Electronic Colloquium on Computational Complexity, Report No. 93 (2013)

A Generalization of Spira's Theorem and Circuits with Small Segregators or Separators

Anna Gál* Jing-Tang Jang†
Dept. of Computer Science, University of Texas at Austin,
Austin, TX 78712-1188, USA

June 20, 2013

Abstract

Spira [36] showed that any Boolean formula of size s can be simulated in depth $O(\log s)$. We generalize Spira's theorem and show that any Boolean *circuit* of size s with segregators of size $f(s)$ can be simulated in depth $O(f(s)\log s)$. If the segregator size is at least s^{ε} for some constant $\varepsilon > 0$, then we can obtain a simulation of depth $O(f(s))$. This improves and generalizes a simulation of polynomial-size Boolean circuits of constant treewidth k in depth $O(k^2 \log n)$ by Jansen and Sarma [21]. Since the existence of small balanced separators in a directed acyclic graph implies that the graph also has small segregators, our results also apply to circuits with small separators. Our results imply that the class of languages computed by non-uniform families of polynomial-size circuits that have constant size segregators equals non-uniform NC^1.

Considering space bounded Turing machines to generate the circuits, for $f(s)\log^2 s$-space uniform families of Boolean circuits our small-depth simulations are also $f(s)\log^2 s$-space uniform. As a corollary, we show that the Boolean Circuit Value problem for circuits with constant size segregators (or separators) is in deterministic $SPACE(\log^2 n)$. Our results also imply that the Planar Circuit Value problem, which is known to be P-Complete [19], is in $SPACE(\sqrt{n}\log n)$. We also show that the Layered Circuit Value and Synchronous Circuit Value problems, which are both P-complete [20], are in $SPACE(\sqrt{n})$.

*Email: panni@cs.utexas.edu. Supported in part by NSF Grant CCF-1018060

†Email: keith@cs.utexas.edu. Supported in part by MCD fellowship from Dept. of Computer Science, University of Texas at Austin, and NSF Grant CCF-1018060

ISSN 1433-8092

1 Introduction

Spira [36] proved the following theorem.

Theorem A. *[36] Let F be any Boolean formula of size s. Then F can be simulated by an equivalent formula of depth $O(\log s)$.*

There are several results improving or extending Spira's theorem. Bonet and Buss [4] improved the constants in the depth bounds and the size of the simulation for Boolean formulas. Spira originally considered formulas over the $\{\wedge, \vee, \neg\}$ basis. Savage [34] generalized the result to all complete bases. Wegener [38] proved the statement for monotone Boolean formulas. Brent [6], Bshouty et. al. [7] extended it to arithmetic formulas. All these results study formulas, i.e. tree-like circuits with fan-out 1.

Valiant, Skyum, Berkowitz and Rackoff [37] showed that arithmetic circuits of size s and degree d can be simulated by arithmetic circuits of size $O((sd)^{O(1)})$ and $O(\log s \log d)$ depth. However, very little is known for size vs. depth for general Boolean circuits. The strongest results so far for general Boolean circuits by Paterson and Valiant [28], and Dymond and Tompa [14] give a simulation of arbitrary Boolean circuits of size s in depth $O(s/\log s)$.

In this paper, we generalize Spira's technique to circuits with small segregators or small separators. Informally, the separator of a graph is a subset of the nodes whose removal yields two subgraphs of comparable sizes. (See the following section for a formal definition.) Graphs with small separators include trees, planar graphs [25], graphs with bounded genus [18], graphs with excluded minors [1], as well as graphs with bounded treewidth [32].

Segregators are a relaxed version of separators of directed acyclic graphs. Paul et al. [29], and Santhanam [33] used segregators to study the computation graph of Turing machines. Directed acyclic graphs with small separators also have small segregators, but the reverse may not necessarily hold. See Section 2 for more details.

Jansen and Sarma [21] studied the question of simulating Boolean circuits with bounded treewidth by small-depth circuits. They showed that polynomial-size circuits with constant treewidth k can be simulated in depth $O(k^2 \log n)$, and thus the class of languages with non-uniform polynomial-size bounded treewidth circuits equals non-uniform NC^1.

We extend this result to arbitrary circuits with small segregators and show that any Boolean circuit of size s with segregators (or separators) of size $f(s)$ can be simulated in depth $O(f(s) \log s)$. For circuits with segregators of size k, thus also for graphs with treewidth k, this gives a simulation in depth $k \log s$, improving the bound in [21]. If the segregator size is at least s^{ε} for some constant $\varepsilon > 0$, then we can obtain a simulation of depth $O(f(s))$. Our results imply that the class of languages computed by non-uniform families of polynomial-size circuits that have constant-size segregators equals non-uniform NC^1.

Barrington [2] showed that the class of languages decided by branching programs of polynomial size and constant width is the same as NC^1 (in both the uniform and non-uniform

settings). In the non-uniform setting, our results together with Barrington's result imply that the class of languages decided by constant-width branching programs of polynomial size is the same as the class of languages decided by polynomial-size circuits with constant-size segregators.

In [16] we observed that the two-person pebble game of Dymond and Tompa can be used to simulate circuits with small separator size in small depth, giving essentially the same dependence of the depth on the separator size as in the current paper. The approach in [16] based on the two person pebble game can also be extended to graphs with small segregators. However, the simulation based on the two person pebble game is non-uniform, and it seems that the resulting circuits cannot be produced efficiently using this approach. Jansen and Sarma's [21] simulation of bounded treewidth circuits is also non-uniform.

For circuits with constant-size segregators or separators, the simulating circuits we obtain in this paper can be generated in space $O(\log^2 s)$. We also note that our simulation works for any circuit, and if the circuit has a segregator of size $f(s)$, we obtain a simulating circuit of depth at most $O(f(s) \log s)$, the value $f(s)$ does not have to be provided in advance. In contrast, the simulation in [21] assumes that the treewidth k is known in advance, and a tree decomposition is available along with the description of the circuit to be simulated. It would be desirable to generate the simulating circuits even more efficiently with respect to space or circuit depth, especially in the case of polynomial-size circuits with constant-size segregators or separators, since in that case, as in the case of formulas in Spira's theorem, the resulting circuits are NC^1 circuits. Note however, that even in the case of formulas (tree-like circuits) Spira's theorem is non-uniform. It is not known if the restructuring procedure for formulas in Spira's theorem producing the simulating $O(\log s)$ depth circuits can be directly implemented in less than $O(\log^2 s)$ space, or less than $O(\log^2 s)$ depth [8, 9].

The question of finding a uniform version of Spira's theorem has direct relevance for the complexity of the Boolean Formula Value problem. While a logspace uniform version of Spira's restructuring algorithm is still not known, it is known that for Boolean formulas presented as parenthesized expressions the Boolean Formula Value problem is in $SPACE(\log n)$ [26], and in $DLOGTIME$-uniform NC^1 [8, 9]. However, when the Boolean formulas are presented as tree-like circuits, the best result so far shows only that the Boolean Formula Value Problem can be solved in $O(\log^2 n)$ space.

We also consider the space complexity of the Circuit Value Problem (CVP). Ladner [23] showed that the Circuit Value Problem is P-complete. The space complexity of the CVP is not known to be $o(n/\log n)$ for general Boolean circuits. There are only a few restricted versions of the Circuit Value Problem that have been previously shown to have small-space complexity. It is a straightforward consequence of Borodin's theorem [5] (see Theorem C) that the CVP for logspace uniform depth d circuits is in $SPACE(d)$ for $d \geq \log n$. It is also easy to see that the CVP for small-width circuits can be solved in small space. The Monotone Planar Circuit Value Problem (MPCVP) is another restricted version of CVP with small-space complexity, where the circuits only have $\wedge$ and $\vee$ gates, positive input literals,

and can be embedded on the plane without crossings. There are numerous results on this problem. The strongest result so far is by Limaye, Mahajan, and Sarma [24], who showed that MPCVP is in SAC^2. Also see [11, 40, 30] for results on MPCVP in the PRAM model. Stronger bounds are known for some restricted versions of MPCVP [10, 19, 13, 3, 22, 31, 40].

Our generalization of Spira's theorem allows us to bound the space complexity of the Circuit Value Problem for circuits with small separators and segregators. We show that the Boolean Circuit Value Problem for circuits with constant-size segregators (or separators) is in deterministic $SPACE(\log^2 n)$. Our results also imply that the Planar Circuit Value problem, which is known to be P-Complete [19], is in $SPACE(\sqrt{n}\log n)$.

In addition we show that the Layered Circuit Value and the Synchronous Circuit Value problems, which are both P-complete [20], are in $SPACE(\sqrt{n})$. However, since layered circuits and synchronous circuits do not necessarily have small separators or segregators, instead of using our generalization of Spira's theorem we use a different approach.

2 Preliminaries

2.1 Space Bounded Turing Machines

For the space complexity of Turing machines, we follow the convention of considering Turing machines with a separate *read-only* input tape, and additional work tapes. If the machine has to produce an output string (instead of just accepting or rejecting its input), then we also assume a separate *write-only* output tape. The space used by a Turing machine on a given input is defined as the number of work tape cells visited during the computation over all work tapes. The input tape and the output tape do not contribute to the space bound of the computation. This allows us to consider computations with sublinear space.

$SPACE(s(n))$ denotes the class of languages decidable by deterministic Turing machines with a separate read-only input tape and a separate write-only output tape using $O(s(n))$ space on the work tapes.

In the following, whenever we talk about space bounds of Turing Machines, it is assumed that the input tape is read-only, the output tape is write-only and the space bound refers to the space used on the work tapes. See Papadimitriou [27] for more details on space bounded Turing machines.

2.2 The Circuit Model

A Boolean circuit is a labeled directed acyclic graph (DAG), where every node is labeled by either a variable from $\{x_1, \ldots, x_n\}$, or a Boolean operation. The set of available operations we are allowed to use is called the *basis* of the circuit. A given basis B is called *complete* if any Boolean function can be computed by a circuit using only operations from B. The

inputs of a Boolean circuit are the nodes with in-degree (fan-in) zero, and the outputs of a Boolean circuit are the nodes with out-degree (fan-out) zero. We refer to the nodes with non-zero in-degree as *gates*. A formula (or tree-like circuit) is a circuit whose fan-out is one for every gate except the output. The size of a Boolean circuit is the number of its gates. We will typically consider Boolean circuits with gates of fan-in at most 2 from the basis $\{\wedge, \vee, \neg\}$ (unless stated otherwise). The *depth of a gate* g is the length of the longest path from any input to g. The *depth of a circuit* C is the depth of the output gate. See [39] for more on Boolean circuits.

There are several common ways to define the description of a circuit. To be specific, we use the following definition.

Definition 2.1. The description of a Boolean circuit is a sequence of circuit inputs $x_1, \ldots, x_n$ and the following quadruples:

$$\langle name, type, child1, child2 \rangle,$$

where *name* is the name of a gate, *type* is one of $\wedge$, $\vee$, or $\neg$, and *child*1 and *child*2 are the inputs to the gate. *child*2 can be empty for gates of type $\neg$.

Note that *child*1 and *child*2 can be either gates or circuit inputs.

Definition 2.2. A family of Boolean circuits $\{C_n\}$ is called $h(n)$*-space uniform*, if there exists a deterministic Turing machine M that on input 1^n, outputs the description of C_n using space $O(h(n))$ for all n. In particular, $\{C_n\}$ is *logspace uniform* if $h(n) = \log n$.

2.3 Separators and Segregators

Informally, a node separator of a graph G is a set of nodes whose removal yields two disjoint subgraphs of G. In this paper we only consider *balanced* separators, that yield subgraphs that are comparable in size. In the next definition each of the two subDAGs could consist of several weakly connected components.

Definition 2.3. A *separator of size* k of a DAG $G = (V, E)$ is a set of k nodes $S \subseteq V$ such that $G \setminus S$ is not weakly connected (i.e. the underlying undirected graph is not connected); and the removal of S partitions $G \setminus S$ into two subDAGs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, such that $|V_i| \leq \frac{2}{3}|V|$ for $i = 1, 2$, and there are no edges either from G_1 to G_2, or from G_2 to G_1 in $G \setminus S$.

Segregators are a relaxation of separators in directed acyclic graphs [29, 33].

Definition 2.4. A *segregator of size* k of a DAG $G = (V, E)$ is a set of k nodes $S \subseteq V$ such that every node in $G \setminus S$ has at most $\frac{2}{3}|V|$ predecessors in $G \setminus S$.

The following lemma follows directly from the definitions.

Lemma 1. *Any DAG with a separator of size k has a segregator of size k.*

Notice that the reverse is not true in general, since a node in a DAG may have much smaller number of predecessors than the size of the component that contains the node in the underlying undirected graph.

3 Boolean Circuits with Small Segregators or Separators

Definition 3.1. We say that a Boolean circuit C has separators of size $f()$ if the underlying DAG of every subcircuit of C with s gates has a separator of size at most $f(s)$.

We say that a Boolean circuit C has segregators of size $f()$ if the underlying DAG of every subcircuit of C with s gates has a segregator of size at most $f(s)$.

The above definition is reasonable, since we typically consider classes of circuits based on properties of their underlying DAGs that are closed with respect to subDAGs, for example planar circuits, circuits with small treewidth, etc.

We talk about constant-size separators (resp. segregators), if the size of the separator (resp. segregator) is bounded by a fixed constant that does not depend on the size of the circuit.

By Lemma 1, if the circuit has separators of size $f()$, then it must also have segregators of size $f()$. Therefore in the following we will focus on circuits with small segregators.

We state and prove the following generalization of Spira's theorem for the basis $\{\wedge, \vee, \neg\}$ with fan-in at most 2. It is easy to see that Theorem 1 can be generalized to Boolean circuits over arbitrary complete basis with bounded fan-in, since any complete basis can implement the selector used in expression (1).

Theorem 1. *Any Boolean circuit of size s with segregators of size $f()$ can be simulated in depth $O(f(s))$ if $f(s) = \Omega(s^{\varepsilon})$ for some constant $\varepsilon > 0$, and in depth $O(f(s) \log s)$ otherwise.*

Proof. The construction is defined recursively. Let $U = \{u_1, \ldots, u_p\}$ be the segregator of C with size $p \leq f(s)$. Let $C_1, \ldots, C_p$ be the subcircuits of C corresponding to the nodes of the segregator, that is the node u_j is the output of the subcircuit C_j, for $j = 1, \ldots, p$. Let g_j be the Boolean function computed by C_j. Let v be the output node of the circuit C, and let $\hat{C}$ be the circuit with output node v, obtained from C by replacing the nodes in U by new variables $y_1, \ldots, y_p$. Thus, if the original circuit C has n variables, then $\hat{C}$ may have up to $p + n$ variables. It is possible that $\hat{C}$ has less than $p + n$ variables, if some of the original

inputs get disconnected from the output v after removing the nodes of the segregator from the circuit.

We enumerate all Boolean vectors $c \in \{0,1\}^p$. Let $c_i = \langle c_{i,1}, c_{i,2}, \ldots, c_{i,p} \rangle$ be the ith Boolean vector of length p, for $i = 1, \ldots, 2^p$, according to some fixed ordering. Let $\hat{C}_i$ be the circuit obtained from $\hat{C}$ by fixing the values of the variables $y_1, \ldots, y_p$ to the bits $c_{i,1}, \ldots, c_{i,p}$, respectively. Let $h_i : \{0,1\}^n \to \{0,1\}$ be the Boolean function computed by the circuit $\hat{C}_i$.

Then, the Boolean function computed by the circuit C can be represented using the following expression:

$$\bigvee_{i=1}^{2^p} \left(h_i \wedge \bigwedge_{j=1}^{p} ((g_j \wedge c_{i,j}) \vee (\neg g_j \wedge \neg c_{i,j})) \right) \tag{1}$$

To see that expression (1) is indeed the function computed by C, note that on each input x, exactly one of the functions H_i evaluates to 1, where H_i is defined by

$$H_i = \bigwedge_{j=1}^{p} ((g_j \wedge c_{i,j}) \vee (\neg g_j \wedge \neg c_{i,j})) .$$

For a given input $x \in \{0,1\}^n$, let $v_x = \langle g_1(x), g_2(x), \ldots, g_p(x) \rangle \in \{0,1\}^p$. Notice that $H_i(x)$ is nonzero if and only if $v_x = c_i$. Finally, note that $C(x) = h_i(x)$ when $v_x = c_i$.

Next we will represent the functions h_i for $i = 1, \ldots, 2^p$ and g_j for $j = 1, \ldots, p$. We could proceed with a straightforward recursion, if we could claim that each subcircuit $C_1, \ldots, C_p$ and each circuit $\hat{C}_i$ for $i = 1, \ldots, 2^p$ has size at most $2s/3$. In fact, we do know that every subDAG of the underlying DAG of C with the nodes of U removed has size at most $2s/3$. However, the output node of the subcircuit C_j is u_j, and u_j is a member of the segregator U. Note that the underlying DAGs of the circuits $\hat{C}_i$ are identical (they only differ from each other in the substituted constants), and their output node v is the output node of the "original" circuit C. The node v may or may not participate in the segregator. If the node v participates in the segregator, then the functions h_i are constants and the recursion stops.

We can compute the function g_j (computed at gate u_j) by an additional gate if we compute the functions computed at the two children of the gate u_j. If none of the children participates in the segregator, then we know that their subcircuits must have size at most $2s/3$. However, it is possible that children of segregator nodes are also included in the segregator. Let S_j be the set of nodes in the segregator, that are predecessors of u_j, such that there is a path from each of them to u_j that consists only of segregator nodes. We also include u_j in S_j. That is, S_j forms a subcircuit with output u_j that consists of segregator nodes. Let B_j be the "boundary" of S_j formed by nodes that are not in the segregator, that is, B_j contains the children of the nodes in S_j that are not included in the segregator. Then we can compute the function g_j from the functions computed at the nodes in B_j (these can be

computed by subcircuits of size at most $2s/3$) with an additional set of gates corresponding to the segregator nodes in S_j. Since $|S_j| \leq p$, this takes additional depth at most p.

To summarize, we can compute the functions h_i and g_j, by first computing in parallel the functions corresponding to all subcircuits after removing the nodes of the segregator. We know that each such subcircuit has size at most $2s/3$, and we can use our construction recursively on these smaller size circuits. Then we finish computing every function h_i and g_j we need, by adding the gates corresponding to the nodes participating in the segregator. This will take at most an additional $p \leq f(s)$ depth. Then we can compute the function computed by C by expression (1). This takes at most an additional $p + \lceil \log(p+1) \rceil + 3 = O(f(s))$ depth. Thus, in each iteration, we increase the depth by at most $O(f(s))$. Since the size is reduced by a constant factor in each iteration, we are done after $O(\log s)$ steps. More precisely, the depth of the final circuit is $O\left(\sum_{i=0}^{\lceil \log_{3/2} s \rceil} f\left((2/3)^i s\right)\right)$. Thus the depth of the final circuit is $O(f(s))$ if $f(s) = s^\epsilon$ for some constant $\epsilon > 0$, or $O(f(s) \log s)$ otherwise. □

Theorem 2. *The class of languages decided by non-uniform families of polynomial-size circuits with constant-size segregators equals non-uniform NC^1.*

Proof. Immediately follows from Theorem 1. □

Robertson and Seymour [32] showed that if a graph has treewidth k, then the graph also has separator size $O(k)$. Together with Lemma 1 and Theorem 2, a polynomial-size circuit with treewidth k can be simulated in depth $O(k \log n)$. This improves a result in [21], which showed that Boolean circuits of size $n^{O(1)}$ and treewidth k can be simulated in non-uniform depth $O(k^2 \log n)$. We refer interested readers to [12] and [15] for more background on treewidth.

3.1 Monotone Circuits

In this section we consider monotone circuits, i.e. circuits over the basis $\{\wedge, \vee\}$ with fan-in 2. As mentioned in the introduction, Wegener [38] proved Theorem A for monotone Boolean formulas. The following theorem generalizes his result to monotone circuits with small segregators.

Theorem 3. *Any monotone Boolean circuit C of size s with segregators of size $f()$ can be simulated by a monotone Boolean circuit in depth $O(f(s))$ if $f(s) = \Omega(s^\varepsilon)$ for some constant $\varepsilon > 0$, and in depth $O(f(s) \log s)$ otherwise.*

Proof. We first give a monotone version of expression (1), and the rest of the proof follows from the proof for Theorem 1. As in the previous section, we enumerate all Boolean vectors $c \in \{0,1\}^p$. Let $c_i = \langle c_{i,1}, c_{i,2}, \ldots, c_{i,p} \rangle$ be the ith Boolean vector of length p, for $i = 1, \ldots, 2^p$, according to some fixed ordering. We assume that the c_1 is the all-zero vector. Let $\hat{C}_i$

be the circuit obtained from $\hat{C}$ by fixing the values of the variables $y_1, \ldots, y_p$ to the bits $c_{i,1}, \ldots, c_{i,p}$, respectively, where $\hat{C}$ and $y_1, \ldots, y_p$ are defined as in the proof of Theorem 1. Let $h_i : \{0,1\}^n \to \{0,1\}$ be the Boolean function computed by the circuit $\hat{C}_i$. Let G_i be defined as follows for $i \geq 2$:

$$G_i = \bigwedge_{\substack{1 \leq j \leq p \\ c_{i,j}=1}} g_j.$$

We claim that the function computed by C can be represented by the following expression:

$$h_1 \vee \bigvee_{i=2}^{2^p} (h_i \wedge G_i) = h_1 \vee \bigvee_{i=2}^{2^p} \left(h_i \wedge \bigwedge_{\substack{1 \leq j \leq p \\ c_{i,j}=1}} g_j \right). \tag{2}$$

To prove this claim, first we define a relation $<$ on Boolean vectors of length p: given $c_u = \langle c_{u,1}, c_{u,2}, \ldots, c_{u,p} \rangle$ and $c_v = \langle c_{v,1}, c_{v,2}, \ldots, c_{v,p} \rangle$, $c_u < c_v$ iff $c_{u,k} \leq c_{v,k}$ for all $1 \leq k \leq p$, and $c_{u,r} < c_{v,r}$ for some $1 \leq r \leq p$.

Let $w_x = \langle g_1(x), g_2(x), \ldots, g_p(x) \rangle \in \{0,1\}^p$. Notice that if $w_x = c_i$, then $G_i(x) = 1$. On the other hand, for any vector c_k such that $w_x < c_k$ or w_x is not comparable to c_k, $G_k(x) = 0$. This can be proved by the following argument. If $c_i < c_k$, then there exists an l such that $c_{i,l} < c_{k,l}$. Then $g_l(x) = c_{i,l} = 0$, which implies that $G_k(x) = 0$. If c_i and c_k are not comparable, then there must also exist an l such that $c_{i,l} = 0$ but $c_{k,l} = 1$. Thus $g_l(x) = 0$ and $G_k(x) = 0$. This implies that on a given $x \in \{0,1\}^n$, expression (2) evaluates to

$$h_1(x) \vee \bigvee_{\substack{2 \leq k \leq 2^p \\ c_k < w_x}} (h_k(x) \wedge G_k(x)).$$

Since C is monotone, $c_k < c_i$ implies $h_k(z) \leq h_i(z)$ for all $z \in \{0,1\}^n$. Note also that since $w_x = c_i$, $G_k(x) \leq G_i(x) = 1$ for any k. Therefore we have

$$\begin{aligned} & h_1(x) \vee \bigvee_{\substack{2 \leq k \leq 2^p \\ c_k < c_i}} (h_k(x) \wedge G_k(x)) \\ = \; & h_1(x) \vee (h_i(x) \wedge G_i(x)) \\ = \; & h_1(x) \vee h_i(x) \\ = \; & h_i(x) \text{ since } c_1 < c_i \end{aligned}$$

Recall that $C(x) = h_i(x)$ when $w_x = c_i$.

□

4 Finding minimum size segregators in small space

4.1 Segregators of directed acyclic graphs

In this section, we give a space-efficient algorithm to find a minimum size segregator in arbitrary directed acyclic graphs.

We will use the following space-efficient algorithm for reachability in directed graphs by Savitch [35], to count the number of predecessors of a given node.

Theorem B. *[35] Given a directed graph G on s nodes and two nodes $u, v \in G$, there exists a deterministic Turing machine that decides if there is a path from u to v in G using space $O(\log^2 s)$.*

Lemma 2. *Let G be a DAG with s nodes. There exists a deterministic Turing machine M such that, on input G, if G has a segregator of size $f(s)$, then M outputs a segregator of G of size at most $f(s)$ using space $O(f(s) \log s + \log^2 s)$.*

Proof. We first define a Turing machine M_1 that takes G and a node $v \in G$ as input, and computes the number of predecessors of v in G, i.e. the number of nodes u such that there exists a directed path from u to v in G. In the beginning M_1 initializes a counter to 1. Then M_1 uses Theorem B to check, one-by-one, for each node $u \in G \setminus \{v\}$ if there is a directed path from u to v in G. For each node $u \in G \setminus \{v\}$ such that v is reachable from u, the counter is incremented. The space used to check the reachability of v from u is reused when checking for reachability from the next node in $G \setminus \{v\}$. Thus M_1 uses $O(\log^2 s)$ space and computes the size of the subDAG with v as the root.

We now define M in Lemma 2 as follows. First M enumerates integers k such that $1 \leq k \leq s$ in increasing order. For a fixed k, M enumerates subsets W of size k of the nodes in G in lexicographic order. For a given W, for every node $u \in G \setminus W$, let $G(u)$ denote the set of predecessors of u in $G \setminus W$. That is, $G(u)$ is the subDAG in $G \setminus W$ with u as its root. M uses M_1 to compute $|G(u)|$. If there exists one node $u \in G \setminus W$ such that $|G(u)| > \frac{2}{3}s$, then M continues to the next W, or the next k if every W of the current size has been already checked. Also, every time before continuing to the next W or the next k, M clears unnecessary information from the work tape.

We now argue that M will find a segregator of the smallest size. Observe that the set of nodes of G is a segregator of size s, so M is guaranteed to find a segregator. Since we try every k in increasing order, and we check for every subset W of size k whether or not it is a segregator, it is guaranteed that we will find a segregator of the smallest possible size in G.

We now argue that M only uses $O(f(s) \log s + \log^2 s)$ space. The description of G can be read using a counter of size $O(\log s)$. The enumeration and the storing of W both take $O(k \log s) = O(f(s) \log s)$ space. The computation of $|G(u)|$ takes $O(\log^2 s)$ space since M_1 uses $O(\log^2)$ space. Thus the space complexity to find a segregator of smallest size is $O(f(s) \log s + \log^2 s)$. □

Note that in the proof for Lemma 2, the input of M consists of only the description of the graph. M does not know the value of $f(s)$ in advance. Also, by Lemma 1, for graphs with separators of size k, the algorithm in Lemma 2 will also find a segregator of size at most k.

4.2 Segregators of uniform circuits

Intuitively, Lemma 2 seems to apply directly to circuits since circuits are also DAGs. However, the input of the Turing machine that has to generate the circuit C_n for a uniform family of circuits, is the unary representation of n (1^n), so the graph of the circuit C_n is not available directly. Since we want to generate the segregator using small space, we cannot store the description of C_n on the work tapes. As it is standard in such situations, we will generate the description of C_n as needed for the machine in the proof of Lemma 2, but never store the complete description. We then have the following lemma.

Lemma 3. *Let $\mathcal{C}$ be a $h(n)$-space uniform family of circuits. Let $C_n \in \mathcal{C}$ be the Boolean circuit in the family with n inputs, and assume that C_n has size $s = s(n)$ and a segregator of size $f(s)$. Then there exists a deterministic Turing machine $\hat{M}$ that on input 1^n, outputs a segregator of C_n of size at most $f(s)$ using space $O(h(n) + f(s) \log s + \log^2 s)$.*

As in the case for directed graphs, for circuits with separators of size $f(s)$, the algorithm in Lemma 3 will also find a segregator of size at most $f(s)$.

5 Generating the simulating circuits in small space

Let v be any node and Z be any set of nodes in the underlying graph of a circuit C_n. We denote by $C_{v,Z}$ the circuit obtained from the subcircuit C_v of C_n with output v by replacing every node in Z that participates in C_v by a new input variable.

Lemma 4. *Let $\mathcal{C}$ be a $h(n)$-space uniform family of circuits. Let $C_n \in \mathcal{C}$ be the circuit with n inputs in the family, and assume that C_n has size $s = s(n)$. Let v be any node and Z be any set of nodes in the underlying graph of C_n. Then there exists a Turing machine M_2 such that on input 1^n, v and Z, M_2 outputs the description of the circuit $C_{v,Z}$. Furthermore, M_2 runs in space $O(h(n) + \log^2 s)$.*

Note that if $Z = \emptyset$, or if Z does not contain any predecessors of v then $C_{v,Z}$ is simply the subcircuit C_v. Similarly to the circuit $\hat{C}$ in the proof of Theorem 1, if the size of Z is r, and C_v depends on n' input variables, then $C_{v,Z}$ may have up to $n' + r$ variables. If $v \in Z$, then $C_{v,Z}$ is simply a new variable.

Proof. Let M_1 be the Turing machine that on input 1^n generates the description of C_n using space $O(h(n))$. M_2 will use M_1 to generate information about the circuit C_n as needed. As before, the full description of C_n will never be stored. M_2 will use Theorem B to check if a given node is part of the subcircuit C_v, using space $O(\log^2 s)$. As before, space can be reused when checking for a new node. Note that the size of the set Z can be larger than the size of the subcircuit C_v, but it will be at most s, which is the size of the whole circuit C_n. M_2 will need to work with a counter of size $\log |Z|$, but $\log |Z| < \log^2 s$. □

Lemma 5. *Let $\mathcal{C}$ be a $h(n)$-space uniform family of circuits. Let $C_n \in \mathcal{C}$ be the circuit with n inputs in the family, and assume that C_n has size $s = s(n)$. Let v be any node and Z be any set of nodes in the underlying graph of C_n. Also assume that C_n has segregators of size $f()$. Then there exists a Turing machine M_3 such that on input 1^n, v and Z, M_3 outputs a minimum size segregator of $C_{v,Z}$ using space $O(h(n) + f(s) \log s + \log^2 s)$.*

Proof. Let M_2 be the Turing machine in Lemma 4 that generates the description of $C_{v,Z}$ in space $O(h(n) + \log^2 s)$. Let M be the Turing machine in the statement of Lemma 2, that takes a directed graph G as input, and outputs a minimum size segregator of G. The machine M_3 will simulate M on the underlying directed graph of $C_{v,Z}$. However, as before, the full description of the graph will never be stored. Instead, whenever M_3 needs some information about the graph, it lets M_2 run, (without recording its output), until the required information is generated. The size of the subcircuit $C_{v,Z}$ is $s' \leq s$. Since C_n has segregators of size $f()$, we know that $C_{v,Z}$ has a segregator of size $f(s')$. Recall that M always finds a minimum size segregator, thus it will find a segregator of size $f(s') \leq f(s)$. Since M runs in space $O(f(s) \log s + \log^2 s)$, the total space used will be $O(h(n) + f(s) \log s + \log^2 s)$. □

Now we are ready to prove a uniform version of Theorem 1.

Theorem 4. *Let $\mathcal{C}$ be an $h(n)$-space uniform family of Boolean circuits. Let $C_n \in \mathcal{C}$ be the Boolean circuit on n inputs with size $s = s(n)$. Suppose that C_n has segregators of size $f()$. Let $g(s) = f(s)$ if $f(s) = \Omega(s^c)$ for some constant $c > 0$ and $f(s) \log s$ otherwise. Then $\mathcal{C}$ can be simulated by a $O(h(n) + g(s) \log s)$-space uniform family of Boolean circuits of depth $O(g(s))$.*

Proof. We show that the construction in the proof of Theorem 1 can be generated by a machine M^* within the appropriate space bounds. M^* on input 1^n will output the description of the depth $O(g(s))$ circuit simulating the circuit $C_n \in \mathcal{C}$.

M^* generates the simulating circuit essentially as described in the proof of Theorem 1. In each step of the recursion, M^* has to do the following:

1. Find a segregator S of the current subcircuit, and store the list of nodes of S in workspace.

2. Find and store the list of nodes that participate in $B = \cup_{j=1}^{|S|} B_j$. Note that a given node may belong to B_j for more than one j, but $|\cup_{j=1}^{|S|} B_j| \leq 2|S|$, since B_j contains only children of segregator nodes. Thus, if $|S| = p$, it takes $O(p \log s)$ space to store the list of nodes in B. We can generate this list using $\hat{M}_1$, where $\hat{M}_1$ is the Turing machine that on input 1^n generates the description of C_n using space $O(h(n))$. We will run $\hat{M}_1$ several times, reusing space, and never store the full description of the circuit, as discussed before. For finding the set B_j, we have to find the set S_j and store it until we are finished generating B_j. For each j this takes $O(p \log s)$ workspace. We reuse this space when we move on to the next j. For each node of B_j that we find, we check if we have already added it to the list, so the full list B takes at most $O(p \log s)$ workspace to store.

3. Output the description of the part of the circuit that corresponds to the current subcircuit. This is based on the expression (1), and the sets B_j and S_j. We produce the description of the part of the circuit to compute g_j, while we have B_j and S_j stored in memory. We reuse space when we move on to the next j. Recall that the output is not part of the space bound. (We do keep S and the full list B until the end of processing the subcircuit, and maybe longer as we see below.)

The recursion will continue to process the subcircuits $\hat{C}_i$ (functions h_i) defined in the proof of Theorem 1, and the subcircuits of the nodes in B. Recall that each of these subcircuits has size at most 2/3 of the last subcircuit. The recursion stops when a subcircuit is either constant or an input variable. We need a counter of size p to enumerate the Boolean vectors substituted, and to enumerate the functions h_i, for $i = 1, \ldots, 2^p$.

We reuse space as we proceed to the next recursive step. However, to be able to proceed with the recursion, we need to retain some information about the segregators S, the sets B and list of values substituted for segregator nodes from previous recursive steps to be able to generate and process the current subcircuits. We process the subcircuits similarly to a depth first search in the recursion tree, starting with the subcircuits corresponding to the set B and leaving the subcircuit for the functions h_i for last. Recall that there is only one subcircuit to consider for the functions h_i, they just differ in the values of constants substituted.

We keep S, B and list of values substituted for nodes in S from previous steps along the current path in the recursion tree. Since there are $\log s$ stages of the recursion, at any point we keep at most $\log s$ segregators with their corresponding set B and list of values. This takes $O(\sum_{i=1}^{\log s} f(s/2^i) \log s) = O(g(s) \log s)$ space.

At the first iteration, we simply use the machine $\hat{M}$ from Lemma 3 to find a segregator. Now we describe how to find a segregator of the current subcircuit during the recursion. To find a segregator for the subcircuits with outputs in the sets B described above, we use M_3 with input 1^n, u where u is the output of the subcircuit, and $Z = \emptyset$. (For processing the subcircuits corresponding to nodes in the sets B we do not need to worry about the

segregators that we stored from previous levels of the recursion.) For the subcircuits $\hat{C}_i$ (functions h_i) we use M_3 with input 1^n, v, where v is the output node of the subcircuits $\hat{C}_i$ (recall that they have the same output node, they only differ in the constants substituted), and Z where Z is the union of all the segregators currently stored.

In each step of the recursion, M_3 finds the current segregator in at most $h(n)+O(\log^2 s+ f(s)\log s)$ space by Lemma 5. Note that after each invocation of Lemma 5, its workspace can be reused.

Thus on input 1^n, the space used to construct the new circuit is at most $O(h(n)+\log^2 s+ g(s)\log s) = O(h(n) + g(s)\log s)$ since $g(s) = \Omega(\log s)$. □

6 Circuit Value Problem

The Boolean Circuit Value problem is defined as follows: given the description of a circuit C and an assignment x to the variables of C, compute the value of the output of the circuit C evaluated on the assignment x. As mentioned in the introduction, it is not known if the general Circuit Value Problem, which is P-complete, can be solved in $o(n/\log n)$ space. In this section, we consider the Circuit Value Problem for three circuit families: planar, layered, and synchronous circuits. It is known that these variants of the Circuit Value Problem are all P-complete. See [19] and [20]. In the following, we first solve the Planar Circuit Value Problem in $O(\sqrt{n}\log n)$ space using Theorem 4. Then we show that the Layered Circuit Value Problem and the Synchronous Circuit Value Problem can be solved in $O(\sqrt{n})$ space.

6.1 Planar Circuits

As an application of Theorem 4, we obtain a bound on the space complexity of the Circuit Value Problem for Boolean circuits with small segregators (or separators). We need the following theorem of Borodin [5].

Theorem C. *[5] Any language decided by a $h(n)$-space uniform circuit family of depth $h(n) \geq \log n$, can be decided by a Turing machine in space $O(h(n))$.*

Theorem 5. *The Boolean Circuit Value problem for circuits that have size s and segregators (or separators) of size $f(s)$ is in $SPACE(f(s)\log s)$ if $f(s) = \Omega(s^\varepsilon)$ for some constant $\varepsilon > 0$, and $SPACE(f(s)\log^2 s)$ otherwise.*

Proof. Let $g(s) = f(s)$ if $f(s) = \Omega(s^\varepsilon)$ for some constant $\varepsilon > 0$, and $g(s) = f(s)\log s$ otherwise. Since the description of C is given in the input, by the proof of Theorem 4, using $O(g(s)\log s)$ space, we can generate a circuit C' of depth $O(g(s))$ that simulates C. Then we can evaluate C' in the given assignment using the argument of Theorem C using space $O(g(s))$. □

Theorem 5 immediately implies the following theorem.

Theorem 6. *The Boolean Circuit Value problem for circuits with constant-size segregators (or separators) is in* $SPACE(\log^2 n)$.

Lipton and Tarjan [25] gave the following "planar separator theorem".

Theorem D. *[25] Any planar graph of size* s *has a separator of size* $O(\sqrt{s})$.

We use this to obtain our result about the space complexity of the Circuit Value Problem for planar graphs.

Theorem 7. *The Planar Circuit Value Problem is in* $SPACE(\sqrt{n}\log n)$.

Proof. Immediately follows from Theorem D and Theorem 5. □

6.2 Layered Circuits and Synchronous Circuits

In the following we consider the Layered Circuit Value Problem and the Synchronous Circuit Value Problem. We first show that the Layered Circuit Value Problem can be solved in $O(\sqrt{s})$ space. Since every synchronous circuit is layered, it then follows that the Synchronous Circuit Value Problem can be also solved in $O(\sqrt{s})$ space. Before stating and proving our results, we need a few definitions first.

Definition 6.1. A circuit is *layered*, if its set of gates can be partitioned into subsets called *layers*, such that every wire connecting two gates in the circuit is between adjacent layers. For circuits with one output, the following is an equivalent definition: A circuit with one output is *layered* if for any gate g all paths from g to the output have the same length.

A circuit is *synchronous* if for any gate g, all paths from the inputs to g have the same length.

It is easy to show that any synchronous circuit is also layered, but the converse is not true. See Gál and Jang [16].

Definition 6.2. Let C be a circuit with one output, and let g be any gate in C. Then the *height of* g is the length of the longest path from g to the output. The *height of* C is the maximum height of all gates in C.

Given a layered circuit with one output, the *ith layer* of the circuit consists of all gates with height equal to i. Note that the 0th layer consists of the output gate.

In the followings we assume that a layered circuit has one output unless stated otherwise.

Lemma 6. *Given the description of a layered Boolean circuit* C *of size* s*, and any gate* g *in* C*, the height of* g *can be computed in* $O(\log s)$ *space.*

Proof. We define the following Turing machine M. M follows a path starting from g until it reaches the output of C. This can be done by scanning the description of C to see which gates in C have g as a child. If the gate g has fan-out more than 1, M chooses arbitrarily which parent of g to visit next, say for example the first such gate in the description. Once a parent of g is found, say h, we let h be the current gate and repeat the above process. M also counts the number of edges in the path and outputs that number when the output gate is reached. It is easy to see that M computes the height of g, since all paths from the gate g to the output of C have the same length. M uses $O(\log s)$ space since it only needs to remember the name of the current gate along the path and a constant number of counters using $O(\log s)$ space for each. □

Lemma 7. *Let C be a layered Boolean circuit of size s and height h. Let $0 \leq i < j \leq h$ be two integers. Let g be any gate in the ith layer, and let $C(g, j)$ be the subcircuit of C whose output is g, and whose inputs are all the gates in the jth layer that are connected to g by a path to g, and those circuit inputs that have a parent gate in layers $i, \ldots, j-1$. Then given the description of C, g, and j, the description of $C(g, j)$ can be computed in $O(\log^2 s)$ space.*

Proof. We now define a Turing machine M^*. We are going to produce a description of $C(g, j)$ according to Definition 2.1. First we list those circuit inputs that are participating in the subcircuit. Then we list the names of those gates u in the jth layer that serve as inputs to $C(g, j)$. We use Lemma 6 to check if a given gate is in the jth layer. We can use Savitch's theorem (Theorem B) to test if a given circuit input or a given gate u is connected to g by a path.

Next for $k = j - 1$ to 0, M^* generates the quadruples of the gates in the kth layer. For a given k, M^* scans the description of C, and for each gate $v \in C$, first check if v is in the kth layer and then check if v is connected to g. If yes, then M^* writes the quadruple corresponding to gate v to the output tape. It is clear that M^* uses $O(\log^2 s)$ space. □

Notice that the above algorithm can be easily modified such that it outputs the description of the subcircuit in any pre-determined format of circuit description. Also observe that the space used is dominated by the space of testing directed connectivity.

Theorem 8. *Given a description of a layered Boolean circuit C of size s, and an input assignment x to C, there exists a Turing machine that evaluates C on x using $O(\sqrt{s})$ space. That is, the Layered Circuit Value Problem can be solved in $O(\sqrt{s})$ space.*

The idea is to evaluate the circuit layer-by-layer if the layers are small, and use Borodin's theorem (Theorem C) when the layers are large. Since there cannot be too many large layers, we can bound the space. We now give the complete proof.

Proof. Let $y > 1$ be some positive number. We call a layer in C *large* if the number of gates in the layer is greater than y, or *small* otherwise. We will determine the value of y later.

We define a Turing machine M as follows. In the beginning M computes the height of C. This can be easily done by applying Lemma 6. Given any $1 \leq j \leq h$, we can compute the number of gates in the jth layer also using Lemma 6.

Let h be the height of C. There are two phases. In the first phase we consider the hth layer. Note that the inputs of all gates in the hth layer are circuit inputs. If the hth layer is small, then M writes the values of the gates in the hth layer on the work tape. These values can be computed from the circuit description and x. If the hth layer is large, then M looks for the largest $i < h$ such that the ith layer is small. Note that this means the depth of every gate in the ith layer is at most s/y, since every layer between the ith layer and the hth layer is large. Similarly to the proof of Lemma 4, given the circuit description of C, we can produce the description of the subcircuit C_v for each gate v in the ith layer. Note however, we do not have enough space to store the circuit description of C_v. Instead we will produce the necessary information about each gate as needed. We use Theorem C to evaluate C_v, and we write the value of each gate in the ith layer on the work tape. Furthermore, M writes down values of gates according to their order in the circuit description of C. In this list, we do not store the names of the gates.

Let m be the largest m such that the mth layer is small. Then at the end of Phase 1, we have all the values of the gates in the mth layer written on the work tape.

Now consider Phase 2. At the beginning of Phase 2, let $j = m$. If $j = 0$, then we are done. Let the kth layer ($0 \leq k < j$) be the next small layer. That is, either $k = j - 1$, or all the layers between the kth and jth layers are large. For a given gate g in the kth layer, we will use Lemma 7 to generate the description of the circuit $C(g, j)$. Note that the inputs to $C(g, j)$ are either circuit inputs, or gates in the jth layer. Also at this point, we have the values of all gates in the jth layer written on the work tape. We need the following claim.

Claim. Given the name of a gate u in the jth layer, we can compute in $O(\log s)$ space the index t such that u is the tth gate within the jth layer, according to the order of the gates in the circuit description of C.

Proof for the claim We enumerate the gates of the jth layer as follows. We consider each gate g in C according to their order in the circuit description of C. For each gate g, we use Lemma 6 to compute its height. If the height of g is j, we increment a counter. We stop when we reach u in the circuit description. The index t of u within the jth layer is the current value of the counter plus 1. It follows from Lemma 6 that the space used is $O(\log s)$. □

Putting it all together, M can produce the values of the gates in the kth layer as follows: M scans the description of C. For each gate g of C, we check (as before) if g is in the kth

layer. If yes, we use Lemma 7 to produce the description of $C(g, j)$. Note that the depth of $C(g, j)$ is at most s/y (since all the layers between the kth layer and the jth layer are large). Then we use Theorem C to evaluate $C(g, j)$. However, we never actually store the description of $C(g, j)$. Instead, each time we need information about a given gate, we run the machine M^* from Lemma 7 (without actually recording its output) until we get the necessary information.

Recall that each input to $C(g, j)$ is either a circuit input (thus its value can be obtained from x) or a gate from the jth layer. Given the name of a gate u in the jth layer that is an input to $C(g, j)$, we can use the above claim to find its value among the values of the gates in the jth layer.

We keep the values of the jth layer on the work tape until we finish computing all the values in the kth layer. Then we let $j = k$ and we repeat the above process re-using space no longer needed.

M continues the above process until $j = 0$. Then M outputs the value of the output gate. The space used by M is bounded by $O(y + s/y + \log^2 s) = O(y + s/y)$. Let $y = \sqrt{s}$, then M uses $O(\sqrt{s})$ space. □

Since every synchronous circuit is layered, the Synchronous Circuit Value Problem can be solved using $O(\sqrt{s})$ space.

Acknowledgements

A preliminary version of this paper appeared in the proceedings of SOFSEM 2012 [17]. We thank the anonymous referees for helpful comments.

References

[1] N. Alon, P. Seymour and R. Thomas: A Separator Theorem for Graphs with an Excluded Minor and its Applications. *Proceedings of STOC*, pp. 293–299 (1990).

[2] D. Barrington: Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. *J. Comput. Syst. Sci*, 38 (1), pp. 150–164 (1989).

[3] D. Barrington, C. Lu, P. B. Miltersen and S. Skyum: On monotone planar circuits. *Proceedings of IEEE Conference on Computational Complexity*, pp. 24–33, (1999).

[4] M. Bonet and S. R. Buss: Size-depth tradeoffs for Boolean formulae. *Information Processing Letters*, 49(3), pp. 151–155 (1994).

[5] A. Borodin: On Relating Time and Space to Size and Depth. *SIAM Journal on Computing*, 6(4), pp. 733–744 (1977).

[6] R. P. Brent: The Parallel Evaluation of General Arithmetic Expressions. *Journal of the ACM*, 21(2), pp. 201–206 (1974).

[7] N. Bshouty, R. Cleve and W. Eberly: Size-Depth Tradeoffs for Algebraic Formulas. *SIAM Journal on Computing*, 24(4), pp. 682–705 (1995).

[8] S. R. Buss: The Boolean formula value problem is in ALOGTIME. *Proceedings of STOC*, pp. 123–131 (1987).

[9] S. Buss, S. Cook, A. Gupta, and V. Ramachandran: An Optimal Parallel Algorithm for Formula Evaluation. *SIAM Journal on Computing*, 21(4), pp. 755–780 (1992).

[10] T. Chakraborty and S. Datta: One-Input-Face MPCVP Is Hard for L, but in LogDCFL. *Proceedings of FSTTCS 2006, LNCS 4337*, pp. 57–68 (2006).

[11] A. L. Delcher and S. R. Kosaraju : An NC Algorithm for Evaluating Monotone Planar Circuits. *SIAM J. Comput*, 24(2), pp. 369–375 (1995).

[12] R. G. Downey and M. R. Fellows: Parameterized Complexity. Springer (1999).

[13] P. Dymond and S. Cook: Complexity Theory of Parallel Time and Hardware. *Information and Computation*, 80(3), pp. 205–226 (1989).

[14] P. Dymond and M. Tompa: Speedups of Deterministic Machines by Synchronous Parallel Machines. *J. Comp. and Sys. Sci.*, 30(2), pp. 149–161 (1985).

[15] J. Flum and M. Grohe: Parameterized Complexity Theory. Springer (2006).

[16] A. Gál and J. Jang: The size and depth of layered Boolean circuits. *Proceedings of LATIN 2010, LNCS 6034*, pp. 372–383 (2010).

[17] A. Gál and J. Jang: A Generalization of Spira's Theorem and Circuits with Small Segregators or Separators. *Proceedings of SOFSEM 2012, LNCS 7147*, pp. 264–276 (2012).

[18] J.R. Gilbert, J.P. Hutchinson, and R.E. Tarjan: A Separator Theorem for Graphs of Bounded Genus. *Journal of Algorithms 5(3)*, pp. 391–407 (1984).

[19] L. Goldschlager: The monotone and planar circuit value problem is complete for P. *SIGACT News*, pp. 25–27 (1977).

[20] R. Greenlaw and H. J. Hoover and W. L. Ruzzo: Limits to parallel computation: *P*-completeness theory. (1995).

[21] M. Jansen and J. Sarma M. N.: Balancing Bounded Treewidth Circuits. *5th International Computer Science Symposium in Russia*, pp. 228–239 (2010).

[22] S. R. Kosaraju: On The Parallel Evaluation of Classes of Circuits. *Proceedings of FSTTCS 1990, LNCS 472*, pp. 232–237 (1990).

[23] R. E. Ladner: The circuit value problem is log-space complete for P. *SIGACT News*, 6(2), pp. 18–20 (1975).

[24] N. Limaye, M. Mahajan, and J. Sarma: Upper bounds for monotone planar circuit value and variants. *Computational Complexity*, 18, pp. 377–412 (2009).

[25] R. Lipton and R.E. Tarjan: A Separator Theorem for Planar Graphs. *SIAM J. Appl. Math.*, 36, pp. 177–189 (1979).

[26] N. A. Lynch: Log space recognition and translation of parenthesis languages. *J. Assoc. Comput. Mach.*, 24, pp. 583–590 (1977).

[27] C. H. Papadimitriou: Computational complexity. Addison-Wesley (1994).

[28] M. S. Paterson and L. G. Valiant: Circuit Size is Nonlinear in Depth. *Theoretical Computer Science*, 2(3), pp. 397–400 (1976).

[29] W. J. Paul, N. Pippenger, E. Szemerédi, and W. T. Trotter: On determinism versus non-determinism and related problems. *Proceedings of FOCS*, pp. 429–438 (1983).

[30] V. Ramachandran and H. Yang: An Efficient Parallel Algorithm for the General Planar Monotone Circuit Value Problem. *SIAM Journal on Computing*, 25(2), pp. 312–339 (1996).

[31] V. Ramachandran and H. Yang: An Efficient Parallel Algorithm for the Layered Planar Monotone Circuit Value Problem. *Algorithmica*, 18(3), pp. 384–404 (1997).

[32] N. Robertson and P. D. Seymour: Graph Minors II. Algorithmic aspects of tree width. *Journal of Algorithms*, 7, pp. 309–322 (1986).

[33] R. Santhanam: On separators, segregators and time versus space. *Proceedings of the Sixteenth Annual Conference on Computational Complexity*, pp. 286–294 (2000).

[34] J. E. Savage: The Complexity of Computing. John Wiley & Sons (1976).

[35] W. J. Savitch: Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4(2), pp. 177–192 (1970).

[36] P. M. Spira: On time-hardware complexity tradeoffs for Boolean functions. *Proc. 4th Hawaii Symp. on System Sciences*, pp. 525–527 (1971).

[37] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff: Fast Parallel Computation of Polynomials Using Few Processors. *SIAM J. Comp.*, 12(4), pp. 641–644 (1983).

[38] I. Wegener: Relating monotone formula size and monotone depth of Boolean functions. *Information Processing Letters*, 16(1), pp. 41–42 (1983).

[39] I. Wegener: The Complexity of Boolean Functions. (1987).

[40] H. Yang: An NC algorithm for the general planar monotone circuit value problem. *SPDP*, IEEE Computer Society, pp. 196–203 (1991).

ECCC ISSN 1433-8092

http://eccc.hpi-web.de