A

The Complexity of the Comparator Circuit Value Problem

STEPHEN A. COOK, Department of Computer Science, University of Toronto
YUVAL FILMUS, Department of Computer Science, University of Toronto
DAI TRI MAN LÊ, Department of Computer Science, University of Toronto

In 1990 Subramanian defined the complexity class CC as the set of problems log-space reducible to the comparator circuit value problem (Ccv). He and Mayr showed that $\mathsf{NL} \subseteq \mathsf{CC} \subseteq \mathsf{P}$, and proved that in addition to Ccv several other problems are complete for CC, including the stable marriage problem, and finding the lexicographically first maximal matching in a bipartite graph. Although the class has not received much attention since then, we are interested in CC because we conjecture that it is incomparable with the parallel class NC which also satisfies $\mathsf{NL} \subseteq \mathsf{NC} \subseteq \mathsf{P}$, and note that this conjecture implies that none of the CC-complete problems has an efficient polylog time parallel algorithm. We provide evidence for our conjecture by giving oracle settings in which relativized CC and relativized NC are incomparable.

We give several alternative definitions of CC, including (among others) the class of problems computed by uniform polynomial-size families of comparator circuits supplied with copies of the input and its negation, the class of problems AC^0-reducible to Ccv, and the class of problems computed by uniform AC^0 circuits with Ccv gates. We also give a machine model for CC, which corresponds to its characterization as log-space uniform polynomial-size families of comparator circuits. These various characterizations show that CC is a robust class. Our techniques also show that the corresponding function class FCC is closed under composition. The main technical tool we employ is universal comparator circuits.

Other results include a simpler proof of $\mathsf{NL} \subseteq \mathsf{CC}$, a more careful analysis showing the lexicographically first maximal matching problem and its variants are CC-complete under AC^0 many-one reductions, and an explanation of the relation between the Gale–Shapley algorithm and Subramanian's algorithm for stable marriage.

This paper continues the previous work of Cook, Lê and Ye which focused on Cook-Nguyen style uniform proof complexity, answering several open questions raised in that paper.

Categories and Subject Descriptors: F.1.3 [**Complexity Measures and Classes**]: Complexity Hierarchies

General Terms: Theory

Additional Key Words and Phrases: Comparator circuits, P-completeness

ACM Reference Format:
Stephen A. Cook, Yuval Filmus and Dai Tri Man Lê. 2014. The complexity of the comparator circuit value problem. *ACM Trans. Comput. Theory* V, N, Article A (January YYYY), 40 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Comparator circuits are sorting networks [Batcher 1968] in which the wires carry Boolean values. A comparator circuit is presented as a set of *m* horizontal lines, which we call *wires*. The left end

We would like to thank Yuli Ye for his contribution in the early parts of this research. A portion of this work was done when the second and third authors received funding from the [European Community's] Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 238381. This work was also supported by the Natural Sciences and Engineering Research Council of Canada.
Author's addresses: Stephen A. Cook, Department of Computer Science, University of Toronto, Toronto, ON; Yuval Filmus (Current address), School of Mathematics, Institute for Advanced Study, Princeton, NJ.

DOI: http://dx.doi.org/10.1145/0000000.0000000

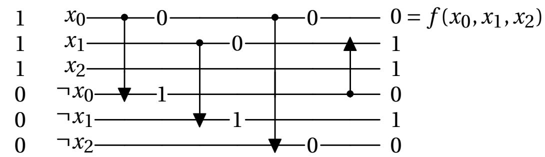


Fig. 1

of each wire is annotated by either a Boolean constant, an input variable, or a negated input variable, and one wire is designated as the output wire. In between the wires there is a sequence of comparator gates, each represented as a vertical arrow connecting some wire w_i with some wire w_j, as shown in Fig. 1. These arrows divide each wire into segments, each of which gets a Boolean value. The values of wires w_i and w_j after the arrow are the maximum and the minimum of the values of wires w_i and w_j right before the arrow, with the tip of the arrow pointing at the position of the maximum. Every wire is initialized (at its left end) by the annotated value, and the output of the circuit is the value at the right end of the output wire. Thus comparator circuits are essentially Boolean circuits in which the gates have restricted fanout.

Problems computed by uniform polynomial size families of comparator circuits form the complexity class CC, defined by Subramanian [1990] in a slightly different guise. Mayr and Subramanian [1992] showed that $\mathsf{NL} \subseteq \mathsf{CC} \subseteq \mathsf{P}$ (where NL is nondeterministic log space), and gave several complete problems for the class, including the stable marriage problem (SM) and lexicographically first maximal matching (LFMM). Known algorithms for these problems are inherently sequential, and so they conjectured that CC is incomparable with NC, the class of problems computable in polylog parallel time. (For the other direction, it is conjectured that the NC^2 problem of raising an $n \times n$ matrix to the nth power is not in CC.) Furthermore, they proposed that CC-hardness be taken as evidence that a problem is not parallelizable.

Since then, other problems have been shown to be CC-complete: the stable roommate problem [Subramanian 1994], the telephone connection problem [Ramachandran and Wang 1991], the problem of predicting internal diffusion-limited aggregation clusters from theoretical physics [Moore and Machta 2000], and the decision version of the hierarchical clustering problem [Greenlaw and Kantabutra 2008]. The maximum weighted matching problem has been shown to be CC-hard [Greenlaw et al. 1995] The fastest known parallel algorithms for some CC-complete problems are listed in [Greenlaw et al. 1995, §B.8].

1.1. Our results

We have two main results. First, we give several new characterizations of the class CC, thus showing that it is a robust class. Second, we give an oracle separation between CC and NC, thus providing evidence for the conjecture that the two classes are incomparable.

1.1.1. Characterizations of CC. Subramanian [1990] defined CC as the class of all languages log-space reducible to the comparator circuit value problem (CCV), which is the following problem: given a comparator circuit with specified Boolean inputs, determine the output value of a designated wire. Cook, Lê and Ye [2011; 2011] considered two other classes, one consisting of all languages AC^0-reducible to CCV, and the other consisting of all languages AC^0-Turing-reducible to CCV, and asked whether these classes are the same as CC. We answer this in the affirmative, and furthermore give characterizations of CC in terms of uniform circuits. Our work gives the following equivalent characterizations of the class CC:

— All languages AC^0-reducible to CCV.
— All languages NL-reducible to CCV.
— All languages AC^0-Turing-reducible to CCV.

— Languages computed by uniform families of polynomial size comparator circuits. (Here *uniform* can be either AC^0-uniform or NL-uniform.)
— Languages computed by uniform families of polynomial size comparator circuits with inverter gates. (An inverter gate has one input and one output wire, and it inverts its input.)

This shows that the class CC is robust. In addition, we show that corresponding function class FCC is closed under composition. The key novel notion for all of these results is our introduction of *universal comparator circuits*.

The characterization of CC as uniform families of comparator circuits also allows us to define CC via certain Turing machines with an implicit access to the input tape.

1.1.2. Oracle separations. Mayr and Subramanian [1992] conjectured that CC and NC are incomparable. We provide evidence supporting this conjecture by separating the relativized versions of these classes, in which the circuits have access to oracle gates. Our techniques also separate the relativized versions of CC and SC, where SC is the class of problems which can be solved simultaneously in polynomial time and polylog space. In fact, it appears that the three classes NC, CC and SC are pairwise incomparable. In particular, although NL is a subclass of both NC and CC, it is unknown whether NL is a subclass of SC.

1.1.3. Other results. The classical Gale–Shapley algorithm for the stable marriage problem [Gale and Shapley 1962] cannot be implemented as a comparator circuit. Subramanian [1994] devised a different fixed-point algorithm which shows that the problem is in CC. We provide an interpretation of his algorithm that highlights its connection to the Gale–Shapley algorithm.

Another fixed-point algorithm is Feder's algorithm for directed reachability (described in Subramanian [1990]), which shows that $\mathsf{NL} \subseteq \mathsf{CC}$. We interpret this algorithm as a form of depth-first search, thus simplifying its presentation and proof. Our exposition follows [Cook et al. 2011; Lê et al. 2011].

As part of proving the CC-completeness of the stable marriage problem, we provide a simple proof that lexicographically first maximal matching (LFMM) is CC-complete. Our exposition again follows [Cook et al. 2011; Lê et al. 2011].

1.2. Background

A fundamental problem in theoretical computer science asks whether every feasible problem can be solved efficiently in parallel. Formally, is $\mathsf{NC} = \mathsf{P}$? It is widely believed that the answer is negative, that is there are some feasible problems which cannot be solved efficiently in parallel. One class of examples consists of P-complete problems, such as the *circuit value problem*. P-complete problems play the same role as that of NP-complete problems in the study of problems solvable in polynomial time:

If $\mathsf{NC} \neq \mathsf{P}$, then P-complete problems are not in NC.

With the goal of understanding which "tractable" problems cannot be solved efficiently in parallel, during the 1980's researchers came up with a host of P-complete problems. Researchers were particularly interested in what aspects of a problem make it inherently sequential. Cook [1985] came up with one such aspect: while the problem of finding a maximal clique in a graph is in NC [Karp, R.M. and Wigderson, A. 1985], if we require the maximal clique to be the one computed by the greedy algorithm (lexicographically first maximal clique), then the problem becomes P-complete. His proof uses a straightforward reduction from the monotone circuit value problem, which is a P-complete restriction of the circuit value problem [Goldschlager 1977].

Anderson and Mayr [Anderson and Mayr 1987] continued this line of research by showing the P-completeness of other problems asking for maximal structures computed by greedy algorithms, such as lexicographically first maximal path. However, one problem resisted their analysis, lexicographically first maximal matching (LFMM), which is the same as lexicographically first independent set in line graphs. While trying to adapt Cook's proof to the case of line graphs, they encountered difficulties simulating fanout. They conjectured that the problem is not P-complete.

Unbeknownst to Anderson and Mayr, Cook had encountered the same problem in 1983 (while working on his paper [Cook 1985]), and was able to come up with a reduction from the comparator circuit value problem (CCV) to LFMM. The hope was that like other variants of the circuit value problem such as the monotone circuit value problem and the planar circuit value problem, CCV would turn out to be P-complete. (The planar monotone circuit value problem, however, is in NC [Yang 1991; Delcher and Kosaraju 1995; Ramachandran and Yang 1996].)

The foregoing prompted Ashok Subramanian (whose advisor was Mayr) to study the relative strength of variants of the circuit value problem restricted by the set of allowed gates. Specifically, Subramanian was interested in the significance of fanout. To that end, Subramanian considered circuits without fanout, but instead allowed multi-output gates such as the COPY gate which takes one input x and outputs two copies of x.

Subramanian classified the relative strength of the circuit value problem when the given set of gates can be used to simulate COPY: there are seven different cases, and in each of them the circuit value problem is either in NC or P-complete. The case when the given set of gates cannot simulate COPY is more interesting: if all gates are monotone then the circuit value problem is either in NC or CC-hard [Subramanian 1990, Corollary 5.22]; for the non-monotone case there is no complete characterization. The class CC thus emerges as a natural "minimal" class above NC for the monotone case.

Cook, Lê and Ye [2011; 2011] constructed a uniform proof theory VCC (in the style of Cook and Nguyen [2010]) which corresponds to CC, and showed that Subramanian's results are formalizable in the theory. The present paper answers several open questions raised in [Cook et al. 2011; Lê et al. 2011].

Paper organization. In Section 2 we introduce definitions of basic concepts, including comparator circuits and the comparator circuit value problem. We also introduce several equivalent definitions of CC, in terms of uniform comparator circuits, and in terms of problems reducible (in various ways) to the comparator circuit value problem.

In Section 3 we construct universal comparator circuits. These are comparator circuits which accept as input a comparator circuit C and an input vector Y, and compute the output wires of C when run on input Y. As an application, we prove that the various definitions of CC given at the end of the preceding section are indeed equivalent, and we provide yet another definition in terms of restricted Turing machines.

In Section 4 we define a notion of relativized CC and prove that relativized CC is incomparable with relativized NC. This of course implies that relativized CC is strictly contained in relativized P. We also argue that CC and SC might be incomparable.

In Section 5 we prove that the lexicographically first maximal matching problem and its variants are complete for CC under AC^0 many-one reductions. (Two of the present authors made a similar claim in [Lê et al. 2011], but that proof works for log space reductions rather than AC^0 reductions.)

In Section 6 we show that the stable marriage problem is complete for CC, using Subramanian's algorithm [Subramanian 1990; Subramanian 1994]. We show that Subramanian's fixed-point algorithm, which uses three-valued logic, is related to the Gale–Shapley algorithm via an intermediate *interval algorithm*. The latter algorithm also explains the provenance of three-valued logic: an interval partitions a person's preference list into *three* parts, so we use three values $\{0,*,1\}$ to encode these three parts of a preference list.

Section 7 includes some open problems and our concluding remarks.

2. PRELIMINARIES

2.1. Notation

We use lower case letters, e.g. $x, y, z, \ldots$, to denote unary arguments and upper case letters, e.g. $X, Y, Z, \ldots$, to denote binary string arguments. For a binary string X, we write $|X|$ to denote the length of X, and $X(i)$ to denote the ith bit of X.

2.2. Function classes and search problems

A complexity class consists of relations $R(X)$, where X is a binary string argument. Given a class of relations C, we associate a class FC of functions $F(X)$ with C as follows. We require these functions to be p-bounded, i.e., $|F(X)|$ is bounded by a polynomial in $|X|$. Then we define FC to consist of all p-bounded string functions whose bit graphs are in C. (Here the *bit graph* of $F(X)$ is the relation $B_F(i, X)$ which holds iff the ith bit of $F(X)$ is 1.) For all classes C of interest here the function class FC is closed under composition. In particular, FNL is closed under composition because NL (nondeterministic log space) is closed under complementation.

Most of the computational problems we consider here can be expressed as decision problems (i.e. relations). However, the stable marriage problem is an exception, because in general a given instance has more than one solution (i.e. there is more than one stable marriage). Thus, the problem is properly described as a search problem. A *search problem* Q_R is a multivalued function with graph $R(X, Z)$, so $Q_R(X) = \{Z \mid R(X, Z)\}$.

The search problem is *total* if the set $Q_R(X)$ is non-empty for all X. The search problem is a *function problem* if $|Q_R(X)| = 1$ for all X. A function $F(X)$ *solves* Q_R if $F(X) \in Q_R(X)$ for all X. We will be concerned only with total search problems in this paper.

2.3. Reductions

Let C be a complexity class. A relation $R_1(X)$ is C many-one reducible to a relation $R_2(Y)$ (written $R_1 \leq^{\mathsf{C}}_{\mathsf{m}} R_2$) if there is a function F in FC such that $R_1(X) \leftrightarrow R_2(F(X))$.

A search problem $Q_{R_1}(X)$ is C many-one reducible to a search problem $Q_{R_2}(Y)$ if there are functions G, F in FC such that $G(X, Z) \in Q_{R_1}(X)$ for all $Z \in Q_{R_2}(F(X))$.

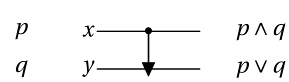
Recall that problems in AC^0 are computed by uniform families of polynomial size constant depth circuits. (See [Barrington et al. 1990] for many equivalent definitions of uniform AC^0, including 'First Order definable'.) We are interested in AC^0 many-one reducibility, but also in the more general notion of AC^0 Turing reducibility. We say that a relation $R_1(X)$ is AC^0 Turing reducible to a relation $R_2(Y)$ if $R_1(X)$ is is computed by some AC^0-uniform family of AC^0 circuits which are equipped with R_2-oracle gates. Such an oracle gate takes a string Y as input and outputs a single bit $R_2(Y)$, and is considered to have depth 1 when counting the overall depth of the circuit. We note that standard small complexity classes, including AC^0, TC^0, NC^1, NL and P, are closed under AC^0 Turing reductions.

In general we say that a family $\langle C_n \rangle_n$ of circuits is *uniform* if the description $D(n)$ of C_n is a uniform AC^0 function of n.

2.4. Comparator circuits

A *comparator gate* is a function $C\colon \{0,1\}^2 \to \{0,1\}^2$ that takes an input pair (p, q) and outputs a pair $(p \wedge q, p \vee q)$. Intuitively, the first output in the pair is the smaller bit among the two input bits p, q, and the second output is the larger bit.

We will use the graphical notation on the right to denote a comparator gate, where x and y denote the names of the wires, and the direction of the arrow denotes the direction to which we move the larger bit as shown in the picture.



A *comparator circuit* consists of m wires and a sequence $(i_1, j_1), \ldots, (i_n, j_n)$ of n comparator gates. We allow "dummy" gates of the form (i, i), which do nothing. A comparator circuit computes a function $f\colon \{0,1\}^m \to \{0,1\}^m$ in the obvious way (See Section 1). A *comparator circuit with negation gates* additionally has negation gates $N\colon \{0,1\} \to \{0,1\}$ which invert their input.

An *annotated comparator circuit* consists of a comparator circuit with a distinguished output wire together with an annotation of the input of each wire by an input bit x_i, the negation of an input bit $\neg x_i$, or a constant 0 or 1. A *positively annotated comparator circuit* has no annotations of the type $\neg x_i$. An annotated comparator circuit computes a function $f\colon \{0,1\}^k \to \{0,1\}$, where k is the largest index appearing in an input annotation x_k.

An AC^0-*uniform* family of annotated comparator circuits is one whose circuit descriptions are computable by a function in FAC^0. An NL-*uniform* family of annotated comparator circuits is one whose circuits are computable in FNL.

Uniform annotated comparator circuits will serve as the basis of some definitions of the complexity class CC. Other definitions will be based on a complete problem, the *comparator circuit value problem.*

Definition 2.1. The comparator circuit value problem (CCV) is the decision problem: given a comparator circuit and an assignment of bits to the input of each wire, decide whether a designated wire outputs one. By default, we often let the designated wire be the 0th wire of a circuit.

Comparator circuits can have some gates pointing up, and others pointing down. The next result shows that in most of our proofs there is no harm in assuming that all gates point in the same direction.

PROPOSITION 2.2. CCV *is* AC^0 *many-one reducible to the special case in which all comparator gates point down (or all point up).*

PROOF. Suppose we have a gate on the left of Fig. 2 with the arrow pointing upward. We can construct a circuit that outputs the same values as those of x and y, but all the gates will now point downward as shown on the right of Fig. 2.

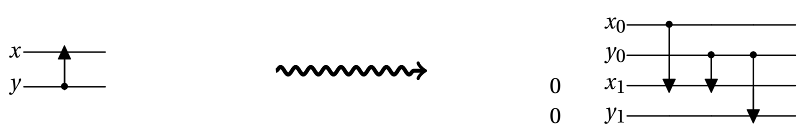


Fig. 2

It is not hard to see that the wires x_1 and y_1 in this new comparator circuit will output the same values as the wires x and y respectively in the original circuit. For the general case, we can simply make copies of all wires for each layer of the comparator circuit, where each copy of a wire will be used to carry the value of a wire at a single layer of the circuit. Then apply the above construction to simulate the effect of each gate. Note that additional comparator gates are also needed to forward the values of the wires from one layer to another, in the same way that we use the gate $\langle y_0, y_1\rangle$ to forward the value carried in wire y_0 to wire y_1 in the above construction.

To carry this out in AC^0, one way would be to add a complete copy of all wires for every comparator gate in the original circuit. Each new wire has input 0. For each original gate g, first put in gates copying the values to the new wires. If g points down, put in a copy of g connecting the new wires, and if g points up, put in the construction in Fig. 2. □

The class CC has many equivalent definitions. We adopt one of them as the "official" definition, and all the rest are shown to be equivalent in Section 3.

Definition 2.3.

— CC is the class of relations computed by an AC^0-uniform family of polynomial size annotated comparator circuits.

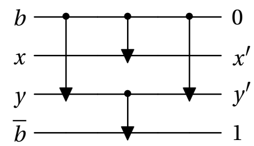


Fig. 3: Conditional comparator gadget

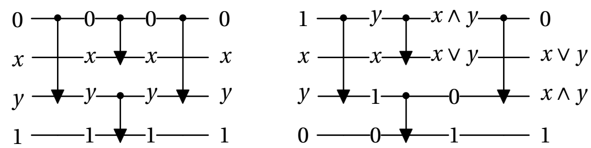


Fig. 4: Operation of conditional comparator gadget

— CC¬ is the class of relations computed by an AC^0-uniform family of polynomial size positively annotated comparator circuits with negation gates.
— AC^0-Ccv is the class of relations AC^0-many-one reducible to Ccv.
— NL-Ccv is the class of relations NL-many-one reducible to Ccv.
— $(\mathsf{AC}^0)^{\text{Ccv}}$ is the class of relations AC^0-Turing reducible to Ccv.

3. UNIVERSAL COMPARATOR CIRCUITS

An annotated comparator circuit $\mathsf{UNIV}_{m,n}$ is *universal* if it can simulate any comparator circuit with at most m wires and n gates, as precisely stated in the following theorem. Here we present the first known (polynomial size) construction of such a universal comparator circuit family. For simplicity, our universal circuit only simulates unannotated comparator circuits, but the idea extends to cover annotated comparator circuits as well.

THEOREM 3.1. *There is an* AC^0*-uniform family of annotated comparator circuits* $\mathsf{UNIV}_{m,n}$ *which satisfy the following property: for any comparator circuit* C *having at most* m *wires and* n *gates, the designated output wire of* $\mathsf{UNIV}_{m,n}$ *fed with inputs* C, Y *equals the* 0*th output wire of* C *fed with input* Y.

PROOF. The key idea is a *gadget* consisting of a comparator circuit with four wires and four gates which allows a conditional application of a comparator gate to two of its inputs x, y, depending on whether a control bit b is 0 or 1. The other two inputs are $\overline{b}$ and b (see Fig. 3). The gate is applied only when $b = 1$ (see Fig. 4).

In order to simulate a single arbitrary gate in a circuit with m wires we put in $m(m-1)$ gadgets in a row, for the $m(m-1)$ possible gates. Simulating n gates requires $m(m-1)n$ gadgets. The bits of C are the control bits for the gadgets. The resulting circuit can be constructed in an AC^0-uniform fashion. □

As a consequence, we can identify the classes CC and AC^0-Ccv.

LEMMA 3.2. *The two complexity classes* CC *and* AC^0-Ccv *are identical.*

PROOF. We start with the easy direction $\mathsf{CC} \subseteq \mathsf{AC}^0$-Ccv. Suppose R is a relation computed by AC^0-uniform polynomial size annotated comparator circuits C_m. Given an input X of length m, we can construct in uniform AC^0 the circuit C_m and replace each input annotation with the corresponding constant, bit of X or its negation. The value of R is computed by applying Ccv to this data.

We proceed to show that AC^0-Ccv $\subseteq \mathsf{CC}$. Our first observation is that every AC^0 circuit can be converted to a polynomial size formula, and so to a comparator circuit. Every relation R in

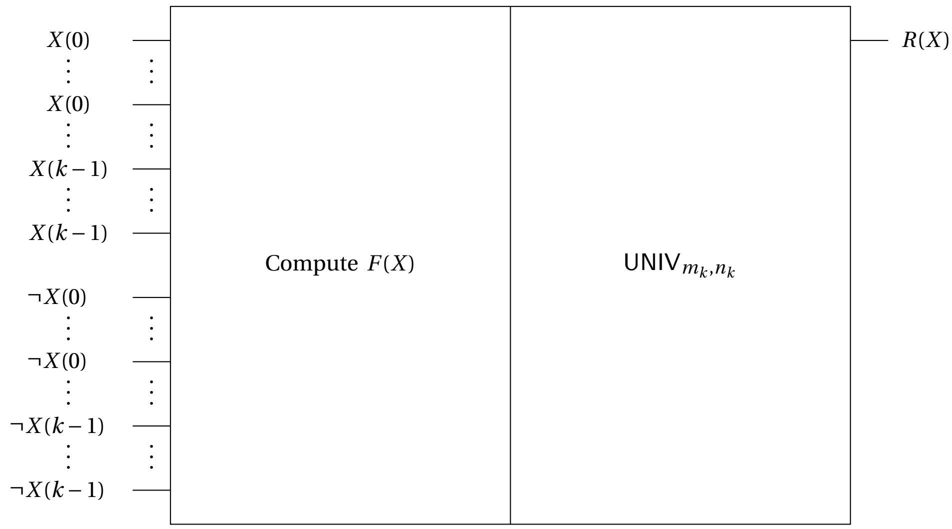


Fig. 5: The construction of a comparator circuit for any relation $R \in \mathsf{AC}^0$-CCV for a fixed input length k.

AC^0-CCV is given by a uniform AC^0 reduction F, such that $R(X) = 1$ if and only if CCV$(F(X)) = 1$. Suppose that the circuit computed by $F(X)$ on an input of size k has m_k wires and n_k comparator gates. We construct an annotated comparator circuit computing $R(X)$ by first computing $F(X)$ and then feeding the result into UNIV_{m_k,n_k} (see Fig. 5). The resulting circuit is AC^0-uniform. □

The preceding construction replaces an arbitrary AC^0 function F with a corresponding comparator circuit. By allowing stronger functions F, we get additional characterizations of CC. In particular, Feder [Subramanian 1990] showed that $\mathsf{NL} \subseteq \mathsf{CC}$, and this allows us to replace AC^0-many-one reductions with NL-many-one reductions. For the sake of completeness, we first reproduce a simple proof that $\mathsf{NL} \subseteq \mathsf{CC}$ taken from [Lê et al. 2011].

THEOREM 3.3 (FEDER [SUBRAMANIAN 1990]). $\mathsf{NL} \subseteq \mathsf{CC}$.

PROOF. Each instance of the REACHABILITY problem consists of a directed acyclic graph $G = (V, E)$, where $V = \{u_0, \ldots, u_{n-1}\}$, and we want to decide if there is a path from u_0 to u_{n-1}. It is well-known that REACHABILITY is NL-complete. Since a directed graph can be converted by an AC^0 function into a layered graph with an equivalent reachability problem, it suffices to give a comparator circuit construction that solves instances of REACHABILITY satisfying the following assumption:

$$\text{The graph } G \text{ only has directed edges of the form } (u_i, u_j), \text{ where } i < j. \tag{3.1}$$

The following construction from [Lê et al. 2011] for showing that $\mathsf{NL} \subseteq \mathsf{CC}$ is simpler than the one in [Subramanian 1990; Mayr and Subramanian 1992]. Moreover, it reduces REACHABILITY to CCV directly without going through some intermediate complete problem, and this was stated as an open problem in [Subramanian 1990, Chapter 7.8.1]. The idea is to perform a depth-first search of the nodes reachable from the source node by successively introducing n pebbles into the source, and sending each pebble along the lexicographically last pebbled path until it reaches an unpebbled node, where it remains. After n iterations, all nodes reachable from the source are pebbled, and we can check whether the target is one of them.

We will demonstrate the construction through a simple example, where we have the directed graph in Fig. 6 satisfying the assumption (3.1). We will build a comparator circuit as in Fig. 7, where the wires $v_0, \dots, v_4$ represent the vertices $u_0, \dots, u_4$ of the preceding graph and the wires $\iota_0, \dots, \iota_4$ are used to feed 1-bits into the wire v_0, and from there to the other wires v_i reachable from v_0. We let every wire ι_i take input 1 and every wire v_i take input 0.

We next show how to construct the gadget contained in each box. For a graph with n vertices ($n = 5$ in our example), the k^{th} gadget is constructed as follows:

Introduce a comparator gate from wire ι_k to wire v_0
for $i = 0, \dots, n-1$ **do**
  for $j = i+1, \dots, n-1$ **do**
    Introduce a comparator gate from v_i to v_j if $(u_i, u_j) \in E$, or a dummy gate on v_i otherwise.
  end for
end for

Note that the gadgets are identical except for the first comparator gate.

We only use the loop structure to clarify the order the gates are added. The construction can easily be done in AC^0 since the position of each gate can be calculated exactly, and thus all gates can be added independently from one another. Note that for a graph with n vertices, we have at most n vertices reachable from a single vertex, and thus we need n gadgets as described above. In our example, there are at most 5 wires reachable from wire v_0, and thus we utilize the gadget 5 times.

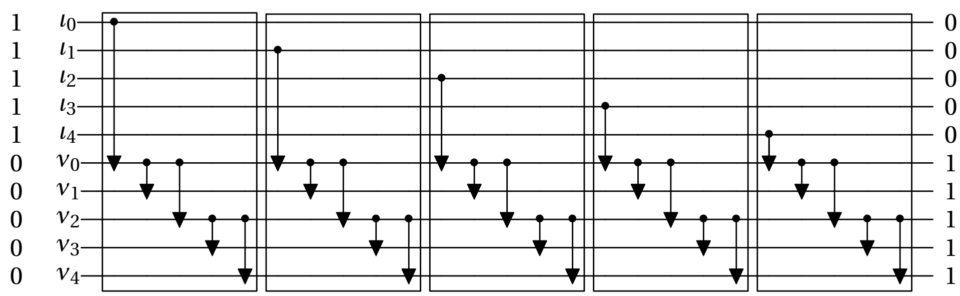


Fig. 7: A comparator circuit that solves REACHABILITY. (The dummy gates are omitted.)

The successive gadgets in the boxes each introduce a 'pebble' (i.e. a 1 bit) which ends up at the next node in the depth-first search (i.e. its wire will now carry 1) and is thus excluded from the search of the gadgets that follow. For example, the gadget from the left-most dashed box in Fig. 7 will move a value 1 from wire ι_0 to wire v_0 and from wire v_0 to wire v_1. This essentially "marks" the wire v_1 since we cannot move this value 1 away from v_1, and thus v_1 can no longer receive any new incoming 1. Hence, the gadget from the second box in Fig. 7 will repeat the process of finding the lex-first maximal path from v_0 to the remaining (unmarked) vertices. These searches end when all vertices reachable from v_0 are marked. □

An immediate corollary of $\mathsf{NL} \subseteq \mathsf{CC}$ is the equivalence of the complexity classes AC^0-CCV and NL-CCV.

LEMMA 3.4. *The two complexity classes* AC^0-CCV *and* NL-CCV *are identical.*

PROOF. It is clear that $\mathsf{AC}^0\text{-CCV} \subseteq \mathsf{NL}\text{-CCV}$. For the other direction, let $R \in \mathsf{NL}$-CCV be computed as $R(X) = \text{CCV}(F(X))$, where $F \in \mathsf{NL}$. Theorem 3.3 shows that $F \in \mathsf{CC}$. We can therefore use the construction of Lemma 3.2. □

Another characterization is via AC^0-Turing reductions. We require a preliminary lemma corresponding to de Morgan's laws.

LEMMA 3.5. *Given an annotated comparator circuit C computing a function $F(X)$, we can construct an annotated comparator circuit C' computing the function $\neg F(X)$ with the same number of wires and comparator gates. Furthermore, C' can be constructed from C in* AC^0.

PROOF. The idea is to push negations to the bottom using de Morgan's laws. Given C, construct C' by negating all inputs (switch 0 and 1, x_i and $\neg x_i$) and flipping all comparator gates: replace each gate $\langle i, j\rangle$ with a gate $\langle j, i\rangle$ which has the same inputs but the reversed outputs. An easy induction shows that $C'(X) = \neg C(X)$. □

LEMMA 3.6. *The two complexity classes* AC^0-CCV *and* $(\mathsf{AC}^0)^{\text{CCV}}$ *are identical.*

PROOF. It is clear that AC^0-CCV $\subseteq (\mathsf{AC}^0)^{\text{CCV}}$. For the other direction, let $R \in (\mathsf{AC}^0)^{\text{CCV}}$ be computed by a uniform AC^0 family of circuits C_k with CCV oracle gates. The basic idea is to replace each oracle gate with the corresponding universal comparator circuit (Theorem 3.1). However, a universal comparator circuit expects each of its inputs to appear polynomially many times, some of them negated. We will handle this by duplicating the corresponding portion of the constructed circuit enough times, using Lemma 3.5 to handle negations. The resulting circuit will be polynomial size since C_k has constant depth.

We proceed with the details. For each gate g in the circuit C_k, we define a transformation $T(g)$ mapping it to an annotated comparator circuit computing the same function, as follows. If $g = x_i$ then $T(g)$ is a circuit with one wire annotated x_i. If $g = \neg g_1$ then $T(g)$ is obtained from $T(g_1)$ via Lemma 3.5. If $g = g_1 \vee \cdots \vee g_\ell$ or $g = g_1 \wedge \cdots \wedge g_\ell$ then $T(g)$ is obtained by constructing fresh copies of $T(g_1), \dots, T(g_\ell)$ and joining them with $\ell - 1$ comparator gates. If g is an oracle gate with inputs $g_1, \dots, g_\ell$ then we take the universal comparator circuit from Theorem 3.1 and replace each input bit by the corresponding copy of $T(g_i)$ or of its negation obtained via Lemma 3.5.

If r is the root gate of C_k then it is clear that $T(r)$ is an annotated comparator circuit computing the same function as C_k. It remains to estimate the size of $T(r)$, and for that it is enough to count the number of comparator gates $|T(r)|$. Suppose that C_k has n gates and depth d, where n is polynomial in k and d is constant. For $0 \le \Delta \le d$, we compute a bound B_Δ on $|T(g)|$ for a gate g at depth $d - \Delta$. If $d = 0$ then $|T(g)| = 0$, and so we can take $B_0 = 0$. For $d = \Delta + 1 \neq 0$, the costliest case is when g is an oracle gate. Suppose the inputs to g are $g_1, \dots, g_\ell$ of depth at least $d - \Delta$; note that $\ell \le n$. We construct $T(g)$ by taking $\ell^{O(1)}$ copies of $T(g_i)$ and appending to them a comparator circuit of size $\ell^{O(1)}$. Therefore $|T(g)| \le \ell^{O(1)}(B_\Delta + 1)$ and so $B_{\Delta+1} \le n^{O(1)}(B_\Delta + 1)$. Solving the recurrence, we deduce $B_d \le n^{O(d)}$ and so $|T(r)| \le B_d$ is at most polynomial in k (since $n = k^{O(1)}$ and $d = O(1)$).

To complete the proof, we observe that $T(r)$ can be computed in AC^0, and so $(\mathsf{AC}^0)^{\text{CCV}} \subseteq \mathsf{CC} = \mathsf{AC}^0$-CCV (using Lemma 3.2). □

Finally, we show that the same class is obtained if we allow negation gates, using a reduction outlined in Section 5.4.

LEMMA 3.7. *The two complexity classes* CC *and* $\mathsf{CC}\neg$ *are identical.*

PROOF. Recall that CC consists of relations computed by uniform annotated comparator circuits, while $\mathsf{CC}\neg$ consists of relations computed by uniform positively annotated comparator circuits with negation gates. It is easy to see that $\mathsf{CC} \subseteq \mathsf{CC}\neg$: every negated input $\neg x_i$ can be replaced by the corresponding positive input followed by a negation gate. For the other direction, Section 5.4 shows how to simulate a comparator circuit with negation gates using a standard comparator circuit. □

In summary we have shown that all complexity classes listed at the end of Section 2.4 are identical.

THEOREM 3.8. *All complexity classes* $\mathsf{CC}, \mathsf{CC}\neg, \mathsf{AC}^0$-CCV, NL-CCV, $(\mathsf{AC}^0)^{\text{CCV}}$ *are identical.*

Subramanian [1990] originally defined CC as the class of relations log-space reducible to CCV. Theorem 3.8 shows that this class is identical to the one we consider in the present paper.

Our methods can also be used to show that the function class FCC corresponding to CC is closed under composition. A function F mapping strings of length k to strings of polynomial length $\ell(k)$ is in FCC if the the bit graph relation $B_F(i, X) \leftrightarrow F(X)(i)$ is in CC.

THEOREM 3.9. *The class* FCC *is closed under composition.*

PROOF. Suppose that the bit graphs $B_F, B_G \in$ FCC are given by uniform positively annotated comparator circuits with negation gates. We describe how to compute $B_{F\circ G}$ using uniform positively annotated comparator circuits with negation gates, where $(F \circ G)(X) = F(G(X))$. For a given input length k, let $\ell(k)$ be the length of $G(X)$ for X of length k, and consider the circuit for $B_F(i, Y)$ for inputs Y of length $\ell(k)$. Prepend to each input wire annotated $Y(j)$ a copy of the circuit for $B_G(j, X)$ (for inputs of length k) in which the first parameter is hard coded to be j. The result is a uniform positively annotated comparator circuit with negation gates computing $B_{F\circ G}(i, X)$. □

Theorem 3.8 implies that CC is the class of all relations computed by L-uniform positively annotated comparator circuits with negation gates. This characterization corresponds to the following Turing machine model.

THEOREM 3.10. *Every relation in* CC *is computable by a Turing machine of the following type. The machine has three tapes: a work tape with one head W, an input tape with one head I, and a 'comparator' tape with two heads M_1, M_2. The work tape is initalized by the size of the input n encoded in binary, and is limited to $O(\log n)$ cells. The input tape is initialized with the input, and the comparator tape is initially blank. Additionally, the machine has a state q out of a finite set of states Q, and starts at some starting state $s \in Q$.*

At each step, the machine reads the contents of the work tape at the position of W, and depending upon the contents and the current state q, it changes the current state q, writes a symbol on the work tape at the position of W, optionally moves each of the heads W, I, M_1, M_2 one step to the left or one step to the right (separately for each head), and optionally performs one of the following instructions:

— *Write a blank cell M_1 (or M_2) with* 0 *(or* 1*).*
— *Copy the value at cell I to a blank cell M_1 (or M_2).*
— *Negate the value at cell M_1 (or M_2).*
— *"Sort" the values at cells M_1 and M_2.*
— *Output the value at cell M_1 (or M_2), and halt the machine.*

Conversely, every relation computed by such a machine is in CC.

PROOF. For every relation $R \in$ CC there is a log-space machine T that on input n outputs a positively annotated comparator circuit with negation gates computing R on inputs of length n. We can convert T to a machine T' of the type described in the theorem by thinking of each cell of the comparator tape as one wire in the circuit. We replace each annotation by the second or third special operation and each gate by one of the following two special operations. The final special operation is used to single out the distinguished output wire.

For the other direction, given a machine T' of the type described we can construct a log-space machine T which on input n outputs a suitable annotated comparator circuit for inputs of length n as follows: T on input n runs T' with its work tape initiated to n in binary, and uses the control signals to the input and comparator tapes to describe the circuit. □

This machine model is resilient under the following changes: allowing more heads on each of the tapes, allowing the second or third special operations to write over a non-blank cell, and adding other operations such as swapping the values of two cells in the comparator tape. We leave the proof to the reader.

4. ORACLE SEPARATIONS

Here we support our conjecture that the complexity classes NC and CC are incomparable by defining and separating relativized versions of the classes. (See Section 1 for a discussion of the conjecture.) Problems in relativized CC are computed by comparator circuits which are allowed to have oracle gates, as well as comparator gates and $\neg$ gates. (We allow $\neg$ gates to make our results more general. Note that $\neg$ gates can be eliminated from nonrelativized comparator circuits, as explained in Section 5.4.) Each oracle gate computes some function $G\colon \{0,1\}^n \to \{0,1\}^n$ for some n. We can insert such an oracle gate anywhere in an oracle comparator circuit with m wires, as long as $m \geq n$, by selecting a level in the circuit, selecting any n wires and using them as inputs to the gate (so each gate input gets one of the n distinct wires), and then the n outputs feed to some set of n distinct output wires. Note that this definition preserves the limited fan-out property of comparator circuits: each output of a gate is connected to at most one input of one other gate.

We are interested in oracles $\alpha\colon \{0,1\}^* \to \{0,1\}^*$ which are length-preserving, so $|\alpha(Y)| = |Y|$. We use the notation α_n to refer to the restriction of α to $\{0,1\}^n$. We define the relativized complexity class $\mathsf{CC}(\alpha)$ based on the circuit-family characterization of $\mathsf{CC}\neg$ given in Definition 2.3. Thus a relation $R(X,\alpha)$ is in $\mathsf{CC}(\alpha)$ iff it is computed by a polynomial size family of annotated comparator circuits which are allowed comparator gates, $\neg$-gates, and α_n oracle gates, where $n = |X|$. We consider both a uniform version (in which each circuit family satisfies a uniformity condition) and a nonuniform version of $\mathsf{CC}(\alpha)$.

Analogous to the above, we define the relativized class $\mathsf{NC}^k(\alpha)$ to be the class of relations $R(X,\alpha)$ computed by some family of depth $O(\log^k n)$ polynomial size Boolean circuits with $\wedge$, $\vee$, $\neg$, and α_n-gates (where $n = |X|$) in which $\wedge$-gates and $\vee$-gates have fan-in at most two, and oracle gates are nested at most $O(\log^{k-1} n)$ levels deep[1]. As above, we consider both uniform and non-uniform versions of these classes. Also $\mathsf{NC}(\alpha) = \bigcup_k \mathsf{NC}^k(\alpha)$.

As observed earlier, flipping one input of a comparator gate flips exactly one output. We can generalize this notion to oracles α as follows.

Definition 4.1. A partial function $\alpha : \{0,1\}^* \rightharpoonup \{0,1\}^*$ which is length-preserving on its domain is *(weakly) 1-Lipschitz* if for all strings X, X' in the domain of α, if $|X| = |X'|$ and X and X' have Hamming distance 1, then $\alpha(X)$ and $\alpha(X')$ have Hamming distance *at most* 1.

A partial function $\alpha : \{0,1\}^* \rightharpoonup \{0,1\}^*$ which is length-preserving on its domain is *strictly 1-Lipschitz* if for all strings X, X' in the domain of α, if $|X| = |X'|$ and X and X' have Hamming distance 1, then $\alpha(X)$ and $\alpha(X')$ have Hamming distance *exactly* 1.

Subramanian [1990] uses a slightly different terminology: (strictly) adjacency-preserving for (strictly) 1-Lipschitz.

Since comparator gates compute strictly 1-Lipschitz functions, it may seem reasonable to restrict comparator oracle circuits to strictly 1-Lipschitz oracles α. Our separation results below hold whether or not we make this restriction.

Roughly speaking, we wish to prove $\mathsf{CC}(\alpha) \not\subseteq \mathsf{NC}(\alpha)$ and $\mathsf{NC}^3(\alpha) \not\subseteq \mathsf{CC}(\alpha)$. More precisely, we have the following result.

THEOREM 4.2.

(i) *There is a relation $R_1(\alpha)$ which is computed by some uniform polynomial size family of comparator oracle circuits, but which cannot be computed by any $\mathsf{NC}(\alpha)$ circuit family (uniform or not), even when the oracle α is restricted to be strictly 1-Lipschitz.*

[1]Cook [1985] defines the depth of an oracle gate of fan-in m to be $\log m$. Our definition follows the one by Aehlig at al. [2007], and has the advantage that it preserves classical results like $\mathsf{NC}^1(\alpha) \subseteq \mathsf{L}(\alpha)$. The class $\mathsf{NC}(\alpha)$ is the same under both definitions.

(ii) There is a relation $R_2(\alpha)$ which is computed by some uniform $\mathsf{NC}^3(\alpha)$ circuit family which cannot be computed by any polynomial size family of comparator oracle circuits (uniform or not), even when the oracle α is restricted to be strictly 1-Lipschitz.

The proof of Theorem 4.2(ii) only uses the fact that comparator gates are 1-Lipschitz, and therefore the lower bound also applies to the more general class APC considered by Subramanian [1990] in which comparator gates are replaced by arbitrary 1-Lipschitz gates. Conversely, the comparator circuit in the proof of Theorem 4.2(i) uses only oracle gates, and so the relation $R_1(\alpha)$ belongs to any circuit class whose depth is unrestricted.

The restriction to strictly 1-Lipschitz oracles might seem severe. However the following lemma shows how to extend every 1-Lipschitz function to a strictly 1-Lipschitz function. As a result, it is enough to prove a relaxed version of Theorem 4.2 where the oracles are restricted to be only (weakly) 1-Lipschitz.

LEMMA 4.3. *Suppose f is a 1-Lipschitz function. Define $g(X)$ by adding a 'parity bit' in front of $f(X)$ as follows:*

$$g(X) = \left[\text{parity}(X) \oplus \text{parity}(f(X))\right] f(X).$$

Then g is strictly 1-Lipschitz.

PROOF. Suppose $d(X,Y) = 1$. Clearly $d(g(X), g(Y)) \leq 2$. On the other hand, parity$(g(X))$ = parity(X), and hence $d(g(X), g(Y))$ is odd. We conclude that $d(g(X), g(Y)) = 1$. □

Given a relation $S(\beta)$ which separates two relativized complexity classes even when the oracle β is restricted to 1-Lipschitz functions, define a new relation $R(\alpha) = S(\text{chop}(\alpha))$, where $\text{chop}(\alpha)_n(X)$ results from $\alpha_{n+1}(0X)$ by chopping off the leading bit. Given an oracle β which is 1-Lipschitz, define a new oracle α by

$$\alpha_{n+1}(bX) = \left[\text{parity}(bX) \oplus \text{parity}(\beta_n(X))\right] \beta_n(X).$$

Lemma 4.3 shows that α is strictly 1-Lipschitz. Notice that $R(\alpha) = S(\beta)$. Hence $R(\alpha)$ separates the two relativized complexity classes even when the oracle α is restricted to strictly 1-Lipschitz functions.

Henceforth we will prove the relaxed version of Theorem 4.2 in which the oracles are only restricted to be (weakly) 1-Lipschitz.

4.1. Proof that $\mathsf{CC}(\alpha)$ is not contained in $\mathsf{NC}(\alpha)$

4.1.1. Proof without restricting the oracles to be 1-Lipschitz. It turns out that item (i) of Theorem 4.2 is easy to prove if we require the $\mathsf{NC}(\alpha)$ circuit family to work on all length-preserving oracles α, and not just 1-Lipschitz oracles. This is a consequence of the next proposition, which follows from the proof of [Aehlig et al. 2007, Theorem 14], and states that the ℓth iteration of an oracle requires a circuit with oracle nesting depth ℓ to compute.

Definition 4.4. The *nesting depth* of an oracle gate G in an oracle circuit is the maximum number of oracle gates (counting G) on any path in the circuit from an input (to the circuit) to G.

PROPOSITION 4.5. *Let $d, n > 0$ and let $C(\alpha)$ be a circuit with any number of Boolean gates but with fewer than 2^n α_n-gates such that the nesting depth of any α_n-gate is at most d. If the circuit correctly computes the first bit of α_n^ℓ (the ℓth iteration of α_n), and this is true for all oracles α_n, then $\ell \leq d$.*

The proof of Proposition 4.5 appears below. We apply the proposition with $d = n$, and conclude that the first bit of α_n^n cannot be computed in $\mathsf{NC}(\alpha)$. But α_n^n obviously can be computed in $\mathsf{CC}(\alpha)$ by placing n oracle gates α_n in series. This proves item (i) of Theorem 4.2 without the 1-Lipschitz restriction.

For the proof of Proposition 4.5 we use the following definition and lemma from [Aehlig et al. 2007].

Definition 4.6. A partial function $f\colon \{0,1\}^n \rightharpoonup \{0,1\}^n$ is called ℓ*-sequential* if (abbreviating 0^n by $\mathbf{0}$)

$$\mathbf{0}, f(\mathbf{0}), f^2(\mathbf{0}), \ldots, f^\ell(\mathbf{0})$$

are all defined, but $f^\ell(\mathbf{0}) \notin \mathrm{dom}(f)$.

Note that in Definition 4.6 it is necessarily the case that $\mathbf{0}, f(\mathbf{0}), f^2(\mathbf{0}), \ldots, f^\ell(\mathbf{0})$ are distinct.

LEMMA 4.7. *Let $n \in \mathbb{N}$ and $f\colon \{0,1\}^n \rightharpoonup \{0,1\}^n$ be an ℓ-sequential partial function. Let $M \subset \{0,1\}^n$ be such that $|\mathrm{dom}(f) \cup M| < 2^n$. Then there is an $(\ell+1)$-sequential extension $f' \supseteq f$ with $\mathrm{dom}(f') = \mathrm{dom}(f) \cup M$.*

PROOF. Let $Y \in \{0,1\}^n \setminus \big(M \cup \mathrm{dom}(f)\big)$. Such a Y exists by our assumption on the cardinality of $M \cup \mathrm{dom}(f)$. Let f' be f extended by setting $f'(x) = Y$ for all $X \in M \setminus \mathrm{dom}(f)$. This f' is as desired.

Indeed, assume that $\mathbf{0}, f'(\mathbf{0}), \ldots, f'^{\ell+1}(\mathbf{0}), f'^{\ell+2}(\mathbf{0})$ are all defined. Then, since $Y \notin \mathrm{dom}(f')$, it follows that all the $\mathbf{0}, f'(\mathbf{0}), \ldots, f'^{\ell+1}(\mathbf{0})$ have to be different from Y. Hence these values have already been defined in f. But this contradicts the assumption that f was ℓ-sequential. □

PROOF OF PROPOSITION 4.5. We use f to stand for the oracle function α_n. Assume that such a circuit computes $f^\ell(\mathbf{0})$ correctly for all oracles. We have to find a setting for the oracle that witnesses $\ell \leq d$.

By induction on $k \geq 0$ we define partial functions $f_k\colon \{0,1\}^n \rightharpoonup \{0,1\}^n$ with the following properties.

— $f_0 \subseteq f_1 \subseteq f_2 \subseteq \ldots$
— The size $|\mathrm{dom}(f_k)|$ of the domain of f_k is at most the number of oracle gates of nesting depth k or less.
— f_k determines the values of all oracle gates of nesting depth k or less.
— f_k is k-sequential.

We can take f_0 to be the totally undefined function, since $f^0(\mathbf{0}) = 0$ by definition, so f_0 is 0-sequential. For the induction step let M be the set of all strings Y of length n such that Y is queried by an oracle gate at level k. Let f_{k+1} be a k+1-sequential extension of f_k to domain $\mathrm{dom}(f_k) \cup M$ according to Lemma 4.7.

For $k = d$ we get the desired bound. As f_d already determines the values of all gates, the output of the circuit is already determined, but $f^{d+1}(\mathbf{0})$ is still undefined and we can define it in such a way that it differs from the first bit of the output of the circuit. □

4.1.2. Proof for 1-Lipschitz oracles. Using very similar ideas, we proceed to prove item (i) of Theorem 4.2 as stated. The idea is to change the problem slightly by using a different function for each iteration, say f_T for the Tth iteration. We "pack" these functions into one single function f by letting the first part of the input encode T, and the second part encode the actual input. We call the functions f_T *slice functions*. We are now faced with two problems: first, we need each function f_T to be 1-Lipschitz; and second, we need the composite function f to be 1-Lipschitz. We handle the two problems separately.

Maintaining the 1-Lipschitz property for the slice functions. Let X_T be the input to the Tth iteration of f. Following the proof of Theorem 4.5, at stage T of the construction we will maintain the invariant that X_T is known but $f_T(X_T)$ is undetermined. The idea is to pick $f_T(X_T)$ to be simultaneously of low Hamming weight and far away from all other points already determined for f_{T+1}. The quantitative meaning of *low* and *far away* will be chosen so that such a point always exists. Since X_T itself is simultaneously of low Hamming weight and far away from all points already determined for f_T, we can complete f_T to a 1-Lipschitz function.

Maintaining the 1-Lipschitz property for the composite function. Recall that the slice functions f_T are packed into a single composite function f, in which the first part of the input encodes T, and the second encodes the input to f_T. In order to separate the different slices from each other, we encode T by repeating each bit many times. This allows us to complete the definition of f to a 1-Lipschitz function whenever all the slice functions are 1-Lipschitz.

We proceed with the formal details.

Notation. We use T to stand for both a bit string and the number it represents in binary. The ith bit of T is $\mathrm{bit}(T,i)$; the least significant bit (lsb) is bit 1. For a bit b, $b^{(m \text{ times})}$ is the bit b repeated m times. The Hamming weight of a string X is $\|X\|$. The Hamming distance between X and Y is $d(X,Y) = \|X \oplus Y\|$. The length of X is $|X|$.

Definition 4.8. Let $n = 2^\ell$, and define $m = 2n+1$. Given $f\colon \{0,1\}^{m\ell+n} \to \{0,1\}^n$, define the slice functions

$$f_T(X) = f\left(\mathrm{bit}(T,1)^{(m \text{ times})} \dots \mathrm{bit}(T,\ell)^{(m \text{ times})} X\right), \text{ where } |T| = \ell \text{ and } |X| = n.$$

Define the iterations

$$X_0 = 0^{(n \text{ times})}, \quad X_{T+1} = f_T(X_T).$$

Finally, define

$$F = \mathrm{bit}\left(X_{\lfloor\sqrt{n}\rfloor}, 1\right).$$

LEMMA 4.9. *The function $F = F(f)$ can be computed using a uniform family of comparator circuits of size polynomial in n which use f as an oracle.*[2]

PROOF. For given n, the comparator circuits consists of $\lfloor\sqrt{n}\rfloor$ oracle gates connected in sequence. The Tth oracle gate has additional constant inputs $\mathrm{bit}(T,1)^{(m \text{ times})} \dots \mathrm{bit}(T,\ell)^{(m \text{ times})}$. The output wire is the first bit of the output of the $\lfloor\sqrt{n}\rfloor$'th oracle gate. □

If f ignores its first $m\ell$ input bits then F is the first bit of of the $\sqrt{n}$-th iteration of f, and hence by Proposition 4.5 any subexponential size circuit computing F requires depth $\sqrt{n}$. In the rest of this section we will show that even if f is assumed to be 1-Lipschitz, F cannot be computed by any circuit with only polynomially many oracle gates which are nested only polylogarithmically deep.

The first step is to reduce the problem of constructing a 1-Lipschitz f to the problem of constructing 1-Lipschitz $f_0, \dots, f_{n-1}$.

Definition 4.10. Let f be a function as in Definition 4.8. Let $R_1 \dots R_\ell X$ be an input to f, where $|R_i| = m$, $|X| = n$. Suppose R_i contains z_i zeroes and o_i ones. Define $t_i = 0$ if $z_i > o_i$ and $t_i = 1$ if $z_i < o_i$ (one of these must happen since m is odd). Let $x_i = \min(z_i, o_i)$ and $x = \max_i x_i$. The values $t_1, \dots, t_\ell$ define a string T. We say that $R_1 \dots R_\ell X$ belongs to the *blob* $B(T,X)$, and is at distance x from the center string $t_1^{(m \text{ times})} \dots t_\ell^{(m \text{ times})} X$. Thus the blobs form a partition of the domain $\{0,1\}^{m\ell+n}$ of f.

We say that f is *blob-like* if for all $R_1, \dots, R_\ell, X$, with T as defined above,

$$f(R_1 \dots R_\ell X) = f_T(X) \wedge \left(0^{(x \text{ times})} 1^{(n-x \text{ times})}\right). \tag{4.1}$$

(Here we use bitwise $\wedge$.) In words, the value of f at a point R which is at distance x from the center of some blob B is equal to the value of f at the center of the blob, with the first x bits set to zero.

We say that f is a *blob-like partial function* if it is a partial function whose domain is a union of blobs, and inside each blob it satisfies (4.1).

Note that the values at centers of blobs are unconstrained by (4.1) because then $x = 0$.

[2]We can pad the output of f with $m\ell$ zeros so that f has the same number of outputs as inputs.

LEMMA 4.11. *If f is blob-like and f_T is 1-Lipschitz for all $0 \le T < n$ then f is 1-Lipschitz.*

PROOF. Let $R_1 \dots R_\ell, X$ be an input to f. We argue that if we change a bit in the input, then at most one bit changes in the output. If we change a bit of X, then this follows from the 1-Lipschitz property of the corresponding f_T. If we change a bit of R_i without changing T, then we change x by at most 1, and so at most one bit of the output is affected. Finally, if we change a bit of R_i and this does change T, then we must have had (without loss of generality) $z_i = n, o_i = n+1$, and we changed a 1 to 0 to make $z_i = n+1, o_i = n$. In both inputs, $x = n$, and so the output is $\mathbf{0}$ in both cases. □

The second step is to find a way to construct 1-Lipschitz functions from $\{0,1\}^n$ to itself, given a small number of constraints.

LEMMA 4.12. *Suppose $g\colon \{0,1\}^n \to \{0,1\}^n$ is a partial function, and $g(P) = \mathbf{0}$ for all $P \in \mathrm{dom}(g)$. Let X be a point of Hamming distance at least d from any point in $\mathrm{dom}(g)$. Then for every Y of Hamming weight at most d, we can extend g to a 1-Lipschitz total function satisfying $g(X) = Y$.*

PROOF. Given X, Y, define $h(Z)$ to be Y with the first $\min(d(Z,X), \|Y\|)$ ones changed to zeros. We have $h(X) = Y$ since $d(X,X) = 0$. For $P \in \mathrm{dom}(g)$, $d(P,X) \ge d$ implies $h(P) = \mathbf{0}$, using $\|Y\| \le d$. Therefore h extends g. On the other hand, h is 1-Lipschitz since changing a bit of the input Z can change $d(Z,X)$ by at most 1, and so at most one bit of the output is affected. □

Finally, we need a technical lemma about the volume of Hamming balls.

Definition 4.13. Let n, d be given. Then $V(n,d)$ is the number of points in $\{0,1\}^n$ of Hamming weight at most d, that is

$$V(n,d) = \sum_{k \le d} \binom{n}{k}.$$

LEMMA 4.14. *For $d \ge 0$, $V(n,d+1)/V(n,d) \le n+1$.*

PROOF. Each point in $V(n,d+1)$ is either already a point of $V(n,d)$, or it can be obtained by taking a point of $V(n,d)$ and changing one bit from 0 to 1. Conversely, for each point of $V(n,d)$, a bit can be changed from 0 to 1 in at most n different ways. □

COROLLARY 4.15. *If $V(n,d) \ge r \ge 1$ then there exists $d' \ge 0$ such that*

$$r \le \frac{V(n,d)}{V(n,d')} < (n+1)r.$$

PROOF. Let d' be the maximum number satisfying $r \le V(n,d)/V(n,d')$. Since $r \le V(n,d) = V(n,d)/V(n,0)$, such a number exists. On the other hand,

$$\frac{V(n,d)}{V(n,d')} \le (n+1)\frac{V(n,d)}{V(n,d'+1)} < (n+1)r.$$

□

We are now ready to prove the main lower bound, which implies item (i) of Theorem 4.2.

THEOREM 4.16. *Let $a > 0$ be given. For large enough n, every circuit $C(f)$ with at most n^a oracle gates, nested less than $\sqrt{n}$ deep, fails to compute F for some 1-Lipschitz function f.*

PROOF. Put $T_{\max} = \lfloor \sqrt{n} \rfloor - 1$. Let $d_0, \dots, d_{T_{\max}}$ be a sequence of positive integers satisfying

$$\frac{V(n,d_T)}{V(n,d_{T+1}-1)} > n^a, \quad 0 \le T < T_{\max}. \tag{4.2}$$

Such numbers exist whenever $2^n \geq (n+1)^{T_{\max}(a+1)}$, which holds when n is large enough. Indeed, we will construct such a sequence inductively using Corollary 4.15, keeping the invariant

$$V(n,d_T) \geq \frac{2^n}{(n+1)^{T(a+1)}}.$$

For the base case, $d_0 = n$ certainly satisfies the invariant. Given d_T, use Corollary 4.15 with $d = d_T$ and $r = (n+1)^a$. Since $V(n,d_T) \geq 2^n/(n+1)^{T(a+1)} \geq (n+1)^{a+1}$ for large n, the corollary supplies us with d' satisfying

$$(n+1)^a \leq \frac{V(n,d_T)}{V(n,d')} < (n+1)^{a+1}.$$

Let $d_{T+1} = d' + 1$. This certainly satisfies condition (4.2), and the invariant is satisfied since

$$V(n,d_{T+1}) > V(n,d') > \frac{V(n,d_T)}{(n+1)^{a+1}} \geq \frac{2^n}{(n+1)^{(T+1)(a+1)}}.$$

We will define the function f in $T_{\max}$ stages, similar to the proof of Proposition 4.5, except we use the notation $f^{(k)}$ instead of f_k. At every stage the function $f^{(k)}$ will be a blob-like partial function which defines the output of every oracle gate in $C(f)$ of nesting depth k or less. The starting point is $f^{(0)}$, which is the empty function. At stage k we will define the partial function $f^{(k+1)}$, which extends $f^{(k)}$, keeping the following invariants:

— $f^{(0)} \subseteq f^{(1)} \subseteq f^{(2)} \subseteq \dots$
— $f^{(k)}$ is a blob-like partial function.
— For $T < k$, $f_T^{(k)}$ is a total 1-Lipschitz function.
— For $T \geq k$, at any point P at which $f_T^{(k)}$ is defined it is equal to $\mathbf{0}$ and some gate in $C(f)$ of nesting depth k or less has its input in the blob $B(T,P)$. Hence $|\operatorname{dom}(f_T^{(k)})| \leq n^a$.
— X_k is defined by $f^{(k)}$.
— $f_k^{(k)}(X_k)$ is undefined.
— $d(P,X_k) \geq d_k$ for any $P \in \operatorname{dom}(f_k^{(k)})$. $\quad (*)$
— $f^{(k)}$ determines that the output of every oracle gate of nesting depth k or less.

It is easy to verify that the empty function $f^{(0)}$ satisfies the invariants. The function $f^{(T_{\max})}$ determines the output of the circuit $C(f)$. However, $X_{\lfloor\sqrt{n}\rfloor} = f_T^{(T_{\max})}(X_{T_{\max}})$ is undefined. We can extend $f^{(T_{\max})}$ to a 1-Lipschitz function in two different ways: Put $f_T = \mathbf{0}$ for $T > T_{\max}$, and let $f_{T_{\max}}$ be either (1) the constant zero function, or (2) the function which is zero everywhere except for $f_{T_{\max}}(X_{T_{\max}}) = 0^{(n-1 \text{ times})}1$. Since F is different in these two extensions, the circuit fails to compute F correctly in one of them.

It remains to show how to define $f^{(k+1)}$ given $f^{(k)}$. Let $\mathbb{G}$ be the set of oracle gates of nesting depth exactly $k+1$. For any $G \in \mathbb{G}$ whose input belongs to a blob $B(T,X)$ for $T > k$, if $f_T^{(k)}(X)$ is undefined, then define $f^{(k+1)}$ so that it extends $f^{(k)}$ and is $\mathbf{0}$ on the entire blob $B(T,X)$ (this is a blob-like assignment). Let $A = \operatorname{dom}(f_{k+1}^{(k+1)})$; note that $|A| \leq n^a$. Condition (4.2) implies

$$V(n,d_{k+1}-1)|A| < V(n,d_k),$$

and so there is a point Y of Hamming weight at most d_k which is of distance at least d_{k+1} from each point in A. Define $f_k^{(k+1)}(X_k) = Y$ (so $X_{k+1} = Y$), and extend $f_k^{(k+1)}$ to a total 1-Lipschitz function using Lemma 4.12 with $d = d_k$ (use invariant $(*)$). Then extend $f^{(k+1)}$ to a blob-like partial function using (4.1). It is routine to verify that the invariants hold for $f^{(k+1)}$. $\square$

Remark 4.17.

— A more natural target function is $F' = \mathrm{bit}(X_n, 1)$. We can easily modify the proof of Theorem 4.16 to handle this function. We set the first $n - \lfloor\sqrt{n}\rfloor$ functions $f^{(0)}, \dots, f^{(n-\lfloor\sqrt{n}\rfloor-1)}$ to be constant, and then run the proof from that point on.
— An even more natural target function has an unstructured f as input, and $F'' = \mathrm{bit}\big(f^{(N)}(\mathbf{0}), 1\big)$. We leave open the question of whether the method can be adapted to work in this case.
— In the paragraph preceding Section 4.1, we have shown how to construct F''' that separates CC and NC even under the restriction that the oracle be strictly 1-Lipschitz. The function F''' basically ignores one of the outputs of the oracle while iterating it. It is possible to slightly modify the proof of Theorem 4.16 so that it directly applies to F even under the restriction that the oracle is strictly 1-Lipschitz.

4.2. Proof that NC(α) is not contained in CC(α)

Here we exploit the 1-Lipschitz property of comparator gates and $\neg$-gates by using oracles which are weakly 1-Lipschitz, so that all gates in the relativized circuits have this property. The idea is to use an oracle α with dn^2 inputs ($d \geq 3$) but only n useful outputs. We can feed the n useful outputs back into another instance of α by using dn copies of each output bit. Because of the 1-Lipschitz property it seems as though a comparator circuit computing the mth iteration of α in this way needs either at least $2^{\Omega(m)}$ copies of α, or alternatively 2^n copies of α and a complicated circuit analyzing the output. When $m = \Omega(\log^2 n)$, this construction requires a super-polynomial size comparator circuit computing the mth iteration of α. On the other hand, for $m = O(\log^2 n)$, the mth iteration can be easily computed in relativized NC^3 (following Aehlig et al. [2007], we require oracle gates to be nested at most $O(\log^{k-1} n)$ deep in relativized NC^k).

To make this argument work we initially assume that instead of computing the m-th iteration of α we compose m different oracles $A_1, \dots, A_m$ in the way just described. The crucial property of comparator circuits we use is the *flip-path* property:

> If one input wire is changed, then (given that each gate has the 1-Lipschitz property) there is a unique path through the circuit tracing the effect of the original flip.

We use a Gray code to order the possible n-bit outputs of the oracle and study the effects of the 2^n flip-paths generated as the definition of the oracle is successively changed. Recall that a Gray code of length n is a list of all n-bit strings ordered so that every two adjacent strings, including the last and the first, differ by exactly one bit. One such code is given by letting the $(x_{n-1} \dots x_0)_2$'th string (counting from zero in base 2) be $x_{n-1}(x_{n-1} \oplus x_{n-2})(x_{n-2} \oplus x_{n-3}) \cdots (x_1 \oplus x_0)$; for example, for $n = 3$ the code is

$$000, 001, 011, 010, 110, 111, 101, 100.$$

In detail, Let $n, m, d \in \mathbb{N}$, with $d \geq 3$. For each $k \in [m]$ and $i \in [n]$, let $a_i^k \colon \{0,1\}^{dn} \to \{0,1\}$ be a Boolean oracle with dn input bits. Let $A^k = (a_1^k, \dots, a_n^k)$. We define a function $y = f[A^1, \dots, A^m]$ as follows:

$$x_i^k = a_i^k(\overbrace{x_1^{k+1}, \dots, x_1^{k+1}}^{d \text{ times}}, \dots, \overbrace{x_n^{k+1}, \dots, x_n^{k+1}}^{d \text{ times}}), \qquad k \in [m],\ i \in [n], \tag{4.3}$$

$$x_i^{m+1} = 0, \qquad i \in [n], \tag{4.4}$$

$$y = x_1^1 \oplus \cdots \oplus x_n^1. \tag{4.5}$$

As stated the oracle a_i^k has dn inputs and just one output, but we can make it fit our convention that an oracle gate has the same number of outputs as inputs simply by assuming that the gate has an additional $dn - 1$ outputs which are identically zero.

Note that the function computed by such an oracle is necessarily (weakly) 1-Lipschitz.

Let $X^k = (x_1^k, \ldots, x_n^k)$ and $A^k = (a_1^k, \ldots, a_n^k)$. Note that an oracle circuit of depth $m + O(\log n)$ with mn gates can compute y simply by successively computing $X_m, X_{m-1}, \ldots, X_1$ and computing the parity of X_1, provided that the circuit is allowed to have gates with fan-out d. However, the fan-out restriction for comparator circuits allows us to prove the following.

THEOREM 4.18. *If $n \geq 3$, then every oracle comparator circuit computing $y = f[A^1, \ldots, A^m]$ has at least*

$$\min\left(2^n, (d-2)^{m-1}\right)$$

gates.

Before proving the theorem, we comment that by setting $m = \log^2 n$ and $d = 4$ this almost proves item (ii) of Theorem 4.2, except we need to argue that the array of oracles a_i^k can be replaced by a single oracle. Later we will show how a simple adaptation of the proof of Theorem 4.18 accomplishes this.

PROOF OF THEOREM 4.18. Fix an oracle comparator circuit C which computes $y = f[A^1, \ldots, A^m]$.

Definition 4.19. We say that an input $(z_1, \ldots, z_{dn})$ to some oracle a_i^k in C is *regular* if it has the form of the inputs in (4.3); that is if $z_{(a-1)d+b} = z_{(a-1)d+c}$ for all $a \in [n]$ and $b, c \in [d]$. We say that an oracle a_i^k is *regular* if $a_i^k(Z) = 0$ for all irregular inputs Z.

Note that any irregular oracle a_i^k can be replaced by an equivalent regular oracle which does not affect (4.3).

Definition 4.20. Let g be the total number of any of the gates a_i^k in the circuit C. For a given assignment to the oracles, a particular gate a_i^k is *active* in C if its input is as specified by (4.3,4.4).

Let g_k be the expected total number of active gates $a_1^k, \ldots, a_n^k$ in C under a uniformly random *regular* setting of *all* oracles.

It is easy to see that

$$g_1 \geq n, \tag{4.6}$$

since we need at least one active gate a_i^1 for each $i \in [n]$.

Let $k \in [m]$ be greater than 1. We will show that

$$g_{k-1} \leq \frac{g}{2^n} + \frac{g_k}{d-2}. \tag{4.7}$$

We use the following consequence of the (weakly) 1-Lipschitz property of all gates in the circuit: If we change the definition of some copy of some gate a_i^k at its input in the circuit C, this generates a unique flip-path which may end at some copy of some other gate, in which case we say that the latter gate *consumes* the flip-path. (The flip-path is a path in the circuit such that the Boolean value of each edge in the path is negated.)

Let $G_1, \ldots, G_{2^n}$ be a Gray code listing all strings in $\{0,1\}^n$, where $G_1 = 0^n$. Thus the Hamming distance between any two successive strings G_i and G_{i+1}, and between G_{2^n} and G_1, is one. Take a random regular setting of all the oracles, and let Z_1 be the value of X_k under this setting. Shift the above Gray code to form a new one $Z_1, \ldots, Z_{2^n}$ by setting $Z_t = G_t \oplus Z_1$. Then for each $t \in [2^n]$, Z_t is uniformly distributed and independent of X_ℓ for $\ell \neq k$. Thus if we change the output of A^k at its active input to Z_t, the result is again a uniformly random regular oracle setting. Let γ_t be the number of active A^k gates (i.e. any active gate of the form a_i^k for some i) after this change, and let δ_t be the number of active A^{k-1} gates after the change. Taking expectations we have for each $t \in [2^n]$

$$\mathbb{E}(\gamma_t) = g_k, \qquad \mathbb{E}(\delta_t) = g_{k-1}. \tag{4.8}$$

We will change the output of A^k (at its active input) successively from Z_1 to Z_{2^n}, and consider the relationship between γ_t and δ_t. The total number of flip paths generated during the process is

$$\sum_{t=1}^{2^n-1} \gamma_t.$$

Each time an A^{k-1} gate is rendered active for the first time, we will call the gate *fresh*. Otherwise, it is *reused*. At time t, let δ'_t (δ''_t) be the number of fresh (reused) A^{k-1} gates. Thus $\delta''_1 = 0$, and

$$\delta_t = \delta'_t + \delta''_t \tag{4.9}$$

Since a given gate can be fresh at most once, we have

$$\sum_{t=1}^{2^n} \delta'_t \le g \tag{4.10}$$

Each time an a_i^{k-1} gate is reused it has consumed at least $d-2$ flip-paths since the last time it was active. This is because at least d consecutive inputs must be changed from all 0's to all 1's (or vice versa), and since the gate is regular, its output will be constantly 0 during at least $d-2$ consecutive changes.

Since there must be at least as many flip-paths generated as consumed, we have

$$(d-2)\sum_{t=1}^{2^n} \delta''_t \le \sum_{t=1}^{2^n} \gamma_t. \tag{4.11}$$

From (4.9), (4.10), (4.11) we have

$$\sum_{t=1}^{2^n} \delta_t \le g + \frac{1}{d-2}\sum_{t=1}^{2^n} \gamma_t. \tag{4.12}$$

Now (4.7) follows from (4.12) and (4.8) by linearity of expectations.

Hence either $g > 2^n$ or

$$g_k \ge (d-2)[g_{k-1} - 1].$$

From this and (4.6) we have a recurrence whose solution shows

$$g_t \ge (d-2)^{t-1} n - \frac{(d-2)^t - (d-2)}{d-3}.$$

If $n \ge 3$ then $g_m \ge (d-2)^{m-1}$, and Theorem 4.18 follows. □

Now we change the setting in Theorem 4.18 so that it applies to a single oracle. The new oracle $a(k, i, x)$ is used in the same way as $a_i^k(x)$. The first two arguments can be encoded in binary or unary, and we don't care what happens when they are not "legal" (we don't require the output to be 0 unless the x argument is illegal). Define active a_i^k gates as gates whose inputs are (k, i, x), where x is the relevant active input. We argue as before, and again conclude that if $n \ge 3$ then $g_m \ge (d-2)^{m-1}$. Hence Theorem 4.18 follows in the single oracle setting, and Theorem 4.2 (ii) follows as explained right after the statement of Theorem 4.18.

4.3. SC vs CC

Uniform SC^k is the class of problems decidable by Turing machines running in polynomial time using $O(\log^k n)$ space. Non-uniform SC^k is the class of problems solvable by circuits of polynomial size and $O(\log^k n)$ width. Just as NC and CC appear to be incomparable, it seems plausible that SC and CC are incomparable. For one direction, NL is a subclass of both CC and NC, but is conjectured not to be a subclass of SC (Savitch's algorithm takes $2^{O(\log^2)}$ time). For the other direction we can give a convincing oracle separation as follows.

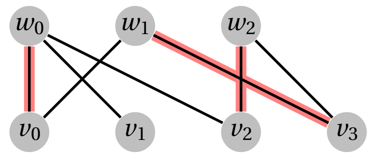


Fig. 8: The thick edges form the lfm-matching of the above bipartite graph.

We apply Theorem 4.18 to a problem with a padded input of length N, and set the 'real' input $n = \log^2 N$, and also $m = \log^2 N$ and $d = 4$. The theorem implies that every comparator circuit solving F has size at least

$$2^{\min(m-1,n)} = 2^{\log^2 N},$$

which is superpolynomial. Thus this padded problem is not in relativized CC.

However, a Turing machine M equipped with an oracle tape which can query $4\log^2 N$ bits of each oracle a_i^k can compute this padded version of F in linear time and $O(n) = O(\log^2 N)$ space, so this problem is in relativized SC^2. The machine M proceeds by successively computing $X_m, X_{m-1}, \ldots, X_1$, writing each of these X_i on its work tape and then erasing the previous one. The machine computes X^k from X^{k+1} bit by bit, making a query of size $4\log^2 N$ to its query tape for each bit. (We assume that M can access its oracle in such a way that it can determine N, and hence m and n.)

5. LEXICOGRAPHICALLY FIRST MAXIMAL MATCHING PROBLEMS ARE CC-COMPLETE

Let $G = (V, W, E)$ be a bipartite graph, where $V = \{v_i\}_{i=0}^{m-1}$, $W = \{w_i\}_{i=0}^{n-1}$ and $E \subseteq V \times W$. We require that $|V| = |W|$ for convenience in proving that the stable marriage problem is CC-complete in Section 6, although the reductions in this section work even without this assumption. The *lexicographically first maximal matching* (lfm-matching) is the matching produced by successively matching each vertex $v_0, \ldots, v_{m-1}$ to the least vertex available in W (see Fig. 8 for an example). We refer to V as the set of bottom nodes and W as the set of top nodes.

In this section we will show that two decision problems concerning the lfm-matching of a bipartite graph are CC-complete under AC^0 many-one reductions. The lfm-matching problem (LFMM) is to decide if a designated edge belongs to the lfm-matching of a bipartite graph G. The vertex version of lfm-matching problem (VLFMM) is to decide if a designated top node is matched in the lfm-matching of a bipartite graph G. LFMM is the usual way to define a decision problem for lfm-matching as seen in [Mayr and Subramanian 1992; Subramanian 1994]; however, as shown in Sections 5.1 and 5.2, the VLFMM problem is even more closely related to the CCV problem.

We will show that the following two more restricted lfm-matching problems are also CC-complete. We define 3LFMM (3VLFMM) to be the restriction of LFMM (VLFMM) to bipartite graphs of degree at most three.

To show that the problems defined above are equivalent under AC^0 many-one reductions, it turns out that we also need the following intermediate problem. A negation gate flips the value on a wire. For comparator circuits with negation gates, we allow negation gates to appear on any wire (see the left diagram of Fig. 13 below for an example). The comparator circuit value problem with negation gates (CCV¬) is: given a comparator circuit with negation gates and input assignment, and a designated wire, decide if that wire outputs 1.

All reductions in this section are summarized in Fig. 9.

5.1. CCV $\leq_m^{\mathsf{AC}^0}$ 3VLFMM

By Proposition 2.2 it suffices to consider only instances of CCV in which all comparator gates point upward. We will show that these instances of CCV are AC^0 many-one reducible to instances of 3VLFMM, which consist of bipartite graphs with *degree at most three*.

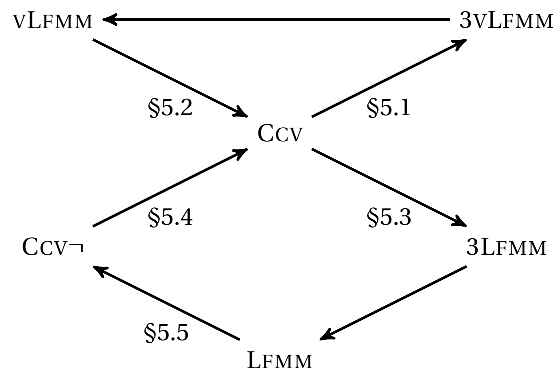
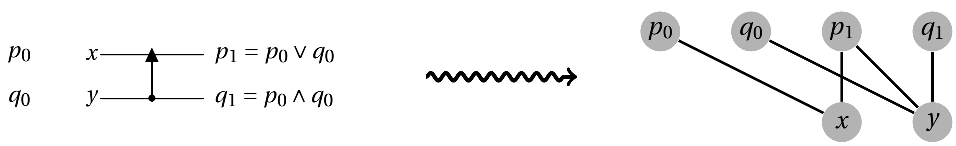


Fig. 9: The label of an arrow denotes the section in which the reduction is described. Arrows without labels denote trivial reductions. All six problems are CC-complete.

The key observation is that a comparator gate on the left below closely relates to an instance of 3vLFMM on the right. We use the top nodes p_0 and q_0 to represent the values p_0 and q_0 carried by the wires x and y respectively before the comparator gate, and the nodes p_1 and q_1 to represent the values of x and y after the comparator gate, where a top node is matched iff its respective value is one.



If nodes p_0 and q_0 have not been previously matched, i.e. $p_0 = q_0 = 0$ in the comparator circuit, then the edges $\langle x, p_0\rangle$ and $\langle y, q_0\rangle$ are added to the lfm-matching. So the nodes p_1 and q_1 are not matched. If p_0 has been previously matched, but q_0 has not, then edges $\langle x, p_1\rangle$ and $\langle y, q_0\rangle$ are added to the lfm-matching. So the node p_1 will be matched but q_1 will remain unmatched. The other two cases are similar.

Thus, we can reduce a comparator circuit to the bipartite graph of a 3vLFMM instance by converting each comparator gate into the "gadget" described above. We will describe our method through an example, where we are given the comparator circuit in Fig. 10.

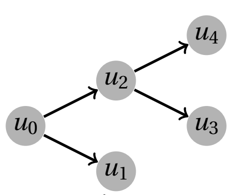


Fig. 6

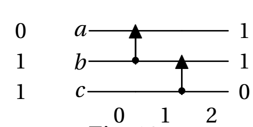
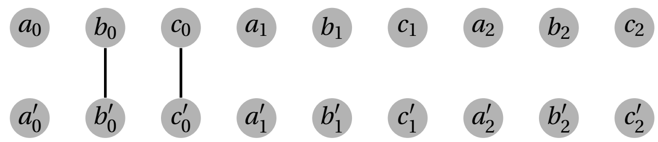


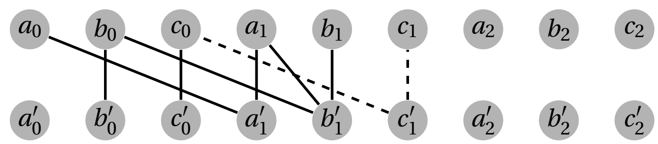
Fig. 10

We divide the comparator circuit into vertical layers 0, 1, 2 as shown in Fig. 10. Since the circuit has three wires a, b, c, for each layer i, we use six nodes, including three top nodes a_i, b_i and c_i representing the values of the wires a, b, c respectively, and three bottom nodes a'_i, b'_i, c'_i, which are auxiliary nodes used to simulate the effect of the comparator gate at layer i.

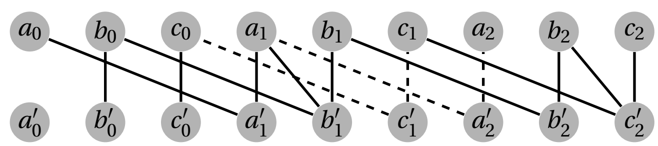
Layer 0: This is the input layer, so we add an edge $\{x_i, x'_i\}$ iff the wire x takes input value 1. In this example, since b and c are wires taking input 1, we need to add the edges $\{b_0, b_0'\}$ and $\{c_0, c_0'\}$.



Layer 1: We then add the gadget simulating the comparator gate from wire b to wire a as follows. (The dashed edges are regular edges which are dashed only for emphasizing their different function in the construction.)



Since the value of wire c does not change when going from layer 0 to layer 1, we can simply propagate the value of c_0 to c_1 using the pair of dashed edges in the picture.
Layer 2: We proceed very similarly to layer 1 to get the following bipartite graph.



Finally, we can get the output values of the comparator circuit by looking at the "output" nodes a_2, b_2, c_2 of this bipartite graph. We can easily check that a_2 is the only node that remains unmatched, which corresponds exactly to the only zero produced by wire a of the comparator circuit in Fig. 10.

It remains to argue that the construction above is an AC^0 many-one reduction. We observe that each gate in the comparator circuit can be independently reduced to exactly one gadget in the bipartite graph that simulates the effect of the comparator gate; furthermore, the position of each gadget can be easily calculated from the position of each gate in the comparator circuit using very simple arithmetic.

5.2. vLFMM $\leq_m^{\mathsf{AC}^0}$ Ccv

Consider the instance of vLFMM consisting of the bipartite graph in Fig. 11. Recall that we find the lfm-matching by matching the bottom nodes x, y, z successively to the first available node on the top. Hence we can simulate the matching of the bottom nodes to the top nodes using the comparator circuit on the right of Fig. 11, where we can think of the moving of a 1, say from wire x to wire a, as the matching of node x to node a in the original bipartite graph. In this construction, a top node is matched iff its corresponding wire in the circuit outputs 1.

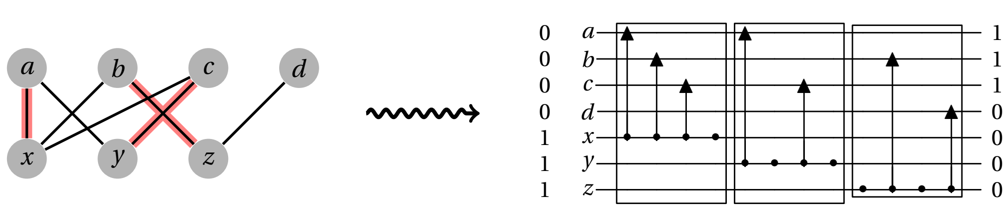


Fig. 11

Note that we draw bullets without any arrows going out from them in the circuit to denote dummy gates, which do nothing. These dummy gates are introduced for the following technical reason. Since the bottom nodes might not have the same degree, the position of a comparator gate really depends on the structure of the graph, which makes it harder to give a direct AC^0 reduction. By using dummy gates, we can treat the graph as if it is a complete bipartite graph, the missing edges represented by dummy gates. This can easily be shown to be an AC^0 reduction from VLFMM to CCV. Together with the reduction from Section 5.1, we get the following theorem.

THEOREM 5.1. *The problems* CCV, 3VLFMM *and* VLFMM *are equivalent under* AC^0 *many-one reductions.*

5.3. CCV $\leq_m^{\mathsf{AC}^0}$ 3LFMM

We start by applying the reduction CCV $\leq_m^{\mathsf{AC}^0}$ 3VLFMM of Section 5.1 to get an instance of 3VLFMM, and notice that the degrees of the top "output" nodes of the resulting bipartite graph, e.g. the nodes a_2, b_2, c_2 in the example of Section 5.1, have degree at most two. Now we show how to reduce such instances of 3VLFMM (i.e. those whose designated top vertices have degree at most two) to 3LFMM. Consider the graph G with degree at most three and a designated top vertex b of degree two as shown on the left of Fig. 12. We extend it to a bipartite graph G' by adding an additional top node w_t and an additional bottom node w_b, alongside two edges $\{b, w_b\}$ and $\{w_t, w_b\}$, as shown in Fig. 12. Observe that the degree of the new graph G' is at most three.

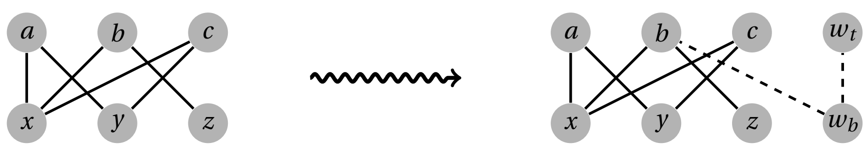


Fig. 12

We treat the resulting bipartite graph G' and the edge $\{w_t, w_b\}$ as an instance of 3LFMM. It is not hard to see that the vertex b is matched in the lfm-matching of the original bipartite graph G iff the edge $\{w_t, w_b\}$ is in the lfm-matching of the new bipartite graph G'.

5.4. CCV¬ $\leq_m^{\mathsf{AC}^0}$ CCV

Recall that a comparator circuit value problem with negation gates (CCV¬) is the task of deciding, given a comparator circuit with negation gates and an input assignment, whether a designated wire outputs one. It should be clear that CCV is a special case of CCV¬ and hence AC^0 many-one reducible to CCV¬. Here, we show the nontrivial direction that CCV¬ $\leq_m^{\mathsf{AC}^0}$ CCV. Our proof is based on Subramanian's idea from [Subramanian 1994].

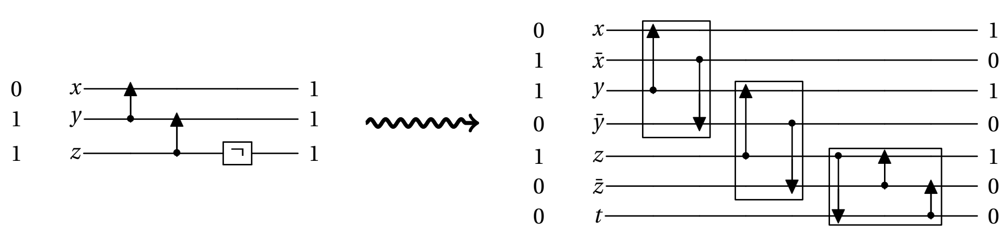


Fig. 13: Successive gates on the left circuit correspond to successive boxes of gates on the right circuit.

The reduction is based on "double-rail" logic, which can be traced to Goldschlager's proof of the P-completeness of the monotone circuit value problem [Goldschlager 1977]. Given an instance of CCV¬ consisting of a comparator circuit with negation gates C with its input I and a designated wire s, we construct an instance of CCV consisting of a comparator circuit C' with its input I' and a designated wire s' as follows. For every wire w in I we put in two corresponding wires, w and $\overline{w}$, in C'. We define the input I' of C' such that the input value of $\overline{w}$ is the negation of the input value of w. We want to fix things so that the value carried by the wire $\overline{w}$ at each layer is always the negation of the value carried by w. For any comparator gate $\langle y,x\rangle$ in C we put in C' the gate $\langle y,x\rangle$ followed by the gate $\langle \overline{x},\overline{y}\rangle$. It is easy to check using De Morgan's laws that the wires x and y in C' carry the corresponding values of x and y in C, and the wires $\overline{x}$ and $\overline{y}$ in C' carry the negations of the wires x and y in C.

The circuit C' has one extra wire t with input value 0 to help in translating negation gates. For each negation gate on a wire, says z in the example from Fig. 13, we add three comparator gates $\langle z,t\rangle$, $\langle \overline{z},z\rangle$, $\langle t,\overline{z}\rangle$ as shown in the right circuit of Fig. 13. Thus t as a temporary "container" that we use to swap the values carried by the wires z and $\overline{z}$. Note that the swapping of values of z and $\overline{z}$ in C' simulates the effect of a negation in C. Also note that after the swap takes place, the value of t is restored to 0. (The more straightforward solution of simply switching the wires z and $\overline{z}$ does not result in an AC^0 many-one reduction.)

Finally note that the output value of the designated wire s in C is 1 iff the output value of the corresponding wire s in C' with input I' is 1. Thus we set the designated wire s' in I' to be s.

5.5. LFMM $\leq_m^{\mathsf{AC}^0}$ CCV¬

Consider an instance of LFMM consisting of the bipartite graph on the left of Fig. 14, and a designated edge $\{y,c\}$. Without loss of generality, we can safely ignore all top vertices occurring after c, all bottom vertices occurring after y, and all the edges associated with them, since they are not going to affect the outcome of the instance. Using the construction from Section 5.2, we can simulate the matching of the bottom nodes to the top nodes using the comparator circuit in the upper box on the right of Fig. 14.

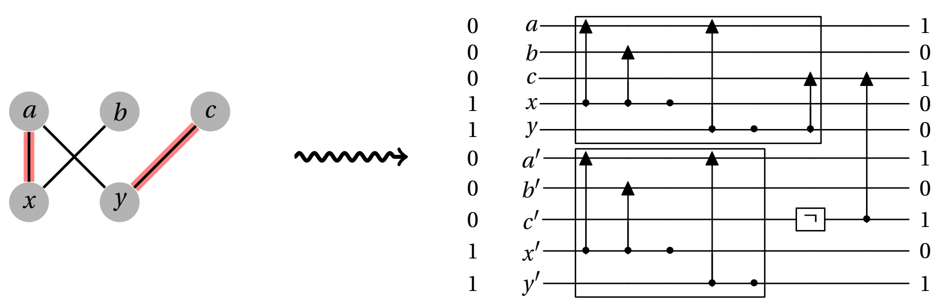


Fig. 14

We keep another running copy of this simulation on the bottom (see the wires labelled a',b',c',x',y' in Fig. 14). The only difference is that the comparator gate $\langle y',c'\rangle$ corresponding to the designated edge $\{y,c\}$ is not added. Finally, we add a negation gate on c' and a comparator gate $\langle c',c\rangle$. We let the desired output of the CCV instance be the output of c, since c outputs 1 iff the edge $\{y,c\}$ is added to the lfm-matching. It is not hard to generalize this construction to an arbitrary bipartite graph and designated edge.

Combined with the constructions from Sections 5.1 and 5.2, we have the following corollary.

COROLLARY 5.2. *The problems* CCV, 3vLFMM, vLFMM, CCV¬, 3LFMM *and* LFMM *are equivalent under* AC^0 *many-one reductions.*

6. THE SM PROBLEM IS CC-COMPLETE

An instance of the stable marriage problem (SM), proposed by Gale and Shapley [1962] in the context of college admissions, is given by a number n (specifying the number of men and the number of women), together with a preference list for each man and each woman specifying a total ordering on all people of the opposite sex. The goal of SM is to produce a perfect matching between men and women, i.e., a bijection from the set of men to the set of women, such that the following *stability* condition is satisfied: there are no two people of opposite sex who like each other more than their current partners. Such a stable solution always exists, but it may not be unique. Thus, SM is a search problem rather than a decision problem.

However there is always a unique *man-optimal* and a unique *woman-optimal* solution. In the man-optimal solution each man is matched with a woman whom he likes at least as well as any woman that he is matched with in any stable solution. Dually for the woman-optimal solution. Thus we define the *man-optimal stable marriage decision problem* (MOSM) as follows: given an instance of the stable marriage problem together with a designated man-woman pair, determine whether that pair is married in the man-optimal stable marriage. We define the *woman-optimal stable marriage decision problem* (WOSM) analogously.

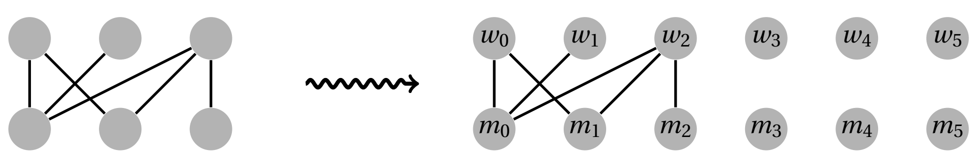
We show here that the search version and the decision versions are computationally equivalent, and each is complete for CC. Section 6.1 shows how to reduce the lexicographically first maximal matching problem (which is complete for CC) to the SM search problem, and Section 6.3 shows how to reduce both the MOSM and WOSM problems to CCV, using Subramanian's algorithm [Subramanian 1990; Subramanian 1994].

6.1. 3LFMM is AC^0 many-one reducible to SM, MOSM and WOSM

We start by showing that 3LFMM is AC^0 many-one reducible to SM when we regard both 3LFMM and SM as search problems. (Of course the lfm-matching is the unique solution to 3LFMM formulated as a search problem, but it is still a total search problem.) Our treatment follows [Cook et al. 2011].

THEOREM 6.1. *3LFMM is AC^0 many-one reducible to SM, MOSM and WOSM.*

PROOF. Let $G = (V, W, E)$ be a bipartite graph from an instance of 3LFMM, where V is the set of bottom nodes, W is the set of top nodes, and E is the edge relation such that the degree of each node is at most three (see the example in the figure on the left below). Without loss of generality, we can assume that $|V| = |W| = n$. To reduce it to an instance of SM, we double the number of nodes in each partition, where the new nodes are enumerated after the original nodes and the original nodes are enumerated using the ordering of the original bipartite graph, as shown in the diagram on the right below. We also let the bottom nodes and top nodes represent the men and women, respectively.



It remains to define a preference list for each person in this SM instance. The preference list of each man m_i, who represents a bottom node in the original graph, starts with all the women w_j (at most three of them) adjacent to m_i in the order that these women are enumerated, followed by all the women $w_n, \dots, w_{2n-1}$; the list ends with all women w_j not adjacent to m_i also in the order that they are enumerated. For example, the preference list of m_2 in our example is $w_2, w_3, w_4, w_5, w_0, w_1$. The preference list of each newly introduced man m_{n+i} simply consists of $w_0, \dots, w_{n-1}, w_n, \dots, w_{2n-1}$, i.e., in the order that the top nodes are listed. Preference lists for the women are defined dually.

We defined the preference lists so that there is a unique stable marriage corresponding to the lfm-matching of G in the following way: for each $i < n$, if m_i is matched with w_j then m_i must marry w_j in any stable marriage, and if m_i is unmatched then he must marry the first available woman in $w_n, \ldots, w_{2n-1}$; a dummy man m_i for $i \geq n$ must marry the first available woman in $w_0, \ldots, w_{2n-1}$. We proceed to prove this fact by induction on i.

Suppose first that $i < n$ and that m_i is matched with w_j in the lfm-matching. The induction hypothesis shows that all neighbors of m_i preceding w_j in G must be married to some men preceding m_i in G, and so m_i cannot be married to a woman preceding w_j in his preference order. Similarly, the induction hypothesis shows that all neighbors of w_j preceding m_i in G must be married to some women preceding w_j in G, and so w_j cannot be married to a man preceding m_i in her preference order. Therefore the marriage can only be stable if m_i is married to w_j.

Suppose next that $i < n$ and that m_i is unmatched in the lfm-matching. The induction hypothesis shows that all neighbors of m_i must be married to some men preceding m_i in G, and so m_i must be married either to some dummy woman w_t (where $t \geq n$) or to some real woman w_t (where $t < n$) who doesn't neighbor him in G. The induction hypothesis prescribes that some of the dummy women be married to some real men. Let w_t (where $t \geq n$) be the first dummy woman which is not married to any man preceding m_i in G; there must be such a woman since at most $i - 1 < n$ dummy women can be married to men preceding m_i in G. By definition, w_t cannot be married to any man preceding m_i in G, and so to any man preceding m_i in her preference order. Similarly, m_i cannot be married to any woman preceding w_t in his preference order. Therefore the marriage can only be stable if m_i is married to w_t.

Finally, suppose that $i \geq n$. The induction hypothesis postulates certain marriages. Let w_t be the first among $w_0, \ldots, w_{2n-1}$ which is not prescribed in this way. Since the induction hypothesis prescribes who $m_1, \ldots, m_{i-1}$ are married to, we see that w_t cannot be married to any man preceding m_i in her preference order (we need to consider separately the two cases $t < n$ and $t \geq n$). A similar fact is true for m_i by the definition of w_t, and so in any stable marriage, m_i is married to w_t.

We conclude that the construction gives us a many-one reduction from 3LFMM to SM as search problems. Moreover, the reduction can be done in AC^0 since the degree of each node is at most three (without this restriction, the reduction requires counting beyond the power of AC^0). Since the stable marriage is unique, the construction also shows that the decision version of 3LFMM is AC^0 many-one reducible to either of the decision problems MOSM and WOSM. □

6.2. THREE-VALUED CCV is CC-complete

In the remainder of the section, we will be occupied with developing an algorithm due to Subramanian [1990; 1994] that finds a stable marriage using comparator circuits, thus furnishing an AC^0 reduction from SM to CCV. To this end, it turns out to be conceptually simpler to go through a new variant of CCV, where the wires are three-valued instead of Boolean. This variant already appears in Subramanian [1990], and our treatment follows [Cook et al. 2011].

We define the THREE-VALUED CCV problem similarly to CCV, i.e., we want to decide, on a given input assignment, if a designated wire of a comparator circuit outputs one. The only difference is that each wire can now take either value 0, 1 or $*$, where a wire takes value $*$ when its value is not known to be 0 or 1. The output values of the comparator gate on two input values p and q will be defined as follows.

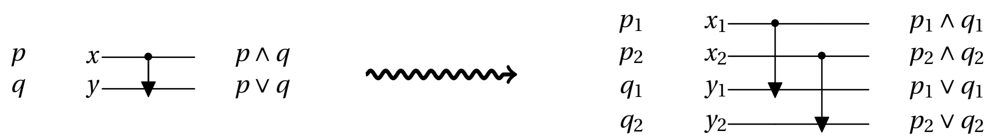
$$p \wedge q = \begin{cases} 0 & \text{if } p = 0 \text{ or } q = 0 \\ 1 & \text{if } p = q = 1 \\ * & \text{otherwise.} \end{cases} \qquad p \vee q = \begin{cases} 0 & \text{if } p = q = 0 \\ 1 & \text{if } p = 1 \text{ or } q = 1 \\ * & \text{otherwise.} \end{cases}$$

Clearly every instance of CCV is also an instance of THREE-VALUED CCV. We will show that every instance of THREE-VALUED CCV is AC^0 many-one reducible to an instance of CCV by using a pair of Boolean wires to represent each three-valued wire and adding comparator gates appropriately to simulate three-valued comparator gates.

THEOREM 6.2 (SUBRAMANIAN [1990]). THREE-VALUED CCV *and* CCV *are equivalent under* AC^0 *many-one reductions.*

PROOF. Since each instance of CCV is a special case of THREE-VALUED CCV, it only remains to show that every instance of THREE-VALUED CCV is AC^0 many-one reducible to an instance of CCV.

First, we will describe a gadget built from standard comparator gates that simulates a three-valued comparator gate as follows. Each wire of an instance of THREE-VALUED CCV will be represented by a pair of wires in an instance of CCV. Each three-valued comparator gate on the left below, where $p, q, p \wedge q, p \vee q \in \{0, 1, *\}$, can be simulated by a gadget consisting of two standard comparator gates on the right below.



The wires x and y are represented using the two pairs of wires $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$, and three possible values 0, 1 and $*$ will be encoded by $\langle 0,0\rangle$, $\langle 1,1\rangle$, and $\langle 0,1\rangle$ respectively. The fact that our gadget correctly simulates the three-valued comparator gate is shown in the following table.

p	q	$\langle p_1, p_2\rangle$	$\langle q_1, q_2\rangle$	$p \wedge q$	$p \vee q$	$\langle p_1 \wedge q_1, p_2 \wedge q_2\rangle$	$\langle p_1 \vee q_1, p_2 \vee q_2\rangle$
0	0	$\langle 0,0\rangle$	$\langle 0,0\rangle$	0	0	$\langle 0,0\rangle$	$\langle 0,0\rangle$
0	1	$\langle 0,0\rangle$	$\langle 1,1\rangle$	0	1	$\langle 0,0\rangle$	$\langle 1,1\rangle$
0	$*$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	0	$*$	$\langle 0,0\rangle$	$\langle 0,1\rangle$
1	0	$\langle 1,1\rangle$	$\langle 0,0\rangle$	0	1	$\langle 0,0\rangle$	$\langle 1,1\rangle$
1	1	$\langle 1,1\rangle$	$\langle 1,1\rangle$	1	1	$\langle 1,1\rangle$	$\langle 1,1\rangle$
1	$*$	$\langle 1,1\rangle$	$\langle 0,1\rangle$	$*$	1	$\langle 0,1\rangle$	$\langle 1,1\rangle$
$*$	0	$\langle 0,1\rangle$	$\langle 0,0\rangle$	0	$*$	$\langle 0,0\rangle$	$\langle 0,1\rangle$
$*$	1	$\langle 0,1\rangle$	$\langle 1,1\rangle$	$*$	1	$\langle 0,1\rangle$	$\langle 1,1\rangle$
$*$	$*$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$*$	$*$	$\langle 0,1\rangle$	$\langle 0,1\rangle$

Using this gadget, we can reduce an instance of THREE-VALUED CCV to an instance of CCV by doubling the number of wires, and replacing every three-valued comparator gate of the THREE-VALUED CCV instance with a gadget with two standard comparator gates simulating it.

The above construction shows how to reduce the question of whether a designated wire outputs 1 for a given instance of THREE-VALUED CCV to the question of whether a *pair* of wires of an instance of CCV output $\langle 1,1\rangle$. However for an instance of CCV we are only allowed to decide whether a *single* designated wire outputs 1. This technical difficulty can be easily overcome since we can use an $\wedge$-gate (one of the two outputs of a comparator gate) to test whether a pair of wires outputs $\langle 1,1\rangle$, and output the result on a single designated wire. □

Subramanian [1990, §6.2.6] generalizes the construction of Theorem 6.2 to 1-Lipschitz gates which are *uniparous*: if the input to a gate contains at most one non-star, then the output contains at most one non-star.

6.3. Algorithms for solving stable marriage problems

In this section, we develop a reduction from SM to CCV due to Subramanian [1990; 1994], and later extended to a more general class of problems by Feder [1992; 1995]. Subramanian did not reduce SM to CCV directly, but to the *network stability problem* built from the less standard X gate, which takes two inputs p and q and produces two outputs $p' = p \wedge \neg q$ and $q' = \neg p \wedge q$. It is important to note that the "*network*" notion in Subramanian's work denotes a generalization of circuits by allowing a connection from the output of a gate to the input of any gate including itself, and thus a network in his definition might contain cycles. An X-network is a network consisting only of X gates under the important restriction that each X gate has fan-out exactly one for each output it computes. The network stability problem for X gates (XNS) is then to decide if an X-network has a stable configuration, i.e., a way to assign Boolean values to the wires of the network so that the values are compatible with all the X gates of the network. Subramanian showed in his dissertation [Subramanian 1990] that SM, XNS and CCV are all equivalent under log space reductions.

We do not work with XNS in this paper since networks are less intuitive and do not have a nice graphical representation as comparator circuits. By utilizing Subramanian's idea, we give a direct AC^0 reduction from SM to CCV, using the three-valued variant of CCV developed in Section 6.2.

We will describe a sequence of algorithms, starting with Gale and Shapley's algorithm, which is historically the first algorithm solving the stable marriage problem, and ending with Subramanian's algorithm. All algorithms other than the Gale–Shapley algorithm and Subramanian's algorithm were invented specifically for the sake of this work.

6.3.1. Notation. Let M denote the set of men, and W denote the set of women; both are of size n. The preference list for a person p is given by

$$\pi_1(p) \succ_p \pi_2(p) \succ_p \cdots \succ_p \pi_n(p) \succ_p \bot.$$

The last place on the list is taken by the placeholder $\bot$ which represents p being unmatched, a situation less preferable than being matched. If p is a man then $\pi_1(p), \ldots, \pi_n(p)$ are women, and vice versa.

The preference relation $\succ_p$ is defined by $\pi_i(p) \succ_p \pi_j(p)$ whenever $i < j$; we say that p prefers $\pi_i(p)$ over $\pi_j(p)$. For a set of women W_0 and a man m, the woman m prefers the most is $\max_m W_0$; if W_0 is empty, then $\max_m W_0 = \bot$. Let S be a set, then we write $q \succ_p S$ to denote that p prefers q to any person in S; similarly, $q \prec_p S$ denotes that p prefers any person in S to q.

A marriage P is a set of pairs (m, w) which forms a perfect matching between the set of men and the set of women. In a marriage P, we let $P(p)$ denote the person p is married. A marriage is stable if there is no unstable pair (m, w), which is a pair satisfying $w \succ_m P(m)$ and $m \succ_w P(w)$, i.e.m and w prefer each other more than their current partner.

6.3.2. Gale–Shapley algorithm. Gale and Shapley's algorithm [Gale and Shapley 1962] proceeds in rounds. In the first round, each man proposes to his top woman among the ones he hasn't proposed, and each woman selects her most preferred suitor. In each subsequent round, each rejected man proposes to his next choice, and each woman selects her most preferred suitor (including her choice from the previous round). The situation eventually stabilizes, resulting in the man-optimal stable marriage.

There are many ways to implement the algorithm. One of them is illustrated below in Algorithm 1. The crucial object is the graph G, which is a set of possible matches. Each round, each man m selects the top woman $\text{top}(m)$ currently available to him. Among all men who chose her (if any), each woman w selects the best suitor $\text{best}(w)$. Whenever any man m is rejected by his top woman w, we remove the possible match (m, w) from G.

Algorithm 1 Gale–Shapley algorithm

$G \leftarrow \{(m, w) : m \in M, w \in W\}$
repeat
  $\text{top}(m) \leftarrow \max_m\{w : (m, w) \in G\}$ for all $m \in M$
  $\text{best}(w) \leftarrow \max_w\{m : \text{top}(m) = w\}$ for all $w \in W$
  Remove $(m, \text{top}(m))$ from G whenever $\text{best}(\text{top}(m)) \neq m$
until G stops changing
return $\{(m, \text{top}(m)) : m \in M\}$

LEMMA 6.3. *Algorithm 1 returns the man-optimal stable matching, and terminates after at most n^2 rounds.*

We present the well-known proof of this lemma in full since the analysis of the other algorithms parallels the analysis of the Gale–Shapley algorithm.

PROOF. **Admissibility:** If a pair (m, w) is removed from G, then no stable marriage contains the pair (m, w). This is proved by induction on the number of pairs removed. A pair (m, w) is removed when $w = \text{top}(m) = \text{top}(m')$ for some other man m', and $m' \succ_w m$. Suppose for a contradiction that P is a stable marriage and if $P(m) = w$. By the induction hypothesis, we know that m' can never be married to any woman w' such that $w' \succ_{m'} w$ since that edge (m', w') was removed previously. Thus $w \succeq_{m'} P(m')$. But then (m', w) would be an unstable pair, a contradiction.

Definiteness: For all men m and at all times, $\text{top}(m) \neq \perp$. For any man m, $(m, \pi_n(m))$ is never removed from G, and so $\text{top}(m)$ is always well-defined. Indeed, for each w, after each iteration $\text{best}(w)$ is non-decreasing in the preference order of w. So if $(m, w) \notin G$, $\text{best}(w) \succ_w m$. On the other hand, for any two women w and w', if $\text{best}(w), \text{best}(w') \neq \perp$ then $\text{best}(w) \neq \text{best}(w')$. Thus, if $(m, w) \notin G$ for all $w \in W$, then best is a injective mapping from W into $M \setminus \{m\}$, contradicting the pigeonhole principle.

Completeness: The output of the algorithm is a marriage. The algorithm ends when $\text{best}(\text{top}(m)) = m$ for every m, which implies that best and top are mutually inverse bijections.

Stability: The output of the algorithm is a stable marriage. Suppose (m, w) were an unstable pair, so at the end of the algorithm, $m \succ_w \text{best}(w)$ and $w \succ_m \text{top}(m)$ (we're using the fact that top and best are inverses at the end of the algorithm). However, $m \succ_w \text{best}(w)$ implies $\text{top}(m) \neq w$, which implies $\text{top}(m) \succ_m w$.

Optimality: The output of the algorithm is the man-optimal stable marriage. This is obvious, since each man gets his best choice among all possible stable marriages.

Runtime: The algorithm terminates in n^2 iterations since at most n^2 edges can be deleted from G. □

The Gale–Shapley algorithm has one disadvantage: it only computes the man-optimal stable matching. This is easy to rectify by symmetrizing the algorithm, resulting in Algorithm 2. While in the original algorithm, only the men propose (select their top choices), and only the women accept or reject (choose the most promising suitor), in the symmetric algorithm, both sexes participate in both tasks in parallel. The algorithm returns both the man-optimal and the woman-optimal stable marriages.

Algorithm 2 Symmetric Gale–Shapley algorithm

```
G ← {(m, w) : m ∈ M, w ∈ W}
repeat
    top(m) ← max_m{w : (m, w) ∈ G} for all m ∈ M
    top(w) ← max_w{m : (m, w) ∈ G} for all w ∈ W
    best(w) ← max_w{m : top(m) = w} for all w ∈ W
    best(m) ← max_m{w : top(w) = m} for all m ∈ M
    Remove (m, top(m)) from G whenever best(top(m)) ≠ m
    Remove (top(w), w) from G whenever best(top(w)) ≠ w
until G stops changing
return {(m, top(m)) : m ∈ M} and {(top(w), w) : w ∈ W} as the man-optimal and the woman-
optimal stable marriages respectively
```

LEMMA 6.4. *Algorithm 2 returns the man-optimal and woman-optimal stable matchings, and terminates after at most n^2 rounds.*

PROOF. The analysis is largely analogous to the analysis of the original algorithm. Every pair (m,w) removed from G belongs to no stable marriage. Furthermore, since a stable marriage exists, $\text{top}(m)$ and $\text{top}(w)$ are always defined after the algorithm finishes. At the end of the algorithm, top and best are mutually inverse bijections on $M \cup W$, hence the outputs are marriages. The same arguments as before show that the marriages returned by the algorithm are man-optimal and woman-optimal stable marriages respectively. Finally, the algorithm terminates in n^2 iterations since we can only remove at most n^2 edges G. □

6.3.3. Interval algorithms. At the end of Algorithm 2, for each man m, his partner in the man-optimal stable marriage is $\text{top}(m)$, while his partner in the woman-optimal stable marriage is $\text{best}(m)$. The same holds for women (with the roles of the sexes reversed). This prompts our next algorithm, Algorithm 3, which explicitly keeps track of an interval $J(p)$ of possible matches for each person p (these are intervals in the person's preference order).

At each round, each person p first picks their top choice $\text{top}(p)$. Then each person q picks their top suitor $\text{best}(q)$, if any. People over whom $\text{best}(q)$ is preferred are removed from $J(q)$. If p was rejected by his top choice $\text{top}(p)$, then $\text{top}(p)$ is removed from $J(p)$. These update rules maintain the contiguous nature of the intervals. The situation eventually stabilizes, and the algorithm returns the man-optimal and the woman-optimal stable marriages.

Algorithm 3 Interval algorithm

$J_0(m) \leftarrow W$ for all $m \in M$
$J_0(w) \leftarrow M$ for all $w \in W$
$t \leftarrow 0$
repeat
  $\text{top}_t(p) \leftarrow \max_p J_t(p)$ for all $p \in M \cup W$
  $\text{best}_t(q) \leftarrow \max_q \{p : q = \text{top}_t(p)\}$ for all $q \in M \cup W$
  Remove p from $J_t(q)$ whenever $p \prec_q \text{best}_t(q)$, for all p, q of opposite sex
  Remove $\text{top}_t(p)$ from $J_t(p)$ if $p \neq \text{best}_t(\text{top}_t(p))$
  $t \leftarrow t+1$
until $J_{t+1}(p) = J_t(p)$ for all $p \in M \cup W$
return $\{(m, \max_m J_t(m)) : m \in M\}$ and $\{(\max_w J_t(w), w) : w \in W\}$ as the man-optimal and the woman-optimal stable marriages respectively

LEMMA 6.5. *Algorithm 3 returns the man-optimal and woman-optimal stable matchings, and terminates after at most $2n^2$ rounds. Furthermore, the man-optimal and woman-optimal matchings are given by*

$$\{(m, \max_m J_t(m)) : m \in M\} \textit{ and } \{(\max_w J_t(w), w) : w \in W\} \textit{ respectively.}$$

PROOF.

Admissibility: In every stable marriage, every person p is matched to someone from $J(p)$. This is proved by induction on the number of rounds. A person q can be removed from $J(p)$ for one of two reasons: either $q \prec_p \text{best}(p)$, or $q = \text{top}(p)$ and $p \neq \text{best}(q)$. In the former case, if p were matched to q, then $(p, \text{best}(p))$ would be an unstable pair, since $p = \text{top}(\text{best}(p))$ implies that $\text{best}(p)$ prefers p to any other partner in $J(\text{best}(p))$. In the latter case, if p were matched to $q = \text{top}(p)$, then $(q, \text{best}(q))$ would be an unstable pair, since q prefers $\text{best}(q)$ over p by definition, and $\text{best}(q)$ prefers q as in the former case.

The remaining analysis of this algorithm is similar to the analysis of the Gale–Shapley algorithm. The outputs of the algorithm are marriages, since the algorithm ends when $\text{best}(\text{top}(p)) = p$ for all p, hence top and best are inverse bijections. The marriages are stable for the same reason given for the Gale–Shapley algorithm. They are man-optimal and woman-optimal for the same reason. The number of iterations is at most $2n^2$ since there are $2n$ intervals, each of initial length n.

Finally, at the termination of the algorithm, $\text{best}(q) = \max_q J(q)$. Since top and best are inverses, this explains the dual formulas for the man-optimal and woman-optimal matchings. □

Our next algorithm introduces a new twist. Instead of removing $\text{top}(p)$ from $J(p)$ whenever $p \neq \text{best}(\text{top}(p))$, we remove $\text{top}(p)$ from $J(p)$ whenever $p \notin J(\text{top}(p))$ as shown in Algorithm 4. The idea is that if at some point $p \neq \text{best}(\text{top}(p))$, then $\text{best}(\text{top}(p)) \succ_{\text{top}(p)} p$, so p is removed from $J(\text{top}(p))$. At the following iteration, $\text{top}(p)$ will be removed from $J(p)$ in reciprocity. Thus, Algorithm 4 emulates Algorithm 3 with a delay of one round. We will later show that the advantage of this strange rule is the nice representation of the same algorithm in three-valued logic which can then be transformed to Subramanian's algorithm, implementable by comparator circuits.

Algorithm 4 Delayed interval algorithm

$J_0(m) \leftarrow W$ for all $m \in M$
$J_0(w) \leftarrow M$ for all $w \in W$
$t \leftarrow 0$
repeat
  $\text{top}_t(p) \leftarrow \max_p J(p)$ for all $p \in M \cup W$
  $\text{best}_t(q) \leftarrow \max_q \{p : q = \text{top}_t(p)\}$ for all $q \in M \cup W$
  Remove p from $J_t(q)$ whenever $p \prec_q \text{best}_t(q)$, for all p, q of opposite sex
  Remove $\text{top}_t(p)$ from $J_t(p)$ if $p \notin J_t(\text{top}_t(p))$
  $t \leftarrow t+1$
until $J_{t+1}(p) = J_t(p)$ for all $p \in M \cup W$
return $\{(m, \max_m J_t(m)) : m \in M\}$ and $\{(\max_w J_t(w), w) : w \in W\}$ as the man-optimal and the woman-optimal stable marriages respectively

LEMMA 6.6. *Algorithm 3 returns the man-optimal and woman-optimal stable matchings, and terminates after at most $2n^2$ rounds. Furthermore, the man-optimal and woman-optimal matchings are given by*

$$\{(m, \max_m J_t(m)) : m \in M\} \text{ and } \{(\max_w J_t(w), w) : w \in W\} \text{ respectively.}$$

PROOF. Clearly Algorithm 4 is admissible, that is p is matched to someone from $J(p)$ in any stable matching. Furthermore, at the end of the algorithm, $p = \text{best}(\text{top}(p))$. Otherwise, there are two cases. If $p \in J(\text{top}(p))$, then p would be removed from $J(\text{top}(p))$, and the algorithm would continue. If $p \notin J(\text{top}(p))$, then $\text{top}(p)$ would be removed from $J(p)$, and the algorithm would continue; note that by definition, at the beginning of the round, $\text{top}(p) \in J(p)$.

The rest of the proof follows the one for Algorithm 3. □

COROLLARY 6.7. *The intervals at the end of Algorithm 3 coincide with the intervals at the end of Algorithm 4.*

PROOF. That follows immediately from the two formulas for the output. □

The delayed interval algorithm can be implemented using three-valued logic. The key is the following encoding of the intervals using matrices, which we call the *matrix representation*.

$$\mathscr{M}(m,w) = \begin{cases} 1 & \text{if } w \succeq_m \max_m J(m) \\ * & \text{if } \max_m J(m) \succ_m w \succeq_m \min_m J(m) \\ 0 & \text{if } \min_m J(m) \succ_m w \end{cases}$$

$$\mathscr{W}(w,m) = \begin{cases} 0 & \text{if } m \succeq_w \max_w J(w) \\ * & \text{if } \max_w J(w) \succ_w m \succeq_w \min_w J(w) \\ 1 & \text{if } \min_w J(w) \succ_w m \end{cases}$$

In other words, for every man m, the array $\mathscr{M}(m,\pi_1(m)),\ldots,\mathscr{M}(m,\pi_n(m))$ has the form

$$1 \cdots \boxed{1 \ * \ \cdots \ *}\ 0 \ \cdots \ 0$$

where the men whose corresponding values are contained in the box are precisely the men in $J(m)$.

For every woman w, the array $\mathscr{W}(w,\pi_1(w)),\ldots,\mathscr{W}(w,\pi_n(w))$ has the form

$$0 \cdots \boxed{0 \ * \ \cdots \ *}\ 1 \ \cdots \ 1$$

where the women whose corresponding values are contained in the box are precisely the women in $J(w)$.

Algorithm 5 is an implementation of Algorithm 4 using three-valued logic. We will show, in a sequence of steps, that at each point in time, the matrices representing the intervals in Algorithm 4 equal the matrices in Algorithm 5.

Algorithm 5 Delayed interval algorithm, three-valued logic formulation

$$\mathscr{M}_0(m,w) = \begin{cases} 1 & \text{if } w = \pi_1(m) \\ * & \text{otherwise} \end{cases}$$

$$\mathscr{W}_0(w,m) = \begin{cases} 0 & \text{if } m = \pi_1(w) \\ * & \text{otherwise} \end{cases}$$

$t \leftarrow 0$

repeat

$$\mathscr{M}_{t+1}(m,\pi_i(m)) = \begin{cases} 1 & \text{if } i = 1 \\ \mathscr{M}_t(m,\pi_{i-1}(m)) \wedge \bigwedge_{j \le i-1} \mathscr{W}_t(\pi_j(m),m) & \text{otherwise} \end{cases}$$

$$\mathscr{W}_{t+1}(w,\pi_i(w)) = \begin{cases} 0 & \text{if } i = 1 \\ \mathscr{W}_t(w,\pi_{i-1}(w)) \vee \bigvee_{j \le i-1} \mathscr{M}_t(\pi_j(w),w) & \text{otherwise} \end{cases}$$

$t \leftarrow t+1$

until $\mathscr{M}_t = \mathscr{M}_{t-1}$ and $\mathscr{W}_t = \mathscr{W}_{t-1}$

$S_M \leftarrow \{(m,w) : \mathscr{M}_t(m,w) = 1 \text{ and } \mathscr{W}_t(w,m) \in \{0,*\}\}$ % man-optimal stable marriage

$S_W \leftarrow \{(m,w) : \mathscr{W}_t(w,m) = 0 \text{ and } \mathscr{M}_t(m,w) \in \{1,*\}\}$ % woman-optimal stable marriage

return S_M, S_W

First, we show that the matrices properly encode intervals.

LEMMA 6.8. *At each time t in the execution of Algorithm 5, and for each man m, the sequence*

$$\mathscr{M}_t(m,\pi_1(m)),\ldots,\mathscr{M}_t(m,\pi_n(m))$$

is non-increasing (with respect to the order $1 > * > 0$). Similarly, for each woman w, the sequence

$$\mathscr{W}_t(w,\pi_1(w)),\ldots,\mathscr{W}_t(w,\pi_n(w))$$

is non-decreasing

PROOF. The proof is by induction. The claim is clearly true at time $t = 0$. For the inductive case, it suffices to analyze $\mathscr{M}$ since $\mathscr{W}$ can be handled dually. Furthermore, at each iteration, we "shift" each sequence $\mathscr{M}(m,\cdot)$ one step to the right and add a 1 to the left end to get the following non-increasing sequence

$$1, \mathscr{M}_{t-1}(m, \pi_1(m)), \mathscr{M}_{t-1}(m, \pi_2(m)), \ldots, \mathscr{M}_{t-1}(m, \pi_{n-1}(m))$$

Then we take a component-wise AND of the above sequence with the non-increasing sequence

$$1, \mathscr{W}_{t-1}(\pi_1(m), m), \big(\mathscr{W}_{t-1}(\pi_1(m), m) \wedge \mathscr{W}_{t-1}(\pi_2(m), m)\big), \ldots, \big(\mathscr{W}_{t-1}(\pi_1(m), m) \wedge \cdots \wedge \mathscr{W}_{t-1}(\pi_{n-1}(m), m)\big).$$

It's not hard to check that the result is also a non-increasing sequence by the properties of three-valued logic. □

Second, we show that the intervals encoded by the matrices can only shrink. This is the same as saying that whenever an entry gets determined (to a value different from $*$), it remains constant.

LEMMA 6.9. *If for some time t, for some man m and some woman w, $\mathscr{M}_t(m, w) \in \{0, 1\}$, then $\mathscr{M}_s(m, w) = \mathscr{M}_t(m, w)$ for $s \geq t$. A similar claim holds for $\mathscr{W}$.*

PROOF. We prove the claim by induction on t. Let $w = \pi_i(m)$. If $i = 1$, then the claim is trivial. Now suppose $i > 1$. If $\mathscr{M}_t(m, \pi_i(m)) = 1$, then

$$\mathscr{M}_{t-1}(m, \pi_{i-1}(m)) = \mathscr{W}_{t-1}(\pi_1(m), m) = \cdots = \mathscr{W}_{t-1}(\pi_{i-1}(m), m) = 1.$$

The induction hypothesis shows that all these elements retain their values in the next iteration and hence $\mathscr{M}_{t+1}(m, \pi_{i+1}(m)) = 1$. If $\mathscr{M}_t(m, \pi_i(m)) = 0$, then at least one of these elements is equal to zero; this element retains its value in the next iteration by the induction hypothesis; hence $\mathscr{M}_{t+1}(m, \pi_{i+1}(m)) = 0$. □

It remains to show that the way that the underlying intervals are updated matches the update rules of Algorithm 4.

LEMMA 6.10. *At each time t, the matrix representation of the intervals in Algorithm 4 is the same as the matrices $\mathscr{M}_t, \mathscr{W}_t$ in the execution of Algorithm 5. Furthermore, both algorithms return the same marriages.*

PROOF. The proof is by induction on t. The base case $t = 0$ is clear by inspection.

We now compare the update rules in some round t of both algorithms. There are two ways an interval $J(m)$ can be updated: either a woman is removed from the bottom of the interval, or a woman is removed from the top of the interval.

In the former case, a woman $\pi_i(m)$ is removed from $J_{t+1}(m)$ since $\pi_i(m) \prec_m \text{best}_t(m)$. Suppose $\text{best}_t(m) = \pi_j(m)$, where $j < i$. Since $m = \text{top}_t(\pi_j(m))$, we know that $\mathscr{W}_t(\pi_j(m), m) = 0$, and so

$$\mathscr{M}_{t+1}(m, \pi_i(m)) = \mathscr{M}_t(m, \pi_{i-1}(m)) \wedge \bigwedge_{j \leq i-1} \mathscr{W}_t(\pi_j(m), m) = 0. \tag{6.1}$$

Conversely, suppose $\mathscr{M}_{t+1}(m, \pi_i(m)) = 0$ while $\mathscr{M}_t(m, \pi_i(m)) = *$. Since $\mathscr{M}_t(m, \cdot)$ is non-increasing, we have $\mathscr{M}_t(m, \pi_{i-1}(m)) \neq 0$. Thus for $\mathscr{M}_{t+1}(m, \pi_i(m)) = 0$, we must have $\mathscr{W}_t(\pi_j(m), m) = 0$ for some $j < i$. Now suppose $m \neq \text{top}_t(\pi_j(m))$, then that equation (6.1) were true at an earlier time $s < t$, at which $\mathscr{M}_s(m, \pi_i(m))$ would have become 0. Hence $m = \text{top}_t(\pi_j(m))$, and $\pi_i(m)$ is removed from $J_{t+1}(m)$.

In the latter case, a woman $\pi_i(m)$ is removed from $J_{t+1}(m)$ since $\pi_i(m) = \text{top}_t(m)$ and $m \notin J_t(\pi_i(m))$. We claim that $m \prec_{\pi_i(m)} J(\pi_i(m))$, since otherwise $m \succ_{\pi_i(m)} J(\pi_i(m))$. Thus $(m, \pi_i(m))$ would be an unstable pair in any marriage produced by the algorithm (m will be matched to a woman inferior to $\text{top}_t(m) = \pi_i(m)$, and $\pi_i(m)$ will be matched to a man from $J(\pi_i(m))$ whom $\pi_i(m)$ doesn't like as much as m), and this contradicts that the algorithm produces some stable marriage. Hence we have $m \prec_{\pi_i(m)} J(\pi_i(m))$ and so $\mathcal{W}_t(\pi_i(m), m) = 1$. For $j \le i-1$, the woman $\pi_j(m)$ must have been removed from $J(m)$ in the past (since women can only be removed from the top of $J(m)$ one at a time), and at that time $\mathcal{W}(\pi_j(m), m) = 1$ (just as at the current time, $\mathcal{W}_t(\pi_i(m), m) = 1$); by Lemma 6.9, this value stays 1. Hence

$$\mathcal{M}_{t+1}(m, \pi_{i+1}(m)) = \mathcal{M}_t(m, \pi_i(m)) \wedge \bigwedge_{j \le i} \mathcal{W}_t(\pi_j(m), m) = 1.$$

Conversely, suppose $\mathcal{M}_{t+1}(m, \pi_{i+1}(m)) = 1$ while $\mathcal{M}_t(m, \pi_{i+1}(m)) = *$. Since $\mathcal{M}_t(m, \pi_i(m)) = 1$, we must have $\pi_i(m) = \mathrm{top}_t(m)$. Further, $\mathcal{W}_t(\pi_i(m), m) = 1$ shows that $m \notin J_t(\pi_i(m))$. Therefore $\pi_i(m)$ is removed from $J_{t+1}(m)$.

Finally, we conclude that the two algorithms return the same man-optimal and woman-optimal stable marriages by Lemma 6.8. □

Let us illustrate the workings of Algorithm 5. The following diagram illustrates a situation in which a man's interval shrinks from the bottom. The diagram illustrates $\mathcal{W}_t(w, \cdot), \mathcal{M}_t(m, \cdot), \mathcal{M}_{t+1}(m, \cdot)$, respectively, where $m = \mathrm{top}_t(w)$. The two elements in red form a pair $\mathcal{M}(m, w), \mathcal{W}(w, m)$.

$$\begin{array}{l}
\qquad\qquad\qquad m \\
\mathcal{W}_t(w,\cdot)\ \ 0\ \boxed{{\color{red}0}\ *\ *\ *\ *\ *}\ 1\ 1 \\
\qquad\qquad\qquad\qquad w \\
\mathcal{M}_t(m,\cdot)\ \ 1\ \ 1\ \boxed{1\ {\color{red}*}\ *\ *}\ 0\ 0\ 0 \\
\qquad\qquad\qquad\qquad\quad \Downarrow \\
\mathcal{M}_{t+1}(m,\cdot)\ \ 1\ \ 1\ \boxed{1\ {\color{red}*}}\ 0\ 0\ 0\ 0\ 0
\end{array}$$

The next diagram illustrates a situation in which a man's interval shrinks from the top. This time $w = \mathrm{top}_t(m)$.

$$\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad m \\
\mathcal{W}_t(w,\cdot)\ \ 0\ \boxed{0\ *\ *\ *\ *\ *}\ 1\ \ {\color{red}1} \\
\qquad\qquad\qquad w \\
\mathcal{M}_t(m,\cdot)\ \ 1\ 1\ \boxed{{\color{red}1}\ \ *\ *\ *}\ 0\ 0\ \ 0 \\
\qquad\qquad\qquad\quad \Downarrow \\
\mathcal{M}_{t+1}(m,\cdot)\ \ 1\ 1\ \ {\color{red}1}\ \boxed{1\ *\ *}\ 0\ 0\ \ 0
\end{array}$$

6.3.4. Subramanian's algorithm. Subramanian's algorithm is very similar to Algorithm 5. The latter algorithm is not implementable using comparator circuits, since (for example) the value $\mathcal{W}_t(\pi_1(m), m)$ is used in the update rule of

$$\mathcal{M}_{t+1}(m, \pi_2(m)), \ldots, \mathcal{M}_{t+1}(m, \pi_n(m)).$$

Subramanian's algorithm, displayed as Algorithm 6, corrects this issue by retaining only the most important term in each conjunction and disjunction.

In the analysis of Algorithm 5, we already saw that Subramanian's update rule works when an entry of $\mathcal{M}$ is set to 1. When Algorithm 5 sets an entry to 0, say $\mathcal{M}_{t+1}(m, \pi_i(m)) = 0$, it is due to $\mathcal{W}_t(\pi_j(m), m) = 0$ for some $j \le i-1$. If $j = i-1$, then Subramanian's algorithm will also set $\mathcal{M}_{t+1}(m, \pi_i(m)) = 0$. Otherwise, Subramanian's algorithm will set $\mathcal{M}_{t+1}(m, \pi_{j+1}(m)) = 0$, and this zero will propagate, so that $\mathcal{M}_{t+i-j}(m, \pi_i(m)) = 0$. This shows that in some sense, Subramanian's algorithm mimics Algorithm 5. It remains to show that Subramanian's algorithm computes the same final $\mathcal{M}$ and $\mathcal{W}$ as Algorithm 5.

First, we notice that the termination conditions of both algorithms are really the same. For Subramanian's algorithm, the conditions are that for $i > 1$,

$$\begin{aligned}
\mathcal{M}(m, \pi_i(m)) &= \mathcal{M}(m, \pi_{i-1}(m)) \wedge \mathcal{W}(\pi_{i-1}(m), m), \\
\mathcal{W}(w, \pi_i(w)) &= \mathcal{W}(w, \pi_{i-1}(w)) \vee \mathcal{M}(\pi_{i-1}(w), w).
\end{aligned} \tag{6.2}$$

Algorithm 6 Subramanian's algorithm

$$\mathcal{M}_0(m,w) = \begin{cases} 1 & \text{if } w = \pi_1(m) \\ * & \text{otherwise} \end{cases}$$
$$\mathcal{W}_0(w,m) = \begin{cases} 0 & \text{if } m = \pi_1(w) \\ * & \text{otherwise} \end{cases}$$
$t \leftarrow 0$
repeat
$$\mathcal{M}_{t+1}(m,\pi_i(m)) = \begin{cases} 1 & \text{if } i = 1 \\ \mathcal{M}_t(m,\pi_{i-1}(m)) \wedge \mathcal{W}_t(\pi_{i-1}(m),m) & \text{otherwise} \end{cases}$$
$$\mathcal{W}_{t+1}(w,\pi_i(w)) = \begin{cases} 0 & \text{if } i = 1 \\ \mathcal{W}_t(w,\pi_{i-1}(w)) \vee \mathcal{M}_t(\pi_{i-1}(w),w) & \text{otherwise} \end{cases}$$
$t \leftarrow t+1$
until $\mathcal{M}_t = \mathcal{M}_{t-1}$ and $\mathcal{W}_t = \mathcal{W}_{t-1}$
$S_M \leftarrow \{(m,w) : \mathcal{M}_t(m,w) = 1 \text{ and } \mathcal{W}_t(w,m) \in \{0,*\}\}$ % man-optimal stable marriage
$S_W \leftarrow \{(m,w) : \mathcal{W}_t(w,m) = 0 \text{ and } \mathcal{M}_t(m,w) \in \{1,*\}\}$ % woman-optimal stable marriage
return S_M, S_W

For Algorithm 5, the termination conditions are

$$\begin{aligned} \mathcal{M}(m,\pi_i(m)) &= \mathcal{M}(m,\pi_{i-1}(m)) \wedge \bigwedge_{j \le i-1} \mathcal{W}(\pi_j(m),m), \\ \mathcal{W}(w,\pi_i(w)) &= \mathcal{W}(w,\pi_{i-1}(w)) \vee \bigvee_{j \le i-1} \mathcal{M}(\pi_j(w),w). \end{aligned} \tag{6.3}$$

LEMMA 6.11. *The matrices $\mathcal{M},\mathcal{W}$ at the end of Subramanian's algorithm satisfy the termination conditions of Algorithm 5, and vice versa. Moreover, these are always matrix representations of intervals.*

PROOF. We observe in both algorithms the update rules guarantee that $\mathcal{M}(m,\cdot)$ is monotone non-increasing and that $\mathcal{W}(w,\cdot)$ is monotone non-decreasing, which implies

$$\mathcal{M}(\pi_{i-1}(w),w) = \bigvee_{j \le i-1} \mathcal{M}(\pi_j(w),w), \qquad \mathcal{W}(\pi_{i-1}(m),m) = \bigwedge_{j \le i-1} \mathcal{W}(\pi_j(m),m).$$

Thus, these two termination conditions are equivalent. □

We call a pair of matrices $(\mathcal{M},\mathcal{W})$ a *feasible pair* if they satisfy the equations in (6.2) or equivalently in (6.3)o, and furthermore $\mathcal{M}(m,\pi_1(m)) = 1$ and $\mathcal{W}(w,\pi_1(w)) = 0$ for all man m and woman w. The following lemma shows that, in some sense, Subramanian's algorithm is admissible.

LEMMA 6.12. *Let $\mathcal{M},\mathcal{W}$ be the matrices at the end of Subramanian's algorithm. If $\mathcal{M}(m,w) = c \neq *$ for some m, w, then $\mathcal{M}'(m,w) = c$ for any feasible pair $(\mathcal{M},\mathcal{W})$. Same for $\mathcal{W}$.*

PROOF. The proof is by induction on the time t in which $\mathcal{M}_t(m,w)$ is set to c. If $t = 0$, then the claim follows from the definition of feasible pair. Otherwise, for some i we have $\mathcal{M}_t(m,\pi_i(m)) = \mathcal{M}_{t-1}(m,\pi_{i-1}(m)) \wedge \mathcal{W}_{t-1}(\pi_{i-1}(m),m)$. If $c = 1$ then $\mathcal{M}_{t-1}(m,\pi_{i-1}(m)) = \mathcal{W}_{t-1}(\pi_{i-1}(m),m) = 1$, and by induction these entries get the same value in all feasible pairs. The definition of feasible pair then implies that $\mathcal{M}'(m,\pi_i(m)) = 1$ in any feasible pair $\mathcal{M}',\mathcal{W}'$. The case when $c = 0$ can be shown similarly. □

The following lemma shows that we can uniquely extract a stable marriage from each 0/1-valued feasible pair, i.e. both of the matrices in the pair have 0/1 values, and vice versa.

LEMMA 6.13. *Suppose $(\mathcal{M}, \mathcal{W})$ is a 0/1-valued feasible pair. If we marry each man m to $\min_m\{w : \mathcal{M}(m,w) = 1\}$, and each woman w to $\min_w\{m : \mathcal{W}(m,w) = 0\}$, then the result is a stable marriage.*

Conversely, every stable marriage P can be encoded as a 0/1-valued feasible pair, as follows: For each man m, we put $\mathcal{M}(m,w) = 1$ if $w \succeq_m P(m)$, and $\mathcal{M}(m,w) = 0$ otherwise. For each woman w, we put $\mathcal{W}(w,m) = 0$ if $m \succeq_w P(w)$, and $\mathcal{W}(w,m) = 1$ otherwise.

The two mappings are inverses of each other.

PROOF. **Feasible pair implies stable marriage.** Suppose $(\mathcal{M}, \mathcal{W})$ is a 0/1-valued feasible pair. We start by showing that the mapping P in the statement of the lemma is indeed a marriage.

We call a person *desperate* if he or she is married to the last choice in his or her preference list.

If a man m is not desperate then for some $1 \le i < n$, $\mathcal{M}(m, \pi_i(m)) = 1$ and $\mathcal{M}(m, \pi_{i+1}(m)) = 0$. This can only happen if $\mathcal{W}(\pi_i(m), m) = 0$, and furthermore if $m \succ_{\pi_i(m)} m'$, then $\mathcal{W}(\pi_i(m), m') = 1$ due to $\mathcal{M}(m, \pi_i(m)) = 1$. This shows that whenever a man m is not desperate, m is married to someone in the marriage P.

If m is desperate and $\mathcal{W}(\pi_i(m), m) = 0$, then m is married as before. Otherwise, no woman is desperate, and so similarly to the previous argument, it can be shown that every woman w is married in P. However, the fact that $P(w) \ne m$ for all women w contradicts the pigeonhole principle. Thus, we conclude that P is a marriage.

It remains to show that P is stable. Suppose that (m, w) were an unstable pair in P. Then $\mathcal{M}(m,w) = 1$ and $\mathcal{W}(w,m) = 0$, and moreover $\mathcal{M}(m,w') = 1$ for some $w' \prec_m w$. Yet (6.3) shows that $\mathcal{M}(m,w') \le \mathcal{W}(w,m)$, and we reach a contradiction.

Stable marriage implies feasible pair. Suppose m is matched to $\pi_k(m)$. Consider first the case $i \le k$. Then $\mathcal{M}(m, \pi_i(m)) = \mathcal{M}(m, \pi_{i-1}(m)) = 1$, and we have to show that $\mathcal{W}(\pi_{i-1}(m), m) = 1$. If the latter weren't true then $(m, \pi_{i-1}(m))$ would be an unstable pair, since $\pi_{i-1}(m) \succ_m \pi_i(m)$ while $\mathcal{W}(\pi_{i-1}(m), m) = 0$ implies that $\pi_{i-1}(m)$ prefers m to every other man which is matched to her. If $i = k+1$ then $\mathcal{M}(m, \pi_i(m)) = 0$ and also $\mathcal{W}(\pi_{i-1}(m), m) = 0$. If $i > k$ then $\mathcal{M}(m, \pi_i(m)) = \mathcal{M}(m, \pi_{i-1}(m)) = 0$.

The mappings are inverses of each other. It is easy to check directly from the definition that if we start with a stable marriage P, convert it to a feasible pair $(\mathcal{M}, \mathcal{W})$, and convert it back into a stable marriage P', then $P = P'$.

For the other direction, Lemma 6.11 shows that if $(\mathcal{M}, \mathcal{W})$ is a 0/1-valued feasible pair then for each man m, $\mathcal{M}(m, \cdot)$ consists of a positive number of 1s followed by 0s, and dually for each woman w, $\mathcal{W}(w, \cdot)$ consists of a positive number of 0s followed by 1s. Thus, given the fact that our rule of converting $(\mathcal{M}, \mathcal{W})$ to a stable marriage P indeed results in a marriage, it is clear that converting P back to a feasible pair results in $(\mathcal{M}, \mathcal{W})$. □

LEMMA 6.14. *Subramanian's algorithm returns the man-optimal and woman-optimal stable marriages. Furthermore, the matrices $\mathcal{M}, \mathcal{W}$ at the end of Subramanian's algorithm coincide with the matrices $\mathcal{M}, \mathcal{W}$ at the end of Algorithm 5.*

PROOF. The monotonicity of $\wedge$ and $\vee$ shows that if we replace every $*$ in $\mathcal{M}, \mathcal{W}$ with 0, then the resulting $(\mathcal{M}, \mathcal{W})$ is still a feasible pair; the same holds if we replace every $*$ with 1.

Lemma 6.12 and Lemma 6.13 together imply that the first output is the man-optimal stable matching, and the second output is the woman-optimal stable matching. Lemma 6.11 shows that at termination, the matrices $\mathcal{M}, \mathcal{W}$ are matrix representations of intervals, hence they must coincide with the matrices at the end of Algorithm 5. □

LEMMA 6.15. *Subramanian's algorithm terminates after at most $2n^2$ iterations.*

PROOF. Since there are $2n^2$ entries in both matrices, and at each iteration at least one entry changes from $*$ to 0 or 1, the algorithm terminates after at most $2n^2$ iterations. □

A formal correctness proof of Subramanian's algorithm can be found in [Lê et al. 2011].

6.3.5. MoSm and WoSm are AC^0 many-one reducible to Ccv. In the remaining section, we will show that Subramanian's algorithm can be implemented as a three-valued comparator circuit.

First, since for each man m, the pair of values $\mathscr{M}_t(m, \pi_{i-1}(m))$ and $\mathscr{W}_t(\pi_{i-1}(m), m)$ is only used once to compute the two outputs $\mathscr{M}_t(m, \pi_{i-1}(m)) \wedge \mathscr{W}_t(\pi_{i-1}(m), m)$ and $\mathscr{M}_t(m, \pi_{i-1}(m)) \vee \mathscr{W}_t(\pi_{i-1}(m), m)$, and then each output is used at most once when updating $\mathscr{M}_{t+1}(m, \pi_i(m))$ and $\mathscr{W}_{t+1}(m, \pi_i(m))$. Thus the whole update rule can be easily implemented using comparator gates.

Second, we know that the algorithm converges within $2n^2$ iterations to a fixed point. Therefore, if we run the loop for exactly $2n^2$ iterations, the result would be the same. Hence, we can build a comparator circuit to simulate exactly $2n^2$ iterations of Subramanian's algorithm.

Finally, we can extract the man-optimal stable matching using a simple comparator circuit with negation gates. Recall that the logical values $0, *, 1$ are represented in reality by pairs of wires with values $(0,0), (0,1), (1,1)$. In the man-optimal stable matching, a man m is matched to $\pi_i(m)$ if $\mathscr{M}(m, \pi_i(m)) = 1$ and either $i = n$ or $\mathscr{M}(m, \pi_{i+1}(m)) \in \{0, *\}$. In the latter case, if the corresponding wires are (α, β) and (γ, δ), then the required information can be extracted as $\alpha \wedge \beta \wedge \neg\gamma$.

THEOREM 6.16. MoSm *and* WoSm *are* AC^0 *many-one reducible to* Ccv¬.

PROOF. We will show only the reduction from MoSm to Ccv¬ since the reduction from WoSm to Ccv¬ works similarly.

Following the above construction, we can define an AC^0 function that takes as input an instance of MoSm with preference lists for all the men and women, and produces a three-valued comparator circuit that implements $2n^2$ iterations of Subramanian's algorithm, and then extracts the man-optimal stable matching. □

Corollary 5.2 and Theorems 6.2, 6.1 and 6.16 give us the following corollary.

COROLLARY 6.17. *The ten problems* MoSm, WoSm, Sm, Ccv, Ccv¬, Three-valued Ccv, 3Lfmm, Lfmm, 3vLfmm *and* vLfmm *are all equivalent under* AC^0 *many-one reductions.*

PROOF. Corollary 5.2 and Theorem 6.2 show that Ccv, Ccv¬, Three-valued Ccv, 3Lfmm, Lfmm, 3vLfmm and vLfmm are all equivalent under AC^0 many-one reductions.

Theorem 6.16 shows that MoSm and WoSm are AC^0 many-one reducible to Three-valued Ccv. Theorem 6.1 also shows that 3Lfmm is AC^0 many-one reducible to MoSm, WoSm, and Sm. Hence, MoSm, WoSm, and Sm are equivalent to the above problems under AC^0 many-one reductions. □

7. CONCLUSION

Although we have shown that there are problems in relativized NC but not in relativized CC (uniform or nonuniform), it is quite possible that some of the standard problems in NC^2 that are not known to be in NL might be in (nonuniform) CC . Examples are integer matrix powering and context free languages (or more generally problems in LogCFL). Of particular interest is matrix powering over the field $GF(2)$. We cannot show this is in CC, even though we know that Boolean matrix powering is in CC because $\mathsf{NL} \subseteq \mathsf{CC}$. Another example is the matching problem for bipartite graphs or general undirected graphs, which is in RNC^2 [Karp et al. 1986; Mulmuley et al. 1987] and hence in nonuniform NC^2. It would be interesting to show that some (relativized) version of any of these problems is, or is not, in (relativized) (nonuniform) CC.

Let SucCC be the class of problems p-reducible to succinct CC (where a description of an exponential size comparator circuit is given using linear size Boolean circuits). It is easy to show that SucCC lies between PSPACE and EXPTIME, but we we are unable to show that it is equal to either class[3]. We are not aware of other complexity classes which appear to lie properly between PSPACE and EXPTIME.

---

[3]Thanks to Scott Aaronson for pointing this out.

We have defined CC in terms of uniform families comparator circuits, analogously to the way that complexity classes such as uniform AC^i and NC^i are defined. The latter also have machine characterizations: NC^i is the class of relations computable by alternating Turing machines in space $O(\log n)$ and $O(\log^i n)$ alternations, and also the class of relations computable by polynomially many processors in time $O(\log^i n)$. Similarly, P can be defined either as those relations computable in polynomial time, or as those relations computable by AC^0-uniform polynomial size circuits. An important open question, appearing already in Subramanian's thesis [1990], is to come up with a similar machine model for CC. While we do outline some machine characterization in Theorem 3.10, the machine model we use is not as natural as the ones for P and NC, and so it is still an open question to come up with a natural machine model for CC.

We believe that CC deserves more attention, since on the one hand it contains interesting complete problems, on the other hand the limitation of fanout restriction of comparator gates has not yet been studied outside this paper. Furthermore, CC provides us another research direction for separating NL from P by analyzing the limitation of the fanout restriction.

REFERENCES

K. Aehlig, S. Cook, and P. Nguyen. 2007. Relativizing Small Complexity Classes and Their Theories. In *Computer Science Logic*, Jacques Duparc and Thomas Henzinger (Eds.). Lecture Notes in Computer Science, Vol. 4646. Springer Berlin / Heidelberg, 374–388.

R. Anderson and E. Mayr. 1987. Parallelism and greedy algorithms. In *Parallel and distributed computing (Advances in computing research)*, Vol. 4. JAI Press, 17–38.

David A. Mix Barrington, Neil Immerman, and Howard Straubing. 1990. On uniformity within {NC1}. *J. Comput. System Sci.* 41, 3 (1990), 274–306. DOI:http://dx.doi.org/10.1016/0022-0000(90)90022-D

K.E. Batcher. 1968. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference 32*. ACM, 307–314.

S. Cook and P. Nguyen. 2010. *Logical foundations of proof complexity.* Cambridge University Press.

Stephen A. Cook. 1985. A taxonomy of problems with fast parallel algorithms. *Information and Control* 64, 1-3 (1985), 2–22. DOI:http://dx.doi.org/10.1016/S0019-9958(85)80041-3

S. A. Cook, D. T. M. Lê, and Y. Ye. 2011. Complexity Classes and Theories for the Comparator Circuit Value Problem. *arXiv* abs/1106.4142 (2011).

A. L. Delcher and S. Rao Kosaraju. 1995. An NC Algorithm for Evaluating Monotone Planar Circuits. *SIAM J. Comput.* 24, 2 (April 1995), 369–375. DOI:http://dx.doi.org/10.1137/S0097539792226278

T. Feder. 1992. A new fixed point approach for stable networks and stable marriages. *J. Comput. System Sci.* 45, 2 (1992), 233–284.

T. Feder. 1995. *Stable Networks and Product Graphs.* American Mathematical Society, Boston, MA, USA.

D. Gale and L.S. Shapley. 1962. College admissions and the stability of marriage. *The American Mathematical Monthly* 69, 1 (1962), 9–15.

Leslie M. Goldschlager. 1977. The monotone and planar circuit value problems are log space complete for P. *SIGACT News* 9, 2 (July 1977), 25–29. DOI:http://dx.doi.org/10.1145/1008354.1008356

R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. 1995. *Limits to parallel computation: P-completeness theory.* Oxford University Press.

R. Greenlaw and S. Kantabutra. 2008. On the parallel complexity of hierarchical clustering and CC-complete problems. *Complexity* 14, 2 (2008), 18–28.

R.M. Karp, E. Upfal, and A. Wigderson. 1986. Constructing a perfect matching is in random NC. *Combinatorica* 6, 1 (1986), 35–48.

Karp, R.M. and Wigderson, A. 1985. A fast parallel algorithm for the maximal independent set problem. *J. ACM* 32, 4 (1985), 762–773.

D. T. M. Lê, S. A. Cook, and Y. Ye. 2011. A Formal Theory for the Complexity Class Associated with the Stable Marriage Problem. In *Computer Science Logic (CSL'11) - 25th International Workshop/20th Annual Conference of the EACSL (Leibniz International Proceedings in Informatics (LIPIcs))*, Marc Bezem (Ed.), Vol. 12. Dagstuhl, Germany, 381–395.

E.W. Mayr and A. Subramanian. 1992. The complexity of circuit value and network stability. *J. Comput. System Sci.* 44, 2 (1992), 302–323.

C. Moore and J. Machta. 2000. Internal Diffusion-Limited Aggregation: Parallel Algorithms and Complexity. *Journal of Statistical Physics* 99 (2000), 661–690. Issue 3.

K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. 1987. Matching is as easy as matrix inversion. *Combinatorica* 7, 1 (1987), 105–113.

V. Ramachandran and L.-C. Wang. 1991. Parallel algorithm and complexity results for telephone link simulation. In *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*. 378 –385.

Vijaya Ramachandran and Honghua Yang. 1996. An Efficient Parallel Algorithm for the General Planar Monotone Circuit Value Problem. *SIAM J. Comput.* 25, 2 (Feb. 1996), 312–339. DOI:http://dx.doi.org/10.1137/S0097539793260775

A. Subramanian. 1990. *The computational complexity of the circuit value and network stability problems.* Ph.D. Dissertation. Dept. of Computer Science, Stanford University.

A. Subramanian. 1994. A new approach to stable matching problems. *SIAM J. Comput.* 23, 4 (1994), 671–700.

Honghua Yang. 1991. An NC algorithm for the general planar monotone circuit value problem. In *Parallel and Distributed Processing, 1991. Proceedings of the Third IEEE Symposium on.* 196–203. DOI:http://dx.doi.org/10.1109/SPDP.1991.218279