August 1989 Report No. STAN-CS-89-1278

The Complexity of Circuit Value and Network Stability

by

Ernst W.Mayr and Ashok Subramanian

Department of Computer Science

Stanford University

Stanford, California 94305

The Complexity of Circuit Value and Network Stability *

Ernst W. Mayr
Fachbereich Informatik (20)
J. W. Goethe Universität
6000 Frankfurt am Main 11
West Germany

Ashok Subramanian
Computer Science Department
Stanford University
Stanford, CA 94305-2140
USA

Abstract. We develop a method for non-trivially restricting fanout in a circuit. We study the complexity of the Circuit Value problem and a new problem, Network Stability, when fanout is limited. This leads to new classes of problems within $\mathcal{P}$. We conjecture that the new classes are different from $\mathcal{P}$ and incomparable to $\mathcal{NC}$. One of these classes, CC, contains several natural complete problems, including Circuit Value for comparator circuits, Lex-first Maximal Matching, and problems related to St able Marriage and St able Roommates.

When fanout is appropriately limited, we get positive results: a parallel algorithm for Circuit Value that runs in time about the square root of the number of gates, a linear-time sequential algorithm for Network Stability, and logspace reductions between Circuit Value and Network Stability.

Abbreviated title. Circuit Value and Network Stability

1 Introduction

1.1 Restricting fanout in circuits

In the circuit model of computation, it is traditional to assume that once a value is computed, copies of this value may be made. If two copies can be made, any number of copies can be made, at little extra cost; so there does not seem to be a meaningful way to restrict fanout without reducing the circuit to a forest. We develop a method for non-trivially restricting fanout by altering the definition of 'circuit'. In our model, each input to the circuit and each output of a gate can either enter one gate or become an output of the circuit, but not both. However, we allow multiple-output gates so that the computing power of our circuits is not unduly restricted. Control of fanout is then obtained by placing certain conditions on the different outputs produced by a gate.

We identify two properties of (multiple-output) gates as being relevant to our enterprise. We define what it means for a gate to *preserve adjacency*, and for a gate to have *scatter*. We offer two definitions of what it might mean for a gate to support fanout: a gate supports fanout in the strong sense if it can simulate copying; a gate supports fanout in the weak sense if it can simulate a gate with more non-trivial outputs than inputs. We can tell whether a gate supports fanout in either

*This work was supported in part by a grant from AT&T and NSF grant DCR-8351757.

sense: a gate supports fanout in the strong sense if and only if it does not preserve adjacency; a gate supports fanout in the weak sense if and only if it has scatter.

1.2 The problems we study

The first problem we study is Circuit Value. Here the task is, given a description of a boolean circuit and its inputs, to compute the value of its output(s). We also define and study the Network Stability problem. For our purposes, a network is a boolean circuit with feedback. In the Network Stability problem, the task is to tell, given a network and its inputs, whether there is a consistent way to assign values to the edges of the network. The focus of our research is on how the complexity of Circuit Value and Network Stability depends on the properties of the gates constituting the circuit or network. The interesting results arise when the gates cannot support fanout, in one or both of the two ways presented earlier.

1.3 Connections with complexity theory

"Is logarithmic space as powerful as polynomial time?" is a long-standing open problem in complexity theory. The answer is widely believed to be "no," but no one has been able to supply a proof. Instead, complexity theorists have developed a hierarchy of classes of problems between logarithmic space and polynomial time: $\mathcal{L} \subseteq \mathcal{NL} \subseteq NC \subseteq \mathcal{P}$. (See [6]) Here $\mathcal{L}$ and $\mathcal{NL}$ are space complexity classes, and capture what can be achieved in logarithmic space on deterministic and non-deterministic Turing machines; $\mathcal{P}$ captures the power of polynomial time on a deterministic Turing machine. The class NC is a parallel complexity class; it contains all the problems that are highly parallelizable using a feasible amount of computational resources. The question "Is $NC = \mathcal{P}$?" is a fundamental question in parallel computation: it asks whether all problems that can be feasibly solved on a sequential machine can be solved very fast in parallel with feasible amounts of hardware. Again, the answer is believed to be "no," but no one has a proof.

In this paper, we identify new classes of problems between $\mathcal{L}$ and $\mathcal{P}$. We conjecture that these classes are properly contained in $\mathcal{P}$, and are incomparable to NC. One of these classes, CC, contains several natural complete problems.

1.4 Summary of results

The complexity of Circuit Value and Network Stability is easily determined if the gates support fanout in the strong sense; usually we get problems that are complete for a standard complexity class. If the gates do not support fanout even in the weak sense, we get efficient algorithms and parallel reductions. The case where the gates support fanout in the weak sense but not in the strong sense has recently been resolved by the work of Feder [9].

We give a parallel algorithm for Circuit Value that runs in time about the square root of the number of gates if the gates are scatter-free. In contrast, the fastest known parallel algorithm for general Circuit Value takes close to linear time. We also give a linear-time sequential algorithm for Network Stability if the gates are scatter-free. In addition, we give logspace reductions between the Network Stability problem over a scatter-free basis and the Circuit Value problem over the same basis.

We identify a class of natural problems that are efficiently interreducible with the problem of determining the output of a comparator circuit. These problems include Lex-first Maximal Matching and several problems related to Stable Matching.

1.5 Outline of the paper

This paper discusses the key results that appeared in the preliminary version [21] in greater depth. The reader interested in a more complete exposition is referred to [32].

Section 2 contains basics. Section 2.2 discusses two definitions of what it might mean for a set of gates to support fanout. Section 3 proves a basic result about simulations. Section 4 studies the complexity of Circuit Value. The parallel algorithm for Circuit Value that runs relatively quickly if the gates are scatter-free is described in Section 4.2.1. Section 5 studies Network Stability. In Section 5.3.1, we present the linear-time algorithm for Network Stability over scatter-free bases, and show how to reduce such a Network Stability problem to a Circuit Value problem over the same basis. Section 6 discusses the class CC. We discuss exciting related research in Section 7, and conclude in Section 8.

2 Preliminaries

2.1 Gates, circuits, and networks

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. A *basis* is a set of gates.

Gate	Inputs	outputs	Preserves adjacency?	Scatter-free?
COPY	i_1	$o_1 = i_1, o_2 = i_1$	*No*	*No*
NOT	i_1	$o_1 = \overline{i_1}$	*Yes*	*Yes*
AND	i_1, i_2	$o_1 = i_1\, i_2$	*Yes*	*Yes*
OR	i_1, i_2	$o_1 = i_1 + i_2$	*Yes*	*Yes*
NAND	i_1, i_2	$o_1 = \overline{i_1} + \overline{i_2}$	*Yes*	*Yes*
XOR	i_1, i_2	$o_1 = i_1\,\overline{i_2} + \overline{i_1}\,i_2$	Yes	Yes
c	i_1, i_2	$o_1 = i_1\, i_2, o_2 = i_1 + i_2$	Yes	Yes
X	i_1, i_2	$o_1 = i_1\,\overline{i_2}, o_2 = \overline{i_1}\,i_2$	Yes	Yes
δ_3	i_1, i_2, i_3	$o_1 = \bar{i}_1\,\bar{i}_2\,\bar{i}_3, o_2 = \overline{i_1}\,i_2\,i_3, o_3 = i_1\,\overline{i_2}\,i_3, o_4 = i_1\,i_2\,\overline{i_3}$	Yes	No

Table 1: Some of the gates that appear in this paper

The *influence graph* of a gate tells which outputs depend on which inputs. The jth input, i_j, of a gate *g influences* the kth output, o_k, if there are two input words s_1 and s_2 differing only in the jth bit, such that $g(s_1)$ and $g(s_2)$ differ in the kth bit. The influence graph of a gate has one vertex for each input and each output of the gate, and an edge between the vertices corresponding to input i_j and output o_k if and only if i_j influences o_k. A gate with a disconnected influence graph is essentially the same as a set of gates; we assume henceforth that each gate has a connected influence graph. In particular, our gates do not have trivial inputs or outputs, i.e., every input contributes in some way to the function g, and every output depends in some way on the inputs.

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 function in the usual 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 copying of computed values is explicit and must occur within a gate, and multiple-output gates are allowed. These provisions help us capture the intuitive notion of fanout without restricting the computing capability of circuits. Our definition of 'network' is non-standard. A network is essentially a combinational circuit with feedback, and is not intended to 'compute' anything.

Some of the gates that appear in this paper are listed in Table 1. The reader is invited to refer to this table while reading the following definitions.

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 X-gate is a restriction of the δ_3-gate, obtained by fixing any one input of the &-gate to 1. The *closure* of a basis Ω is the basis Ω^* consisting of all restrictions of gates in Ω.

Gate g can be *simulated* by basis Ω if there is a circuit over Ω^* that computes the function g. 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. Two bases are *equivalent* if each can simulate the other. For example, {X} is equivalent to {C, *NOT*}.

Two binary words of the same length are *adjacent* if they differ in at most one bit. A gate preserves adjacency (is *adjacency-preserving)* if it maps adjacent input words to adjacent output words. The Hamming distance between two binary words of the same length is the number of bit positions where the two words differ. It is clear that a gate preserves adjacency if and only if it does not increase Hamming distances: If g preserves adjacency, and s_1 and s_2 are two input words to g, the Hamming distance between $g(s_1)$ and $g(s_2)$ is at most the Hamming distance between s_1 and s_2. A basis preserves adjacency if every gate in it preserves adjacency.

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. Notice that a gate preserves adjacency if and only if every one-input restriction has at most one output; hence every scatter-free gate preserves adjacency. However, adjacency-preserving gates may have scatter; in fact, the number of outputs of an adjacency-preserving gate may be exponential in the number of inputs. An example is the δ_λ gate, with λ inputs and $2^{\lambda-1}$ outputs. We say that an input word to a gate is ***even*** if it contains an even number of 1s. The δ_λ gate has $2^{\lambda-1}$ even input words, and is defined as follows: the jth output bit is 1 if and only if the input word to the gate is the jth even input word. It may be verified that this gate preserves adjacency.

All the adjacency-preserving gates that have arisen in applications so far, e.g., the C-gate (the comparator) and the X-gate, are scatter-free.

2.2 When does a basis support fanout?

We consider two definitions. In both cases, the property of *not* supporting fanout is invariant under the operations of restriction, simulation, taking subsets, and taking finite unions. In other words, if Ω has the property, so will any basis composed of restrictions of gates in Ω, or any basis that can be simulated by Ω, or any subset of Ω; if Ω_1 and Ω_2 have the property, so will $\Omega_1 \cup \Omega_2$. Both characterizations are effective: there is an algorithm to tell if a finite basis supports fanout.

Definition: A basis supports fanout in the strong sense if it can simulate *{COPY}*. A basis supports fanout in the weak sense if it can simulate a gate with more outputs than inputs.
Algorithmic characterizations follow from Lemma 1 and Corollary 3 below.

Lemma 1 *A **basis supports** fanout **in** the **strong** sense **if** and only **if** it does not preserve adjacency.*

Proof: Suppose basis Ω does not preserve adjacency. Then it contains a gate g_1 that does not preserve adjacency. This gate has a restriction g_2 with one input i_1 and two outputs; each output equals either i_1 or $\overline{i_1}$. If both outputs equal i_1, then g_2 is the *COPY* gate. Otherwise, g_2 can simulate {NOT}, and we can get the *COPY* gate from g_2 by attaching NOT gates to a subset of the outputs. In either case, Ω can simulate {COPY}; hence it supports fanout in the strong sense.

Suppose basis Ω preserves adjacency. A gate preserves adjacency if and only if it has the following property: when one input bit is changed, at most one output bit changes. Circuits over an adjacency-preserving basis also inherit this property; hence no such circuit can simulate the *COPY* gate. It follows that Ω does not support fanout in the strong sense. II

Lemma 2 *Basis Ω is scatter-free if and only if every network over Ω^* has at most us many outputs us inputs.*

Proof: Let N be a network over Ω^*; let *Gates* be the set of gates of N. Then

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

If Ω is scatter-free, so is Ω^*. Thus each summand on the right hand side is non-negative; hence the sum is non-negative, and the number of outputs of N does not exceed the number of inputs of N. On the other hand, if Ω has scatter, there is a gate in Ω^* with more outputs than inputs. □

Corollary 3 *A basis supports fanout in the weak sense if and only **if** it has scatter.*

2.3 Basic Complexity Theory

For an introduction to complexity theory, see [30, 13]; for an introduction to parallel computation, see [6, 16].

In this paper, the problems are decision problems, and the reductions are many-one logspace-uniform $\mathcal{NC}^1$-reductions. Two problems are *equivalent* if they are reducible to each other. A problem is *complete* for a class if it is in the class and every problem in the class reduces to it.

3 A simulation result

Lemma 1 tells us that bases that do not preserve adjacency can simulate {COPY}. We use this to show that these bases form exactly seven classes of equivalent bases. Table 2 gives a classification algorithm.

Assume that Ω is a basis that does not preserve adjacency.
Ω is monotone $\Rightarrow$
Ω is AND-type and OR-type $\Rightarrow \Omega \equiv$ {AND, *OR, COPY).*
Ω is AND-type but not OR-type $\Rightarrow \Omega \equiv$ *(AND, COPY}.*
Ω is OR-type but not AND-type $\Rightarrow \Omega \equiv$ *{OR, COPY}.*
Ω is neither AND-type nor OR-type $\Rightarrow \Omega \equiv$ *{COPY}.*
Ω is non-monotone $\Rightarrow$
Ω is AND-type $\Rightarrow \Omega \equiv$ *{NAND, COPY}.*
Ω is OR-type $\Rightarrow \Omega \equiv \{NAND, COPY\}$.
Ω is neither AND-type nor OR-type $\Rightarrow$
Ω contains at least one gate that has two or more inputs $\Rightarrow \Omega \equiv \{XOR,$ *COPY}.*
Every gate in Ω has a single input $\Rightarrow \Omega \equiv$ {NOT, *COPY}.*

Table 2: Classifying bases that do not preserve adjacency

We begin with some definitions and observations. A X-input, p-output gate may be regarded as a function from $\mathbf{2}^{\lambda}$ to 2^{μ}; the domain and range of this function are boolean lattices under the natural ordering $\prec$. Gate g is said to be *monotone* if $s_1 \prec s_2 \Rightarrow g(s_1) \prec g(s_2)$; otherwise it is *non-monotone.* A basis is said to be monotone if every gate in it is monotone. It is easily checked that a basis is monotone if and only if it cannot simulate {NOT}.

Let g be a gate with a single output. Define $frac(g)$, the *l-fraction* of g, to be the fraction of input words of g on which the output of g is 1. Basis Ω is said to be an AND-type basis if Ω^* contains a single-output gate g with $0 < frac(g) < 0.5$. Every AND-type basis can simulate *{AND}.* To see this, observe that any gate g_1 with $0 < frac(g_1) < 0.5$ has a two-input restriction g_2 with $frac(g_2) = 0.25$.

In addition, if Ω is monotone, we claim that the following three statements are equivalent: Ω is AND-type; Ω can simulate *{AND};* Ω cannot be simulated by *{OR, COPY}.* We give the proof in the case where Ω consists of a single gate g with a single output; the proof in the general case is a straightforward generalization. Define a *minterm* of a boolean function to be a minimal set of assignments to the input variables that forces the boolean function to take the value 1. If every minterm of g involves exactly one input variable, then g is just the OR of a subset of the inputs; hence all three statements are false. If g has a minterm involving two or more input variables, then g has a restriction that is the *AND* of two inputs; hence all three statements are true. This proves the claim.

Basis Ω is said to be an OR-type basis if Ω^* contains a single-output gate g with $0.5 < f\ rac(g) < 1$. The following assertions are obtained by duality. Every OR-type basis can simulate *{OR}.* If Ω is monotone, the following are equivalent: Ω is OR-type; Ω can simulate *{OR};* Ω cannot be simulated by {AND, *COPY}.*

A gate is said to be *linear,* if every output function of the gate can be expressed as a $GF(2)$-linear function of the inputs and the constant 1. A basis is said to be linear if every gate in it is

linear. It may be checked that the following statements are equivalent: basis Ω is linear; basis Ω can be simulated by *{XOR, COPY};* basis Ω is neither AND-type nor OR-type.

Theorem 4 *Let* Ω *be a basis that does not preserve adjacency. Then it is equivalent to exactly one of the following buses: (NAND, COPY), (AND, OR, COPY), (AND, COPY), (OR, COPY), (XOR, COPY}, {NOT, COPY}, {COPY}.*

Proof: A classification algorithm is given in Table 2. The correctness of the algorithm follows from previous observations and basic boolean algebra. cl

4 Circuit Value Problems

An instance I of Circuit Value consists of a circuit C, say with λ inputs, an input assignment $s_{in} \in \{0, 1\}^{\lambda}$ to the circuit, and an output edge $\tilde{e}$ of C. The Circuit Value problem (CV) is "Given an instance I of Circuit Value, is edge $\tilde{e}$ assigned the value 1 when circuit C is evaluated on input assignment s_{in}?" The complementary problem, *co*-CV, asks if edge $\tilde{e}$ is assigned the value 0. These problems are P-complete [19]. Given a basis Ω, we define Ω-CV and *co*-Ω-CV to be the special cases of CV and *co*-CV, respectively, in which the circuits are required to be over Ω. Under reasonable assumptions about the form of the input, the Circuit Value problem over any basis Ω can be solved in linear time. In this section, we study how the parallel complexity of Ω-CV depends on Ω.

4.1 Bases that do not preserve adjacency

Basis Ω	Complexity of R-CV
{NAND,COPY}	Same as Circuit Value, so P-complete [19]
{*AND,OR,COPY*}	Same as Monotone Circuit Value, so P-complete [11]
{*OR,COPY*}	NL-complete; in $\mathcal{NC}^2$ because $\mathcal{NL} \subseteq \mathcal{NC}^2$ [3]
{*AND,COPY*}	co-NG-complete; also $\mathcal{NL}$-complete because $\mathcal{NL} =$ *co*-$\mathcal{NL}$ [14]
{*XOR,COPY*}	in $\mathcal{NC}^2$
{*NOT,COPY*}	in $\mathcal{L}$ (in fact, NC^1-complete (see [6, 7]) for $\mathcal{L}$)
(COPY)	in $\mathcal{L}$ (in fact, $\mathcal{NC}^1$-complete for $\mathcal{L}$)

Table 3: The complexity of Circuit Value if the basis does not preserve adjacency

Equivalent bases define equivalent Circuit Value problems, so the complexity of Ω-CV when Ω does not preserve adjacency may be determined by case analysis. The results in Table 3 are obtained by standard techniques. It is interesting to note that six of the seven circuit value problems are complete for natural complexity classes. Many of the results in this table have previously been obtained in [12]. An inspection of the seven cases yields:

Theorem *5 If* Ω *does not preserve adjacency, R-CV is equivalent to co-*Ω*-CV.*

Remark: Immerman's theorem [14] is the special case of Theorem 5 with $\Omega = \{OR, COPY\}$; for this reason, we require his proof to prove Theorem 5. Clearly, Ω-CV and *co*-Ω-CV are equivalent if Ω is non-monotone; we do not know whether Ω-CV and *co*-Ω-CV are equivalent for all bases.

4.2 Bases that preserve adjacency

Our understanding of the complexity of CV over adjacency-preserving bases is highly incomplete. In Section 4.2.1, we give a parallel algorithm for CV when the gates lack scatter.

4.2.1 A parallel algorithm for Circuit Value if the basis lacks scatter

Each gate of a circuit prescribes certain relationships between the values of its inputs and the values of its outputs. These relationships may be written as implications, in the form $\bigwedge(i_j \leftarrow v_j) \Rightarrow (o_k \leftarrow v_k)$, where the i_j are input edges, the o_k are output edges, and the v_j and v_k are values from $\{0, 1\}$. In some of these implications, the partial assignment that appears on the left hand side consists of an assignment to a single edge. We call these the *short* implications; all other implications are *long*. For example, we can write the following implications for an *OR* gate with inputs i_1, i_2 and output o: $i_1 \leftarrow 1 \Rightarrow o \leftarrow 1$; $i_2 \leftarrow 1 \Rightarrow o \leftarrow 1$; and $\bigwedge(i_1 \leftarrow 0, i_2 \leftarrow 0) \Rightarrow o \leftarrow 0$. Here the first two implications are short. The circuit value problem over any basis may be solved by starting with the input assignments and enforcing the implications until every edge of the circuit is assigned a value. The reason we single out the short implications is because these implications are easily enforced in parallel. In other words, every assignment implied by a given partial assignment and a set of short implications may be determined in NC, by directed graph transitive closure. This suggests the following parallel algorithm (Algorithm 1) to solve Circuit Value. The algorithm is similar in spirit to the DTEP algorithm of [20] and the Contraction algorithms of Miller [22, 23, 24]: Step 1 is a 'compress' step; Step 2 is a 'rake' step.

Algorithm 1 Initialize S, the set of assignments known to be true so far, to the input assignments. Initialize *Short* to be the set of all correct short implications for all the gates of the circuit. Then alternate the following two steps until each edge has been assigned a value:

1. Augment S using *Short*. (Use a transitive closure algorithm.)
2. Consider each long implication. If each assignment on the left hand side is in S, use the assignment on the right hand side to augment S. If all but one assignment on the left hand side are in S, we get a short implication, which we use to augment *Short*.

The algorithm is correct because circuits are acyclic. Lemma 6 below shows that the algorithm runs relatively quickly if the basis is scatter-free.

Lemma *6 Given a circuit over a scatter-free basis and an input assignment to it, Algorithm 1 deduces the value* ***of*** *all edges in* $O(\sqrt{m})$ *iterations, where m is the size* ***of*** *the circuit.*

Proof: Algorithm 1 may be regarded as a simplification algorithm. In other words, each iteration takes in a circuit and an input assignment to it, deduces the values of some edges in Step 1, and uses the resulting partial assignment to simplify the circuit in Step 2; the result of an iteration is a simplified circuit and an input assignment to it. After sufficiently many iterations, the circuit is fully simplified, i.e., each edge has been assigned a value.

We consider a conservative variant of the simplification algorithm. The new algorithm has the overall structure of the old algorithm; however, it does not always make all possible assignments or simplifications, but only a subset of them. As a result, the assignments inferred by the new

(conservative) algorithm after k iterations will be a subset of those inferred by the old algorithm after k iterations. We then complete the proof of Lemma 6 by showing that the conservative algorithm terminates in $2\lceil\sqrt{m}\rceil$ iterations.

Iteration k of the conservative algorithm begins with a circuit C_k and an input assignment to C_k. The partially simplified 'gates' that appear in C_k might not have connected influence graphs; so we call these entities 'elements'. Each element of C_k will have an equal number of inputs and outputs. (This invariant may be achieved at the beginning of the first iteration by adding extra outputs as needed to the elements of Cr.) Hence C_k has an equal number of inputs and outputs; define the *width* of iteration k to be equal to this common number. Since each element is derived from a single scatter-free gate of the original circuit, fixing any λ inputs of an element will always permit us to assign values to λ outputs of the element.

The first step of an iteration of the conservative algorithm identifies a set of short implications and runs a transitive closure algorithm on these implications to derive new assignments. The set of implications is chosen so that fixing any one input of any element causes exactly one output of the element to be fixed via these short implications. In other words, for each edge **i** that is an input edge to some element, we choose two short implications of the form $i \leftarrow 0 \Rightarrow e_0 \leftarrow v_0$ and $i \leftarrow 1 \Rightarrow e_1 \leftarrow v_1$. Our choice of short implications guarantees that the first step can never assign values to more outputs of an element than to its inputs. The second step of the iteration restores the invariant that each element has an equal number of inputs and outputs, by fixing the correct number of additional outputs of each element and discarding fixed inputs and outputs. The edges that are assigned values in the second step become input edges of the simplified circuit.

We now show that the conservative algorithm takes at most $2\lceil\sqrt{m}\rceil$ iterations. Each input edge of C_k is removed during the second step of iteration k. Hence, within the first $\lceil\sqrt{m}\rceil$ iterations, there will be an iteration with width at most $\lceil\sqrt{m}\rceil$. We show that there can be at most $\lceil\sqrt{m}\rceil$ more iterations. Notice that if C_k is not the empty circuit, at least one output edge of C_k is assigned a value during the first step of iteration k. This means that C_{k+1} has fewer outputs than C_k; hence the width of iteration $(k + 1)$ is strictly less than the width of iteration k. It follows that the width of the circuit becomes zero within $2\lceil\sqrt{m}\rceil$ iterations. □

Remark: We prove that the bound in Lemma 6 is tight, by exhibiting an infinite family of comparator circuits on which Algorithm 1 takes $\Theta(\sqrt{m})$ iterations. Let C_k be the circuit with k^2 comparators arranged in a $k \times k$ grid, as follows: the inputs of each comparator come from the left and below; the 'min' output of each comparator goes upward; and the 'max' output of each comparator goes to the right. Let the input assignment s_k be as follows: each horizontal input edge is assigned the value 0; each vertical input edge is assigned the value 1. Suppose the simplification algorithm is presented with circuit C_k and input assignment s_k. It can be checked that the output produced after one iteration is the simplified circuit C_{k-1} and the input assignment s_{k-1}. Hence Algorithm 1 takes k iterations to deduce the value of all edges of C_k.

Theorem 7 *Let Ω be a scatter-free basis. Then there is a PRAM (Parallel Random Access Machine) algorithm for Ω-CV that runs in $O^*(\sqrt{m})$ time[1] using a polynomial number of processors, where m is the size of the circuit.*

[1] The **'O' notation means that polylogarithmic multiplicative factors are suppressed.**

Proof: Lemma 6 tells us that Algorithm 1 takes $O(\sqrt{m})$ iterations; each iteration can be implemented in polylogarithmic time on a PRAM with a polynomial number of processors. □

5 Network Stability Problems

The Network Stability problem (NS) is a question about the existence of *configurations* of a network consistent with a given input assignment. In this section, we study the sequential and parallel complexity of Network Stability as a function of the kinds of gates allowed in the network.

An instance I of Network Stability consists of a network N with $\lambda \geq 0$ inputs, and an input assignment $s_{in} \in \{0,1\}^{\lambda}$ to N. A *configuration* 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 is "Given an instance I of Network Stability, does $[N, s_{in}]$ have 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 R-NS to be the special case of NS in which the network is required to be over Ω. For instance, *{AND,OR,COPY}-NS* is the stability problem for monotone networks. In general, network stability problems are not known to be equivalent to their complementary problems.

The following lemma shows that Network Stability is usually at least as hard as Circuit Value:

Lemma *8 Let Ω be any basis that can simulate (NAND). Then R-CV reduces to R-NS.*

Proof: The proof uses the idea of a *forcer*. A v-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. 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. □

5.1 Networks over monotone bases

Theorem 9 *Every network over a monotone basis Ω has a stable configuration; **in fact, there is** a (unique) 'most-O' stable configuration Q^0 with the property that **if** an edge e is 0 in any stable configuration Q **of** the network, it is 0 in Q^0. This stable configuration can be found in linear time. Furthermore, the problem "Is edge $\tilde{e}$ assigned the value 1 in Q^0?" is equivalent to Ω-CV.*

Proof: We give the reduction to Ω-CV. Pretend that each edge e of the monotone network N has unit delay. Consider the following network process. Initialize N to the all-0 stable configuration. At time t = 0, assign each input edge of N its correct value. Now let the network 'run'. The values on edges of the network will change during this process; since the network is monotone, every change will be from a 0 value to a 1 value. Thus the value on any given edge will change at most once; eventually the network will reach a stable configuration. This is Q^0, because the value of an edge becomes 1 only if it must be 1 in order to satisfy the equations of the network.

The network process terminates in at most m time units, where m is the size of N. We may unravel the network computation into a circuit computation, by replacing each gate g of N by m gates (g, t), one for each time-step. The resulting circuit is over Ω. Its size is $O(m^2)$. *cl*

5.2 Networks over non-monotone bases that do not preserve adjacency

Equivalent bases define equivalent network stability problems. Theorem 4 tells us that there are only three inequivalent bases to consider. We consider these in turn.

We show that {NAND,COPY}-NS is $\mathcal{NP}$-complete. The problem "Does the given circuit C_0 have an input assignment that makes its output 1?" is $\mathcal{NP}$-complete. We may assume that C_0 is over $\{NAND,COPY\}$. Let ***flout*** be the O-input, l-output network obtained by connecting one output of a COPY gate to its input. Attach a ***float*** network to each input of C_0, and feed the output of C_0 into a l-forcer. The resulting input-free network has a stable configuration if and only if C_0 has an input assignment that makes its output 1. This completes the proof.

Constructing a stable configuration of a network over $\{XOR,COPY\}$ is the same as solving a system of linear equations over *GF(2);* such systems can be solved in $\mathcal{NC}^2$ [26, 4].

Networks over $\{NOT,COPY\}$ have an especially simple structure: each connected component may be rendered into a tree by deleting at most one edge. This can be exploited to find stable configurations in linear time, and in logarithmic space. It can also be shown by standard methods that co-$\{NOT,COPY\}$-NS is $\mathcal{NC}^1$-complete for $\mathcal{L}$.

5.3 Networks over non-monotone bases that preserve adjacency

The interesting question is: How hard is Network Stability if the basis preserves adjacency and can simulate *{NAND}?* We give a linear-time algorithm to construct a stable configuration if the basis is scatter-free. In addition, for such bases, we show that Ω-NS is equivalent to Ω-CV. Let $\mathcal{SFC}$ be the class of problems that are reducible to Circuit Value over scatter-free bases. Then the Network Stability problem over scatter-free bases is complete for $\mathcal{SFC}$.

5.3.1 Scatter-free networks

We use the characterization of Lemma 2 to give a linear-time algorithm that constructs a stable configuration of a scatter-free network, whenever one exists.

We are looking for stable configurations in a network N over a scatter-free basis Ω. We may assume that Ω is closed under restriction, since restrictions of scatter-free gates are scatter-free. The first step is to 'eliminate the inputs', i.e., simplify the problem so that N becomes input-free. The idea behind input elimination is that knowing the value of an input to a gate enables us to do two things: deduce the value of some (maybe none) of its outputs, and replace the gate by one of its restrictions. This process of network simplification may be iterated; it produces, in linear time, 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 are obtained by augmenting the stable configurations of N' with *P*. In particular, N has a stable configuration if and only if N' does. Since Ω is scatter-free, Lemma 2 applies to N'; since N' has no inputs, it 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]. Applying Lemma 2 to the network that remains after the propagation, we see that edge e_{out} must be assigned a value v_{out} during 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 that assign edge e the value v are obtained by augmenting the stable configurations of N' with P. If, on the other hand, $v_{out} \neq v$, no stable configuration of N may assign edge e the value v; in this case, we say the propagation is *unsuccessful.*

The *type* of an edge in an input-free scatter-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. It turns out that the type of an edge is preserved by successful propagations:

Lemma 10 *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: The key idea is that propagations only make forced moves. Hence every assignment made in the propagation of $[e_2, v_2]$ in N will be made during the propagation of $[e_2, v_2]$ in N', unless the same assignment has already been made during the propagation of $[e_1, v_1]$ in N. □

Lemma 10 justifies the following polynomial-time algorithm for constructing stable configurations in scatter-free networks:

Algorithm 2 Eliminate all inputs. Pick any edge e. If it is contradicting, there are no stable configurations; otherwise, suppose [e, v] is viable. Propagate [e, v], thus obtaining a partial assignment P and a simplified network N'. 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 11 *Let N be a network over a scatter-free basis* Ω. *Then there is a linear-time algorithm that will construct a stable configuration if there is one, or conclude that none exists.*

Proof: Algorithm 2 can be made to run in linear time. The idea is to keep the work done by the algorithm proportional to the amount of simplification achieved, i.e., the number of edges of the network that are assigned values. To do this, propagate [e, 0] and [e, I] 'in parallel', and use the propagation that succeeds first. □

Theorem 12 *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, there is clearly no stable configuration. If there is no contradicting edge, Algorithm 2 will find a stable configuration. □

Theorem 13 *Let* Ω *be any scatter-free basis that can simulate (NAND). Then* Ω-NS *is equivalent to* Ω-CV.

Proof: We use Theorem 12 to give a reduction from Ω-NS to Ω-CV; the reduction in the other direction follows from Lemma 8. Consider the following network process *Prl(e, v)* on the network N. Break edge e, thus creating a new input e_{in} and a new output e_{out}. Assign all input edges of N their correct values, and assign edge e_{in} the value v. Perform input elimination. Since the gates are scatter-free, each output will be assigned a value. Let $v_{out}(e, v)$ be the value assigned to edge e_{out}. We say that process Prl(e,v) *succeeds* if $v_{out}(e,$ v$) = v$, otherwise it *fails.* The crucial

observation is that both $Pr1(e, 0)$ and $Pr1(e, 1)$ fail if and only if e is a contradictory edge. The proof of this observation follows from the fact that the result of input elimination is insensitive to the order in which the inputs are eliminated.

Consider the following network process $Pr2(e, v)$: Pretend that each edge has unit delay. Break edge e of N, thus creating input e_{in} and output e_{out}. Initialize the network to an arbitrary configuration. Assign all input edges of N their correct values, and assign edge e_{in} the value v. Now let the network run. Every edge that was assigned a value during process $Pr1(e, v)$ will stabilize within m time units at the same value in $Pr2(e, v)$. (Here m is the size of N.) In particular, edge e_{out} will stabilize at $v_{out}(e, v)$. Unravelling the network process $Pr2(e, v)$ into a circuit computation yields a circuit over Ω that computes $v_{out}(e, v)$.

Given all the values $v_{out}(e, v)$, we may use NAND gates to combine the results and determine whether N has a stable configuration. The resulting circuit has size $O(m^3)$. □

6 The Comparator Circuit Value Problem, and the class CC

The Comparator Circuit Value problem, C-CV, is the circuit value problem over the monotone scatter-free basis {C}. Define CC to be the class of problems reducible to C-CV. In this section, we discuss some properties of C-CV and exhibit some natural CC-complete problems.

The outputs of the comparator are threshold functions. For $\lambda \geq 2$, define the τ_λ gate to be the monotone scatter-free gate with λ inputs $i_1 \ldots i_\lambda$ and λ outputs $o_1 \ldots o_\lambda$, where o_j is the jth threshold function $T_j(i_1, \ldots, i_\lambda)$, i.e., output o_j is 1 if and only if at least j inputs are 1. The comparator is just the τ_2 gate. The τ_λ gate essentially sorts its inputs. Since sorting circuits can be built with comparators, each τ_λ gate can be simulated by the comparator. It follows that $\{\tau_\lambda\}$ is equivalent to {C}, for each $\lambda \geq 2$.

The 'double rail' method of Goldschlager [11] may be used to reduce $\{\tau_\lambda$, NOT}-CV to $\{\tau_\lambda\}$-CV. The idea is to keep each variable around in both the complemented and uncomplemented forms. The key step is to produce both forms of the outputs of a gate, given both forms of the inputs. For a NOT gate, this is trivial. For a τ_λ gate, we can do this with two τ_λ gates: the first τ_λ gate takes in the uncomplemented inputs and produces the uncomplemented outputs; the second τ_λ gate takes in the complemented inputs and produces the complemented outputs in reverse order. Thus $\{\tau_\lambda$, NOT}-CV is CC-complete. Since Ω-CV = co-Ω-CV when Ω is non-monotone, it follows that the class CC is closed under complementation.

Applying Theorem 13, we see that $\{\tau_\lambda$, *NOT}-NS* is $\mathcal{CC}$-complete; in particular, {C, *NOT}-NS* is CC-complete. Theorem 14 below, due to Feder [8], uses this to show that $\mathcal{NL} \subseteq$ CC. Since {X} is equivalent to {C, *NOT},* it follows that X-CV and X-NS are $\mathcal{CC}$-complete[2]. We use these facts in Sections 6.1 and 6.2.

Theorem 14 $\mathcal{NL} \subseteq$ **CC.**

Proof: The problem *GAP:* "Given a directed graph G and two vertices s and t, is there a directed path from s to t?" is complete for $\mathcal{NL}$ [15]. The problem remains complete even if the following conditions are imposed: s must be a source vertex; t must be a sink vertex; and G must be acyclic. There is no directed s-t path if and only if the vertices of G may be two-colored with the colors 0

[2] These **problems are properly called { X}-CV and { X}-NS; we drop the braces for convenience.**

and 1 so that $color(s)$ = 1; color(t) = 0; and colors never decrease along directed edges, i.e., if there is a directed edge from vertex v_1 to vertex v_2, then $color(v_1) \leq color(v_2)$. We build a network over {C, NOT} to enforce these 'non-decreasing' conditions.

A comparator circuit may be drawn as a collection of wires with comparator gates between them; see, e.g., Section 5.3.4 of [17]. Consider the following comparator circuit: introduce a wire for each vertex of G, and a comparator gate for each edge of G; the gate is oriented so that if the edge is directed from vertex v_1 to vertex v_2, then the associated comparator places its 'min' output on the wire associated with vertex v_1 and its 'max' output on the wire associated with vertex v_2. (The relative order of the comparators on the wires is immaterial.) The resulting circuit has one input edge and one output edge corresponding to each vertex of G. In addition, since G is acyclic, the circuit has the following property: on any input assignment, the output word of the circuit equals the input assignment if and only if none of the comparators are 'used' on that input assignment. We use this property to complete the reduction.

Convert the circuit into a network N over {C, *NOT*} as follows: feed the output edge corresponding to s into a l-forcer; feed the output edge corresponding to t into a O-forcer; for every other vertex v of G, connect the output edge corresponding to v to the input edge corresponding to v. Let s_{in} be the input assignment to N that assigns the value 1 to the input edge of N corresponding to vertex s, and assigns the value 0 to the input edge corresponding to vertex t. Given a stable configuration Q of [N, s_{in}], we show how to find an appropriate two-coloring of G. Since Q is stable, no comparator gets used. Hence Q assigns identical values to all the edges on any given wire of N; let this common value be the color of the associated vertex. It is easy to check that this coloring meets all the required conditions. Conversely, if G has an appropriate two-coloring, we can construct a stable configuration of [N, s_{in}]. Hence [N, s_{in}] has a stable configuration if and only if there is no directed s-t path in G. Thus we have reduced *GAP* to co-{C, *NOT}-NS*, which is CC-complete. cl

6.1 Lex-first Maximal Matching is CC-complete

Lex-first problems are often P-complete [6, 2, 25], but LFMM, the Lex-first Maximal Matching problem, appears not to be so. Attempts to reduce Circuit Value to LFMM fail because of an inability to 'fanout information.' [2]. The 'reason' for this phenomenon is that LFMM is $\mathcal{CC}$-complete.

An instance I of LFMM consists of an undirected graph G = (V, E), a total order $\prec$ on E, and a distinguished edge $\tilde{e}$. If $e_1 \prec e_2$, we say that e_1 *precedes* e_2. A matching M is a subset of E with the property that at most one edge in M is incident to any vertex of G. A matching is maximal if it is not properly contained in any other matching. The total order $\prec$ on the edges allows us to regard a matching M as a sequence S_M = $(e_1, e_2, \ldots)$ of edges in ascending order, *i.e.*, $j < k \Rightarrow e_j \prec e_k$. Given two (maximal) matchings M and N, we say that M << N if S_M lexicographically precedes S_N. The relation $\ll$ defines a total order over all maximal matchings. The minimum element, M_{lexfirst}, of this order is called the lex-first maximal matching of (G, $\prec$). The Lex-first Maximal Matching problem is "Given an instance I of LFMM, is $\tilde{e}$ in M_{lexfirst}?"

The following greedy algorithm finds M_{lexfirst}: Start with the empty matching, inspect the edges in ascending order, and augment the current matching whenever possible. Cook [5] showed that LFMM is equivalent to X-CV; hence LFMM is CC-complete. Before giving the reductions, we introduce some terminology.

An X-circuit is a circuit over {X}. 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-circuit in terms of *snakes.* A snake is a sequence of edges; in our application, the sequence is always 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 a snake is an input edge of the circuit; the last element of a snake is an output edge. Two snakes *meet* if there are two edges, one in each snake, that are input edges to the same gate. The snakes of an X-circuit constitute a partition of its edges.

An X-circuit may be described by naming its snakes and telling, for each snake, which other snakes it meets and in what order; in such a description, there is a gate wherever two snakes meet. There are efficient transformations between such a description and a standard description of an X-circuit.

Reducing LFMM to X-CV: Introduce a snake q_v for each vertex v of G. Two snakes meet if the corresponding vertices in G are neighbours, i.e., if they share an edge. Thus the gates of the X-circuit correspond to the edges of G. Let g_e denote the gate corresponding to edge e. The order in which the snake corresponding to a vertex meets the snakes corresponding to its neighbour vertices is induced by the total order $\prec$ on the edges of G, as follows: if vertex v is adjacent to vertices w and x in G, and (v, w) $\prec$ (v, x), then snake q_v meets snake q_w before it meets snake q_x. This completes the description of the X-circuit C. The relevant input assignment is **1**, i.e., the input assignment that assigns each input edge the value 1.

We show that edge e is matched in M_{lexfirst} if and only if both inputs to gate g_e are 1 when circuit C is evaluated on input assignment **1.** (Given this fact, it is easy to produce the instance of X-CV corresponding to instance *I* of LFMM from circuit C.) Observe that we may list the gates of C in topological order as follows: list the edges of G in ascending order of $\prec$; then read off the corresponding gates. Evaluating circuit C in this topological order simulates the greedy algorithm to find a lex-first maximal matching of G. Let e be any edge of G; let v be an endpoint of edge e. Then the following claim is easily proven by induction: the edge on snake q_v that enters gate g_e is assigned the value 1 when C is evaluated on input assignment 1 if and only if vertex v is unmatched at the time the greedy algorithm considers edge e. The proof is completed by observing that the greedy algorithm matches edge e if and only if both endpoints of e are unmatched at the time the algorithm considers this edge.

Remark: The reduction from LFMM to X-CV takes a graph with $|E|$ edges into a circuit with $|E| + 1$ gates. It follows from Theorem 7 that there is a parallel algorithm for LFMM that runs in $O^*(\sqrt{|E|})$ time.

Reducing X-CV to LFMM: Suppose we are given an X-circuit and an input assignment to it; suppose we wish to tell if output edge e_{out} gets value 1. Feed this edge into a new X-gate $\tilde{g}$ whose other input is 1. Let C be the resulting X-circuit. Then the answer to the X-CV instance is 'yes' if and only if gate $\tilde{g}$ has both inputs 1 when circuit C is evaluated on its input assignment. We may assume that a topological order of the gates of C is available (see [28]) because there is a fast parallel transformation that takes as input any circuit C and an input assignment s_{in} to C, and returns a (larger) circuit C' over the same basis, a topological order of the gates of 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 may assume that each pair of snakes of C meet at most once, because removing every intersection except the first does not change the values assigned to the edges of C. Notice that if the input edge of a snake is assigned the value 0 by the input assignment to C, all edges on this snake get the value 0; furthermore, we may delete this snake without affecting the values assigned to the other edges of C. Thus we may assume that the input assignment to C is **1.** Now construct a graph G = *(V, E) as* follows: introduce a vertex for each (remaining) snake; introduce an edge between two vertices if the corresponding snakes meet. The total order $\prec$ on E is read off from the topological order on the gates of C. The distinguished edge $\tilde{e}$ of the LFMM instance is the edge corresponding to gate $\tilde{g}$. It is easy to verify that edge $\tilde{e}$ is matched in the lex-first maximal matching if and only if the original X-CV instance has a 'yes' answer. This completes the reduction. The size of the LFMM instance is quadratic in the size of the X-CV instance; however, this increase in size may be avoided if a topological ordering of the X-circuit is available.

6.2 Stable Matching is CC-complete

The Stable Matching problem (see [10, 18,271 for an introduction) is essentially the same as Network Stability for networks over {X}. The Stable Marriage problem is a special case of Stable Matching; it is the same as Network Stability for comparator networks. Since X-NS and C-CV are equivalent, several Stable Matching problems are CC-complete. We state these results here; for proofs and extensions, we refer the reader to [31].

Theorem 15 *The following problems are CC-complete:*

a) Stable roommates: Does the given instance of Stable Roommates have a stable roommate assignment?

b) 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?

c) Stable pair: Is the given pair of persons a stable pair of the given instance of Stable Matching, i.e., is it true that there is a stable matching that matches these two persons to each other?

d) Minimum-regret: Given an instance I of Stable Matching and an integer k, is it true that there is a stable matching of I in which every person has regret at most k?

e) Man-optimal Stable Marriage: Given an instance of Stable Marriage, is man m matched to woman w in the man-optimal stable marriage?

7 Related research

Recently, Feder [9] has reported considerable progress in solving Network Stability for arbitrary adjacency-preserving bases. He shows how to find a stable configuration for a network over an arbitrary adjacency-preserving basis in cubic time. He also generalizes Theorem 13 to all adjacency-preserving bases that can simulate *{NAND}.* It follows that if $\mathcal{APC}$ denotes the class of problems that are reducible to Circuit Value over adjacency-preserving bases, then the Network Stability problem over adjacency-preserving bases is complete for $\mathcal{APC}$. Finally, for any adjacency-preserving basis Ω that can simulate {NAND}, he shows that constructing a stable configuration is equivalent to Ω-CV.

8 Conclusions

The complexity of Circuit Value and Network Stability has been studied as a function of the kinds of gates permitted in the network. The availability of fanout plays a crucial role in determining the complexity. The interesting new problems studied are Circuit Value and Network Stability over scatter-free and adjacency-preserving bases. The overall picture that emerges from our work and [9] is

$$\mathcal{NL} \subseteq \mathcal{CC} \subseteq \mathcal{SFC} \subseteq \mathcal{APC} \subseteq \mathcal{P}$$

where $\mathcal{SFC}$ and $\mathcal{APC}$ are the classes of problems that are reducible to Circuit Value over scatter-free and adjacency-preserving bases respectively. We conjecture that the three new classes are different from $\mathcal{P}$ and incomparable to **NC.**

We conclude with some directions for further research:

1. Is there a machine characterization of the new classes discussed in this paper?

2. Our definitions of fanout are 'all-or-nothing' definitions; for instance, a circuit over an adjacency-preserving basis cannot simulate even one COPY gate. Can anything interesting be said about the complexity of Circuit Value and Network Stability when a limited amount of copying is allowed?

3. What can be said about the relative difficulty of Circuit Value over different adjacency-preserving bases? Is there a linear-time algorithm for Network Stability over arbitrary adjacency-preserving bases?

4. Find other natural problems that are equivalent to Circuit Value or Network Stability over adjacency-preserving bases.

5. The reduction of Section 6.1 from X-CV to LFMM can be modified so that the graph G in the LFMM instance is bipartite. It follows that the Assignment problem (Bipartite Weighted Matching) is CC-hard. (A full proof appears in [31].) Of course, it is well known that the Assignment problem is in $\mathcal{P}$. What is the relationship between the Assignment problem and the classes $\mathcal{SFC}$ and $\mathcal{APC}$?

6. Find better parallel algorithms for C-CV, or for special cases of any CC-complete problem.

Acknowledgements

We have given an $O^*(\sqrt{n})$-time parallel algorithm for C-CV. Soroker [29] has independently discovered an $O^*(\sqrt{n})$-time algorithm for a CC-complete problem called Width-3 Priority Matching. There are $\mathcal{NC}^1$, linear-size reductions between this problem and C-CV, so a fast algorithm for one problem translates to a fast algorithm for the other. Anderson [1] shows how to implement our algorithm for C-CV in $O^*(\sqrt{n})$ time with $\sqrt{n}$ processors.

The second author would especially like to thank Tomás Feder and Christos Papadimitriou for many absorbing discussions in connection with this research. He would also like to acknowledge useful conversations with Richard Anderson, Richard Cleve, Andrew Goldberg, Phokion Kolaitis, Shaibal Roy, Jack Snoeyink, Danny Soroker, and Jeff Ullman.

References

[1] R. Anderson, personal communication.

[2] R. Anderson and E. W. Mayr, Parallelism and greedy algorithms. In "Advances in Computing Research", vol. 4, 17-38, JAI Press, 1987.

[3] A. Borodin, On relating time and space to size and depth, *SIAM J. Comput.* **6** (1977), 733–744.

[4] A. Borodin, J. von zur Gathen, and J. Hopcroft, Fast parallel matrix and GCD computations, *Inform. and Control,* **52** (1982), **241-256.**

[5] S. A. Cook, personal communication.

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

[7] S. A. Cook and P. McKenzie, Problems complete for deterministic logarithmic space, *J. Algorithms,* **8** (1987), 385-394.

[8] Tom&s Feder, personal communication.

[9] Tom&s Feder, A new fixed point approach for stable networks and stable marriages. In Proc. 21st Annual ACM Symposium on Theory of Computing, 1989, 513-522.

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

[11] L. M. Goldschlager, The monotone and planar circuit value problems are logspace complete for P, *SIGACT News,* **9(2)** (1977), 25-29.

[12] L. M. Goldschlager and I. Parberry, On the construction of parallel computers from various bases of boolean functions, *Theoret. Comput. Sci.,* **43** (1986), 43-58.

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

[14] N. Immerman, Nondeterministic space is closed under complementation, *SIAM J. Comput.* **17** (1988), *935-938.*

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

[16] 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, March 1988. Also in "Handbook of Theoretical Computer Science," North-Holland, to appear.

[17] D. E. Knuth, "The Art of Computer Programming," vol. 3, Addison-Wesley, 1973.

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

[19] R. E. Ladner, The circuit value problem is logspace complete for P, *SIGACT News, 7(l)* (1975), 18-20.

[20] E. W. Mayr, The dynamic tree expression problem. Tech. report STAN-CS-87-1156, Stanford University, May 1987. Also in "Concurrent Computations: Algorithms, Architecture, and Technology," 157-179, Plenum Press, New York, 1988.

[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, 114-123, 1989.

[22] G. L. Miller and J. H. Reif, Parallel tree contraction, part 1: Fundamentals. In "Advances in Computing Research," vol. 5, JAI Press, 1988, to appear.

[23] G. L. Miller and S.-H. Teng, Dynamic parallel complexity of computational circuits. In Proc. 19th Annual ACM Symposium on the Theory of Computing, 254-263, 1987.

[24] G. L. Miller and S.-H. Teng, Systematic methods for tree based parallel algorithm development. In Proc. Second International Conference on Supercomputing, vol. 2, 392-403, 1987.

[25] S. Miyano, The lexicographically first maximal subgraph problems: P-completeness and NC algorithms. In Proc. 14th International Conference on Automata, Languages, and Programming. Lecture Notes in Computer Science, vol. 267, 425-434, Springer, Berlin, 1987.

[26] K. Mulmuley, A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. In Proc. 18th Annual Symposium on the Theory of Computing, 338-339, 1986.

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

[28] W. L. RUZZO, On uniform circuit complexity, *J. Comput. System Sci.,* **22** (1981), 365–383.

[29] D. Soroker, personal communication.

[30] L. Stockmeyer, Classifying the computational complexity of problems, *J. Symbolic Logic,* **52** (1987), 1-43.

[31] A. Subramanian, A new approach to stable matching problems, Tech. Report STAN-CS-89-1275, Stanford University, 1989.

[32] A. Subramanian, "The Complexity of Circuit Value and Network Stability," Ph.D. dissertation, Stanford University, 1989 (expected).