N, 1] of Y-Network Stability. The Y-gate is a natural generalization of the X-gate. It has three inputs i_1, i_2, i_3 and three outputs o_1, o_2, o_3. The function Y : i_1, i_2, $i_3 \rightarrow o_1$, o_2, o_3 is defined as follows: Y(l,l,l) = $(0,0,0)$; on all other input words, $Y(i_1, i_2, i_3)$ = (i_1, i_2, i_3). Input i_j and output o_j are associated. This association allows us to describe Y-networks in terms of snakes, just as we did for X-networks.

Network N contains one acyclic snake per participant. Snakes meet in triples at Y-gates. The sequence of Y-gates on the snake of a participant corresponds exactly to the sequence of triples that appear on his preference list. Proceeding in much the same way as in Section 3, it can be shown that in any stable configuration of [N, 1], snakes drop in triples, and that the stable matchings of *I* are in one-to-one correspondence with the stable configurations of [N, 1]: three persons are matched in *I* if and only if their snakes drop together in [N, 1]. This completes the reduction from Three-party Stable Marriage to Y-Network Stability.

The reduction from Y-Network Stability to Three-party Stable Marriage is similar to the reductions in Section 7.3. Given a Y-network N and its input assignment s_{in}, we construct an auxiliary Y-network $\tilde{N}$, in such a way that the stable configurations of [N, s_{in}] are in one-to-one correspondence with the stable configurations of [$\tilde{N}$, 1]; furthermore, network $\tilde{N}$ has the following properties: every snake is acyclic; no two snakes meet more than once; no snake meets itself; each snake is assigned a *color* $\in \{1,2,3\}$; if an edge of snake q is input i_j or output o_j of some gate g of $\tilde{N}$, the color of snake q is j.

A snake whose input edge **is** assigned the value 0 by s_{in} may be deleted from the network, just as we did for X-networks. Next, assign each edge a *color* $\in \{1,2,3\}$ as follows: if edge e is output o_j of some gate, set $color(e)$ equal to j; if e is an input edge of the network and is input o_j to some gate, set *color(e)* equal to j. (Edges that are both input and output edges of the network may be assigned an arbitrary color.) There are two difficulties to resolve: the network may have cyclic snakes, and there may be color conflicts, i.e., an edge with color j may be input i_k to a gate, with $j \neq k$.

Notice that $Y(1,1,a)$ = $(\overline{a},\overline{a},0)$ and $Y(1,\overline{a},1)$ = $(a,0,a)$. Connect these two gates together by making the second output of the first gate be the second input of the second gate. Color the resulting circuit in the obvious fashion. The resulting circuit serves to 'change the color' of an edge from 3 to 1. (Just ignore all outputs except output o_1 of the second gate.) Simple variants of this circuit can be used to change from any color j to any color *k.* Inserting such circuits in every non-input edge of the Y-network resolves all color conflicts and makes each snake acyclic. Let the color of a snake be the common color of its edges. The resulting network is $\tilde{N}$; it is easy to verify that it has all the desired properties.

We can read off an instance of Three-Party Stable Marriage from network $\tilde{N}$, as follows: Introduce a participant for each snake of $\tilde{N}$, and read off the preference ordering of a participant from the order in which his snake participates in triples. A participant is a man, woman, or dog if and only if his snake is colored 1, 2, or 3 respectively. There are two difficulties: in the resulting instance, the numbers of the three different kinds of participants may not be equal, and each participant may not list all the triples he is supposed to. These difficulties may be resolved as in the proof of Lemma 19 by introducing extra snakes and appending a circuit. □

Theorem 37 *Three-party Stable Marriage is $\mathcal{NP}$-complete.*

Proof: By Lemma 36, it suffices to show that Y-NS is $\mathcal{NP}$-complete. The Y-gate can simulate and be simulated by $\{NAND,COPY\}$, so Y-NS is equivalent to $\{NAND,COPY\}$-NS, which is known to be $\mathcal{NP}$-complete [21]. □

9 Open problems

We seek new applications of the network approach to Stable Matching. We would like to find a fast parallel algorithm for Stable Matching. We would also like to find a fast parallel reduction from the Stable Marriage problem to the Assignment problem that preserves the structure of solutions. Finally, we would like to be able to analyse the performance of the randomized algorithm of Section 6.2.3. In particular, can anything be said about the average number of propagations taken by the randomized algorithm to find a stable matching?

Acknowledgements

This research has benefited from the ideas and suggestions of many. Ernst Mayr introduced me to the X-gate, and Danny Soroker introduced me to Stable Marriage. Discussions with Richard Anderson, Richard Cleve, Shaibal Roy, and Danny Soroker were very helpful in the early stages of this research. Marianne Baudinet helped me read portions of [20] and pointed me to Tarski's theorem; David Wolfe suggested that I look at Three-party Stable Marriage; and Alan Hoffman pointed out the importance of the non-algorithmic proof of Section 6.2.1. Tomás Feder deserves very special thanks for simplifying the proof of Lemma 4, for many absorbing discussions, and for extending the network approach in wonderful ways in [5].

I would especially like to thank Ernst Mayr and Christos Papadimitriou for many useful discussions, and for steadfast guidance and encouragement throughout the tenure of this research. I would also like to thank Andrew Goldberg, Dan Gusfield, Don Knuth, and Jeff Ullman for their comments at various stages of this research. This research has also been enriched by many intangible contributions from my fellow students and friends.

References

[1] G. Birkhoff, *Lattice Theory,* American Mathematical Society Colloquium Publications, volume 25, American Mathematical Society, 1967.

[2] S. A. Cook, indirect personal communication.

[3] S. A. Cook, *A taxonomy of problems with fast parallel algorithms,* Inform. and Control, 64(1985), pp. 2-22.

[4] S. A. Cook and h/l. Luby, *A simple parallel algorithm for finding a satisfying truth assignment to a 2-CNF formula,* Inform. Process. Lett., 27(1988), pp. 141-145.

[5] Tomás Feder, *A new fixed point approach for stable networks and stable marriages,* in Proc. 21st Annual ACM Symposium on Theory of Computing, 1989, pp. 513-522.

[6] Tomás Feder, personal communication.

[7] D. Gale and L.S. Shapley, *College admissions and the stability of marriage,* Amer. Math. Monthly, 69(1962), pp. 9-15.

[8] D. Gale and M. Sotomayor, *Some remarks on the stable matching problem,* Discrete Appl. Math., 11(1985), pp. 223-232.

[9] L. M. Goldschlager, *The monotone* and *planar circuit value problems* are *logspace* complete for *P,* SIGACT News, 9(1977), pp. 25-29.

[10] D. Gusfield, *Three fast algorithms for four problems in stable marriage,* SIAM J. Comput ., 16(1987), pp. 111-128.

[11] D. Gusfield, *The structure of the stable roommate problem: Efficient representation and enumeration of all stable assignments,* SIAM J. Comput., 17(1988), pp. 742-769.

[12] D. Gusfield and R. W. Irving, *The Stable Matching Problem: Structure and Algorithms,* MIT Press Series in the Foundations of Computing, MIT Press, 1989.

[13] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley, 1979.

[14] R. W. Irving, *An efficient algorithm for the stable roommates problem,* **J.** Algorithms, 6(1985), pp. 577-595.

[15] R. W. Irving, *On the stable room-mates problem,* Tech. Report CSC/86/R5, Universitv of Glasgow, 1986.

[16] R. W. Irving and P. Leather, *The complexity of counting stable marriages,* SIAM J. Comput., 15(1986), pp. 655–667.

[17] R. W. Irving, P. Leather, and D. Gusfield, *An efficient algorithm for the optimal stable marriage,* J. Assoc. Comput. Mach., 34(1987), pp. 532-543.

[18] *N.* D. Jones, *Space-bounded reducibility among combinatorial problems,* J. Comput. System Sci., 11(1975), pp. 68–85.

[19] R. M. Karp and V. Ramachandran, *A survey of parallel algorithms for shared-memory machines*, Tech. Report UCB-CSD 88/408, University of California, Berkeley, 1988.

[20] D. E. Knuth, *Mariages stables et leurs relations avec d'autres problèmes combinatoires,* Les Presses de l'Université de Montreal, 1976.

[21] E. W. Mayr and A. Subramanian, *The complexity of circuit value and network stability,* in Proc. 4th Annual Conference on Structure in Complexity Theory, 1989.

[22] C. Ng and D. S. Hirschberg, *Complexity of the stable marriage and stable roommate problems in three dimensions,* manuscript, 1988.

[23] G. Pólya, R. E. Tarjan, and D. R. Woods, *Notes on Introductory Combinatorics,* Birkhäuser, 1983.

[24] W. L. R*uzzo, On uniform circuit complexity,* J. Comput. System Sci., 22(1981), pp. 365-383.

[25] D. Soroker, personal communication.

[26] L. Stockmeyer, *Classifying the computational complexity of problems,* J. Symbolic Logic, 52(1987), pp. l-43.

[27] A. Subramanian, *The Computational Complexity of the Circuit Value and Network Stability Problems,* doctoral dissertation, Stanford University, 1989 (expected).

[28] A. Tarski, *A lattice-theoretical fixpoint theorem and its applications,* Pacific J. Math., 5(1955), pp. 285–309.

[29] L. G. Valiant, *The complexity of enumeration and reliability problems,* SIAM J. Comput., 8(1979), pp. 410-421.