The Size and Depth of Layered Boolean Circuits

Anna Gál* and Jing-Tang Jang**

Dept. of Computer Science, University of Texas at Austin,
Austin, TX 78712-1188, USA
{panni, keith}@cs.utexas.edu

Abstract. We consider the relationship between size and depth for layered Boolean circuits, synchronous circuits and planar circuits as well as classes of circuits with small separators. In particular, we show that every layered Boolean circuit of size s can be simulated by a layered Boolean circuit of depth $O(\sqrt{s \log s})$. For planar circuits and synchronous circuits of size s, we obtain simulations of depth $O(\sqrt{s})$. The best known result so far was by Paterson and Valiant [16], and Dymond and Tompa [6], which holds for general Boolean circuits and states that $D(f) = O(C(f)/\log C(f))$, where $C(f)$ and $D(f)$ are the minimum size and depth, respectively, of Boolean circuits computing f. The proof of our main result uses an adaptive strategy based on the two-person pebble game introduced by Dymond and Tompa [6]. Improving any of our results by polylog factors would immediately improve the bounds for general circuits.

Key words: Boolean circuits, circuit size, circuit depth, pebble games

1 Introduction

In this paper, we study the relationship between the size and depth of fan-in 2 Boolean circuits over the basis $\{\vee, \wedge, \neg\}$. Given a Boolean circuit C, the size of C is the number of gates in C, and the depth of C is the length of the longest path from any input to the output. We will use the following notation for complexity classes. $DTIME(t(n))$ and $SPACE(s(n))$ are the classes of languages decidable by deterministic multi-tape Turing machines in time $O(t(n))$ and space $O(s(n))$, respectively. Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, define $C(f)$ to be the smallest size of any circuit over $\{\vee, \wedge, \neg\}$ computing f, and define $D(f)$ to be the smallest depth of any circuit over $\{\vee, \wedge, \neg\}$ computing f. Note that $C(f)$ and $D(f)$ are not necessarily achieved by the same circuit. We also need the following conventions to compare computation times in uniform models (Turing machines) and in non-uniform models (Boolean circuits and PRAMs). Given $L \subseteq \{0,1\}^*$, let $L_n = L \cap \{0,1\}^n$. We will also view L_n as the

* Supported in part by NSF Grant CCF-0830756
** Supported in part by MCD fellowship from Dept. of Computer Science, University of Texas at Austin, and NSF Grant CCF-0830756

following Boolean function: $L_n : \{0,1\}^n \to \{0,1\}$ such that $L_n(x_1, \ldots, x_n) = 1$ iff $x_1 \ldots x_n \in L_n$. In other words, we will use the notation L_n to denote both the set L_n and its characteristic function. Now define $SIZE(t(n)) = \{L : C(L_n) = O(t(n))\}$ and $DEPTH(s(n)) = \{L : D(L_n) = O(s(n))\}$. We will also consider uniform versions of these classes, i.e. $logspace\text{-}uniform\text{-}SIZE(t(n))$ and $logspace\text{-}uniform\text{-}DEPTH(s(n))$.

Pippenger and Fischer [17] showed that for $t(n) \geq n$, $DTIME(t(n)) \subseteq logspace\text{-}uniform\text{-}SIZE(t(n)\log t(n))$. Thus, circuit size is related to sequential computation time. Furthermore, Borodin [4] showed that for $s(n) \geq \log n$, $logspace\text{-}uniform\text{-}DEPTH(s(n))$ is a subset of $SPACE(s(n))$, and $SPACE(s(n))$ is a subset of $logspace\text{-}uniform\text{-}DEPTH(s^2(n))$. Thus, circuit depth is closely related to sequential computation space.

For the PRAM model, define $P_{unit}(f)$ and $P_{log}(f)$ to be the minimum computation time to compute f in the unit-cost and log-cost PRAM models, respectively. For our purposes, it is sufficient to consider CRCW PRAMs. Define $PRAM_{unit}(t(n)) = \{L : P_{unit}(L_n) = O(t(n))\}$ and $PRAM_{log}(t(n)) = \{L : P_{log}(L_n) = O(t(n))\}$. Similarly to circuit classes, we will also consider uniform versions of PRAM classes.

There is a tight connection between circuit depth and PRAM time. Stockmeyer and Vishkin [21] showed that $PRAM_{log}(t(n)) \subseteq DEPTH(t(n)\log m(n))$ and $DEPTH(s(n)) \subseteq PRAM_{log}(s(n))$, where $m(n)$ is the maximum of $t(n)$, the number of processors, and the input length n. These results hold even for the unit-cost PRAM model as long as multiplication is not counted as a unit-cost instruction.

The above results show that the study of circuit size versus depth helps to investigate the relationship between sequential and parallel computation time, as well as time versus space in sequential computation. However, very little is known about the size versus depth question for general Boolean circuits. The best known result so far is the following theorem, which was first proved by Paterson and Valiant [16], and later proved by Dymond and Tompa [6] using another method.

Theorem A *[16, 6] Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, we have $D(f) = O(C(f)/\log C(f))$, or $SIZE(t(n)) \subseteq DEPTH(t(n)/\log t(n))$.*

On the other hand, it can be easily shown that $D(f) = \Omega(\log C(f))$. Theorem A leaves a huge gap ($\log C(f)$ versus $C(f)/\log C(f)$) for circuits of any size. McColl and Paterson [13] showed that every Boolean function depending on n variables has circuit depth at most $n+1$. There is an even stronger result by Gaskov [7] showing that circuit depth is at most $n - \log\log n + 2 + o(1)$. This gives a much stronger bound on depth than Theorem A for functions that require circuits of large size. In particular, for $f : \{0,1\}^n \to \{0,1\}$ such that $C(f)$ is exponential in n, [13] and [7] give essentially tight bounds on depth. However, for functions that can be computed by subexponential-size circuits, there is still a large gap. Note that Theorem A gives a stronger result than [13] and [7] only when $C(f) = o(n\log n)$. Improving Theorem A would yield improvements over [13] and [7] for larger $C(f)$ as well.

Because of the connections mentioned above, there are other important consequences if Theorem A can be improved. Hopcroft, Paul, and Valiant [10] proved the following analogous theorem about sequential time and space, and Adleman and Loui [1] later gave an alternative proof.

Theorem B *[10, 1]* $DTIME(t(n)) \subseteq SPACE(t(n)/\log t(n))$.

By the results of Pippenger and Fischer, and Borodin mentioned above, improving Theorem A by at least a polylog factor immediately improves Theorem B.

Dymond and Tompa [6] showed that $DTIME(t(n)) \subseteq PRAM_{unit}(\sqrt{t(n)})$ for the unit-cost PRAM model. (This also holds for logspace uniform unit-cost PRAM.) However, no such result is known for the log-cost PRAM model. Since $DEPTH(s(n)) \subseteq PRAM_{log}(s(n))$, improving Theorem A by at least a polylog factor will also imply non-trivial relationship between $DTIME$ and log-cost $PRAM$ computation time.

For general Boolean circuits, the simulating depth $O(t(n)/\log t(n))$ in Theorem A is very close to the circuit size. On the other extreme, consider tree-like circuits, where every gate has fan-out at most 1. Spira [20] showed that given any tree-like Boolean circuit C of size $t(n)$, we can always simulate C by another tree-like Boolean circuit of depth $O(\log t(n))$. Note that tree-like circuits are commonly referred to as *formulas* in circuit complexity. We will use the term tree-like circuits to avoid any ambiguity. It is unlikely that Spira's result holds for general Boolean circuits, since that would imply $P = NC_1$. Still, it is possible that Theorem A can be improved. We indeed achieve improved simulations for special classes of Boolean circuits.

1.1 Our results

We consider the size versus depth problem for special classes of Boolean circuits. As far as we know, previously no better bounds were known for these classes than what follows from the bounds for general circuits [16, 6]. We obtain significant improvements over these general bounds for layered circuits, synchronous circuits, and planar circuits as well as classes of circuits with small separators. Informally, a circuit is layered if its set of gates can be partitioned into subsets called layers, such that every wire in the circuit is between adjacent layers. A circuit is synchronous if for any gate g, every path from the inputs to g has the same length. Synchronous and planar circuits have been extensively studied before. Synchronous circuits were introduced by Harper [9]. Planar circuits were introduced by Lipton and Tarjan [12]. Layered circuits are a natural generalization of synchronous circuits, but as far as we know they have not been explicitly studied. Layered graphs have been studied by Paul, Tarjan, and Celoni [14] (they call these "level graphs" in their paper). Belaga [3] defined locally synchronous circuits, which is a subclass of layered circuits, with the extra condition that each input variable can appear at most once. The synchronous circuits form a proper subset of layered circuits. (See next section for more details.) Furthermore, Turán

[22] showed that there exists a function f_n such that any synchronous circuit for f_n has size $\Omega(n \log n)$, but there exists a layered circuit for f_n with size $O(n)$. See Belaga [3] for the same gap for functions with multiple outputs. This distinguishes synchronous circuits and layered circuits with respect to their computational powers. Notice that every Boolean function can be computed by circuits from each of the classes we consider. Furthermore, these classes of circuits are quite frequently used in various situations.

Our main result is for layered circuits.

Theorem 1. *Every layered Boolean circuit of size s can be simulated by a layered Boolean circuit of depth $O(\sqrt{s \log s})$ computing the same function.*

We obtain slightly better bounds for synchronous circuits and planar circuits.

Theorem 2. *Every synchronous Boolean circuit of size s can be simulated by a synchronous Boolean circuit of depth $O(\sqrt{s})$ computing the same function.*

A circuit is planar if its underlying graph can be embedded in the plane without crossings of the wires.

Theorem 3. *Every planar Boolean circuit of size s can be simulated by a planar Boolean circuit of depth $O(\sqrt{s})$ computing the same function.*

For planar circuits, we use the fact that every planar circuit of size s has a separator of size $O(\sqrt{s})$ [11]. Informally, the separator of a graph is a subset of the nodes whose removal yields two subgraphs of comparable sizes. This allows us to use a divide-and-conquer strategy. Graphs with small separators include trees, planar graphs [11], graphs with bounded genus [8], and graphs with excluded minors [2]. In fact, we can get similar results for arbitrary classes of circuits with small separators.

On the other hand, not all synchronous circuits and layered circuits have small separators. See [19] for many examples. So we need strategies other than the divide-and-conquer approach. Our idea is to consider *cuts*, which separate the graph into two subgraphs that are not necessarily comparable in size. For synchronous circuits, our technique is to find a relatively small cut such that the function can be computed by the composition of two circuits of small depths. This gives a simple proof for synchronous circuits, but the same method cannot be applied to the more general layered circuits. For layered circuits, we develop an adaptive strategy in the two-person pebble game, such that the sizes of the cuts are taken into account during the game. Note that both [16] and [6] use the notion of separators in their proofs. Our results for synchronous circuits and layered circuits show that the minimum circuit depth does not necessarily grow with the separator size of the minimum-size circuit.

Finally we note that an arbitrary circuit of size s can be converted to either a planar or a synchronous circuit of size $O(s^2)$ [24]. Thus improving our results by polylog factors for any of the classes we considered would also yield improvements over the best known bounds for general circuits.

2 Definitions and Backgrounds

2.1 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 an operation from $\{\wedge, \vee, \neg\}$. 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. The size of a Boolean circuit is the number of its gates. We will consider Boolean circuits with gates of fan-in at most 2 from the basis $\{\wedge, \vee, \neg\}$. We refer interested readers to [24] for more background on Boolean circuits.

A circuit is *planar* if we can find an embedding in the plane for the circuit such that no two edges cross [12].

Definition 1. *[9]* ***Synchronous circuits*** *A circuit is* synchronous *if for any gate g, all paths from the inputs to g have the same length.*

Definition 2. ***Layered circuits*** *A circuit is* layered*, if its set of gates can be partitioned into subsets called* layers*, such that every wire 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.*

Definition 3. ***Depth and height*** *Let C be a circuit, and let g be any gate in C. The* depth of g *is the length of the longest path from any input to g. The* depth of C *is the depth of the output gate.*

For circuits with one output, the height of g *is the length of the longest path from g to the output.*

Definition 4. ***Levels and layers*** *The i*th level *of a circuit consists of all gates with depth equal to i. For circuits with one output, the i*th layer *of the circuit consists of all gates with height equal to i.*

Note that the 0th layer in a circuit with one output consists of the output gate, and the 0th level in any circuit consists of the inputs. Also note that "levels" and "layers" are usually used interchangeably in the literature, and distinguishing them this way is just our terminology. The following lemma is straightforward from the definitions, and it shows that every synchronous circuit is layered. A simple example shows that the converse is not true: consider the circuit with inputs x_1, x_2, x_3, gates $g_1 = x_1 \wedge x_2$, $g_2 = g_1 \wedge x_3$, and g_2 being the output.

Lemma 1. *A circuit C is synchronous if and only if every wire in C is between adjacent levels. Thus, every synchronous circuit is also a layered circuit.*

2.2 The Two-Person Pebble Game

Several variants of pebble games have been invented to study questions related to the space requirements of computation, e.g. [15, 5]. See [18] for a survey. Here we focus on the two-person pebble game, which was defined by Dymond and Tompa [6]. The game is played on a DAG G. There are two players, the challenger and the pebbler. The challenger starts the game by challenging any single node of G, then the pebbler puts some pebbles on a subset of the nodes. From this point on, the challenger can only challenge a node that was either challenged or pebbled in the previous round. The game continues until at the beginning of the pebbler's move, all the predecessors of the currently challenged node w are already pebbled. Then we say the challenger *loses G at w*. If, under the best defense of the challenger, the pebbler can win with t number of pebble placements, then we say that G can be *two-person pebbled in time t*. Notice that the pebbler does not remove pebbles once a node is pebbled.

The next two theorems give an alternative proof of Theorem A.

Theorem C *[6] Let G be a DAG with node set V. Then the pebbler can win the game in time $O(|V|/\log|V|)$.*

Theorem D *(Theorem 3 in [6]) Let C be a Boolean circuit computing a function f. If C can be two-person pebbled with t pebbles, then there exists a tree-like circuit of depth $2t+1$ that also computes f.*

We will use Theorem D to obtain our results for layered circuits and circuits with small separators. Note that Paul, Tarjan, and Celoni [14] gave a pebbling strategy for layered graphs but used the rules of a different pebble game, which does not imply bounds on the depth.

3 Size versus Depth for Layered Circuits

The following lemma gives an adaptive strategy in the two-person pebble game for layered circuits.

Lemma 2. *Let C be a layered circuit of size s. Then C can be two-person pebbled in time $O(\sqrt{s\log s})$. That is, the pebbler can win by using $O(\sqrt{s\log s})$ pebbles.*

Proof. First note that at any point in the game, we only need to consider the subcircuit whose single output is the currently challenged node. Thus, in the proof we can assume without loss of generality that the circuit C has only one output, and that the first move of the challenger is to challenge the output gate.

Let $L(0),\ldots,L(d)$ be all the layers, where d is the depth of C, and $L(i)$ is the set of gates with height i. (See previous section for definitions.) Note that $L(0)$ consists of the output gate. We say that a layer $L(i)$ is large if $|L(i)| > y$ and small otherwise. We shall determine the value of y later.

The strategy of the pebbler has two phases. During the first phase, the pebbler forces the challenger to go into a subcircuit between two small layers such

that every layer between the two small layers is large, or into a subcircuit such that all nodes of the subcircuit belong to large layers. In the second phase, the pebbler will win the game within that subcircuit.

During the first phase, the pebbler always pebbles a small layer S_α with $\alpha > \beta$, where S_β is the layer where the challenged node resides in that round. The pebbler continues this phase until the small layer S_α with $\alpha > \beta$ closest to the challenged node is pebbled, or until there are no more such small layers. Note that the pebbler pebbles the small layers $S_0, S_1, \ldots, S_m$ in a divide-and-conquer way depending on the location of the challenged node in each round. Since there are at most s small layers, the number of pebbles used in the first phase is at most $y\lceil \log s \rceil$.

Phase I. Let $S_0, S_1, \ldots, S_m$ be the small layers numbered starting from the output. Note that $S_0 = L(0)$ since $L(0)$ contains only one gate. Define $h(j)$ to be the height of the gates in the jth small layer S_j. We shall define the strategy inductively.

At the beginning of the game (round 1), the challenger challenges the output node, which belongs to $S_0 = L(0)$.

Suppose that for $r \geq 1$ at the beginning of round r the challenger challenges a node $w \in S_j$. Note that since during phase I pebbles are only placed on nodes in small layers, the challenged node w belongs to a small layer in every round within phase I. We have three cases.

1. No small layer $L(h(b)) = S_b$ with $b > j$ exists. That is, every layer $L(k)$ with $k > h(j)$ is a large layer. Then the pebbler continues with the second phase.
2. None of the small layers $L(h(b)) = S_b$ with $b > j$ is pebbled. The pebbler then puts a pebble on each node of $S_{\lceil \frac{m+j}{2} \rceil}$.
3. There exists a small layer $L(h(b)) = S_b$ with $b > j$, such that all nodes in S_b are pebbled, and none of the small layers between S_j and S_b is pebbled. The pebbler then puts a pebble on each node of $S_{\lfloor \frac{b+j}{2} \rfloor}$ if $\lfloor \frac{b+j}{2} \rfloor \neq b$. If $\lfloor \frac{b+j}{2} \rfloor = b$, then there are no small layers between S_j and S_b, and the pebbler continues with the second phase.

Phase II. The pebbler's strategy in the second phase is as follows: Suppose that the challenger challenges node w in the beginning of the kth round for some k. Then the pebbler puts pebbles on the two inputs of w, say u and v. In the $(k+1)$st round, if the challenger stays on w, then the pebbler wins the game. On the other hand, if the challenger challenges one of the inputs of w, WLOG u, then the pebbler puts pebbles on the two inputs of u in the $(k+1)$st round. The game continues inductively this way until at the beginning of pebbler's move either the currently challenged node w is an input of C, or the two immediate predecessors of w are already pebbled. Thus the pebbler wins in the second phase.

Note that in this phase, the pebbler only spends at most two pebbles in each round, and the two pebbles are put on nodes in large layers of C. Moreover, during k rounds of the second phase, the pebbler pebbles nodes from k different

large layers. Since the number of large layers in C is at most $\frac{s}{y}$, the second phase must terminate in at most $\frac{s}{y}$ rounds. Thus, the number of pebbles used in this phase is at most $\frac{2s}{y}$.

The total number of pebbles used throughout the game is at most $p = y\lceil \log s \rceil + 2s/y$. The minimum of this expression is $p = 2\sqrt{2s\lceil \log s \rceil}$, achieved when $y = \sqrt{\frac{2s}{\lceil \log s \rceil}}$. This proves the lemma. □

Proof of Theorem 1. Follows immediately from Theorem D and Lemma 2.

4 Size versus Depth for Synchronous Circuits

The following simple lemma was given in [24]. The results by McColl and Paterson [13], and Gaskov [7] mentioned in the introduction give stronger results. But for our purposes, this slightly weaker bound is sufficient, and we include a simple proof for completeness.

Lemma 3. *[24] For every function $f : \{0,1\}^n \to \{0,1\}$, there exists a synchronous circuit of depth at most $n + \log n + 1$ computing f.*

Proof. The proof is based on considering any DNF of f. The terms can be computed in parallel with depth at most $\log n + 1$, and the number of terms is at most 2^n. This gives the desired depth. Note that any circuit can be made synchronous without increasing its depth. □

Next we prove Theorem 2. Note that in the proof, we use the property of synchronous circuits that given any level $LV(i)$, f is a function of exactly those functions computed at the gates in $LV(i)$. This property allows us to do function composition in terms of two circuits. However, for layered circuits, inputs could be in the jth layer for $j > i$. Thus the property no longer holds for layered circuits that are not synchronous.

Note that the notation LV stands for "levels", while in the previous section we used L for "layers".

Proof of Theorem 2. Let f be the function computed by C. (If C has more than one output, the proof can be applied by considering each output function separately, and combine the resulting small depth circuits.) Since C is synchronous, every level in C forms a cut. Furthermore, given any level $LV(i)$, f is a function of exactly those functions computed at the gates in $LV(i)$. We shall use this special property of synchronous circuits to compute f by the composition of two circuits.

Let $LV(0), LV(1), \ldots, LV(d)$ be the levels in C, where d is the depth of C, and $LV(0)$ contains the inputs. Let y be an integer whose value will be determined later. We say that a level $LV(i)$ is *small* if $|LV(i)| \leq y$ and *large* otherwise.

If C has many outputs, then it is possible that all the levels are large, but then the depth of C is at most $\frac{s}{y}$. Assume that C has at least one small level. Let $LV(k_0)$ be the small level farthest from the output of C.

Now let $g_1, \ldots, g_m$ be the gates in $LV(k_0)$, and let γ_i be the function computed at g_i. As noted above, f is a function of $\gamma_1, \ldots, \gamma_m$. Let $f = f'(\gamma_1, \ldots, \gamma_m)$. Then by Lemma 3, given $\gamma_1, \ldots, \gamma_m$ as inputs, f' can be computed by a synchronous circuit F of depth $O(|LV(k_0)|) = O(y)$.

Let C' be the multiple-output subcircuit of C with outputs $g_1, \ldots, g_m$. That is, C' consists of the levels $LV(0), \ldots, LV(k_0)$ in C, where $LV(k_0)$ contains the outputs of C'. (If $LV(k_0) = LV(0)$, then C' consists of only one level, formed by the inputs of C.) We use the outputs of C' as inputs for the circuit F. The resulting combined circuit F' is a synchronous circuit computing f. Note that all the levels $LV(0), \ldots, LV(k_0 - 1)$ are large. Since there are at most $\frac{s}{y}$ large levels in C, the depth of F' is at most $O(y + \frac{s}{y})$.

Thus, we obtain a synchronous circuit of depth at most $O(y + \frac{s}{y})$. Letting $y = \sqrt{s}$, we can simulate C by a synchronous circuit of depth $O(\sqrt{s})$. □

5 Size versus Depth for Planar Circuits and Circuits with Small Separators

Informally, a node separator of a graph G is a set of nodes whose removal yields two disjoint subgraphs of G that are comparable in size. The following gives a formal definition of a node separator in the fashion of Ullman [23].

Definition 5. *A DAG $G = (V, E)$ has an* $h(t)$-separator*, and we say G is* $h(t)$-separable*, if G has only one node, or it satisfies the following two properties.*

1. *There exists an $S \subseteq V$ of size at most $h(|V|)$ whose removal disconnects G into two subDAGs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, that satisfy $|V_i| \geq \frac{1}{3}|V|$ for $i = 1, 2$.*
2. *The two subDAGs G_1 and G_2 have $h(t)$-separators.*

Note that the two subDAGs could be disconnected within themselves.

Although the separator serves as a natural tool to define a divide-and-conquer strategy, we need something stronger than the separator for the two-person pebble game. This is because the challenged nodes in the game could be arbitrary nodes in the graph. Thus we need to consider every possible subDAG of G.

Definition 6. *A DAG G is* everywhere-$h(t)$-separable*, if any subDAG H of G with one output such that the underlying undirected graph of H is connected, also has an $h(t)$-separator. A circuit C is* everywhere-$h(t)$-separable *if its underlying DAG is everywhere-$h(t)$-separable.*

Theorem 4. *Let $C = (V, E)$ be an everywhere-$h(t)$-separable Boolean circuit, where $h(t) = o(t)$. Then C can be two-person pebbled in time*

$$O\left(\sum_{i=0}^{\lceil \log_{3/2} |V| \rceil} h\left((2/3)^i |V|\right)\right).$$

Proof. Let $p(|V|)$ be the number of pebbles required by the pebbler to win in the two-person pebble game on C. In each round, the pebbler will separate C into two subDAGs to see which subDAG the currently challenged node belongs to, and then recurse on that subDAG. Notice that in general, both components may contain several disjoint subcomponents.

We now define the strategy recursively. If C has only one node, then the pebbler wins immediately after the challenger's initial move. Now suppose that the challenger puts a challenge on a node $g \in C$. Then the pebbler places pebbles on the nodes of some appropriately chosen separator S of C_1, where C_1 is the unique maximal subcircuit of C with g as its output. Let $i \geq 1$. There are two cases in the $(i+1)$th round.

1. The challenger re-challenges g. Let S_i be the separator of C_i chosen in the ith round. Let C_{i+1} be the unique maximal subcircuit of C_i with g as its output, such that the inputs of C_{i+1} are some inputs of C_i upon which g depends, or some nodes in the separator S_i, and the underlying undirected graph of C_{i+1} is connected. Furthermore, we require that every path from the inputs of C_{i+1} to g does not contain any node in S_i. Then the pebbler applies this strategy recursively, and in the next round, looks for an appropriate separator of C_{i+1}.
2. The challenger puts a new challenge on a node $w \in S$. Let C_{i+1} be defined as above, but with w as its output. Then, as above, the pebbler applies this strategy recursively.

Let C_{i1} and C_{i2} be the two subDAGs of C_i defined by the separator S_i. Note that C_{i1} and C_{i2} might be disconnected, but we defined C_{i+1} to be a connected subcircuit of either C_{i1} or C_{i2}. We claim that the pebbler will win after at most $O(\log |V|)$ rounds. To see this, notice that after the first round, the challenger can only challenge a node that has just been challenged, or a node in the separator. So the challenged node is restricted to either $C_{i1} \cup S_i$ or $C_{i2} \cup S_i$ that have sizes at most $(\frac{2}{3} + o(1))|C_i|$ because by assumption C is everywhere-$h(t)$-separable and $h(t) = o(t)$. This also implies that the size of C_{i+1} is at most $(\frac{2}{3} + o(1))|C_i|$, and the game must terminate in at most $O(\log |V|)$ rounds.

For the number of pebbles used, we have the following recursion:

$$p(1) = 0, p(|C_i|) \leq h(|C_i|) + p\left(\left(\frac{2}{3} + o(1)\right)|C_i|\right).$$

Solving the above recursion yields $p(|V|) = O\left(\sum_{i=0}^{\lceil \log_{3/2} |V| \rceil} h\left((2/3)^i |V|\right)\right)$.

□

For planar graphs, the following theorem is due to Lipton and Tarjan [11].

Theorem E *[11] Given a planar graph G with node set V, G has a separator of size $O(\sqrt{|V|})$.*

Since any subgraph of a planar graph is still planar, every planar graph is everywhere-$O(\sqrt{t})$-separable.

Proof of Theorem 3. The claim follows by Theorem 4, Theorem D, and the observation that any tree-like circuit is also planar. □

For graphs with bounded genus, we have the following theorem due to Gilbert, Hutchinson, and Tarjan [8].

Theorem F *[8] Given a graph G with node set V and genus g, G has a separator of size $O(\sqrt{g|V|})$.*

Since any subgraph of a graph G with genus g cannot have genus more than g, G is everywhere-$O(\sqrt{gt})$-separable. By Theorem 4 and Theorem D we obtain:

Theorem 5. *Let C be a Boolean circuit of size s such that its underlying graph of C has genus g. Then there exists a tree-like circuit F of depth $O(\sqrt{gs})$ that computes the same function.*

Let K_k denote the complete graph on k nodes. For graphs having no K_k as a minor, we have the following theorem due to Alon, Seymour, and Thomas [2].

Theorem G *[2] Given a graph G with node set V and having no K_k as a minor, G has a separator of size $O(k^{\frac{3}{2}}|V|^{\frac{1}{2}})$.*

If a graph G does not have K_k as its minor, neither can any subgraph of G. So G is everywhere-$O(k^{3/2}t^{1/2})$-separable. As above, this gives:

Theorem 6. *Let C be a Boolean circuit of size s such that its underlying graph of C does not contain K_k as a minor. Then there exists a tree-like circuit F of depth $O(k^{\frac{3}{2}}s^{\frac{1}{2}})$ that computes the same function.*

Acknowledgements

We thank the anonymous referees for helpful comments.

References

1. Leonard M. Adleman and Michael C. Loui: Space-bounded simulation of multitape turing machines. Theory of Computing Systems V.14 NO.1, 215–222 (1981)
2. Noga Alon, Paul Seymour, Robin Thomas: A Separator Theorem for Graphs with an Excluded Minor and its Applications. Proceedings of the ACM Symposium on Theory of Computing, 293–299 (1990)
3. Edward G. Belaga: Locally Synchronous Complexity in the Light of the Trans-Box Method. STACS, Lecture Notes in Computer Science V.166, 129–139 (1984)
4. Allan Borodin: On Relating Time and Space to Size and Depth. SIAM Journal on Computing V.6 NO.4, 733–744 (1977)

5. Stephen A. Cook: An Observation on Time-Storage Trade Off. Journal of Computer and System Sciences V.9 NO.3, 308–316 (1974)
6. Patrick Dymond and Martin Tompa: Speedups of Deterministic Machines by Synchronous Parallel Machines. J. Comp. and Sys. Sci. V.30 NO.2, 149–161 (1985)
7. Gaskov: The depth of Boolean functions. Probl. Kibernet. V.34, 265–268 (1978)
8. J.R. Gilbert, J.P. Hutchinson, R.E. Tarjan: A Separator Theorem for Graphs of Bounded Genus. Journal of Algorithms V.5 NO.3, 391–407 (1984)
9. L.H. Harper: An $n \log n$ Lower Bound on Synchronous Combinational Complexity. Proc. AMS V.64 NO.2, 300–306 (1977)
10. J. Hopcroft and W. Paul and L. Valiant: On Time Versus Space. Theory of Computation V.24 NO.2, 332–337 (1977)
11. R. Lipton and R.E. Tarjan: A Separator Theorem for Planar Graphs. SIAM J. Appl. Math. V.36, 177–189 (1979)
12. R. Lipton and R.E. Tarjan: Applications of a Planar Separator Theorem. SIAM Journal on Computing V.9 NO.3, 615–627 (1980)
13. W.F. McColl and M.S. Paterson: The depth of all Boolean functions. SIAM J. on Comp. V.6, 373–380 (1977)
14. W. Paul and R.E. Tarjan and J. Celoni: Space Bounds for a Game on Graphs. Mathematical Systems Theory V.10, 239–251 (1977)
15. M.S. Paterson and C.Hewitt: Comparative Schematology. MIT AI Memo 464(1978)
16. M.S. Paterson and L.G. Valiant: Circuit Size is Nonlinear in Depth. Theoretical Computer Science V.2 NO.3, 397-400 (1976)
17. N. Pippenger and M. Fischer: Relations Among Complexity Measures. Journal of the ACM V.26, 361–381 (1979)
18. N. Pippenger: Pebbling. Mathematical Foundations of Computer Science (1980)
19. Arnold Rosenberg and Lenwood Heath: Graph Separators with Applications (2001)
20. P.M. Spira: On time-hardware complexity tradeoffs for Boolean functions. In Proc. 4th Hawaii Symp. on System Sciences, 525–527 (1971)
21. Larry Stockmeyer and Uzi Vishkin: Simulation of Parallel Random Access Machines by Circuits. SIAM Journal on Computing V.13 NO.2, 409–422 (1984)
22. G. Turán: On restricted Boolean circuits. Fund. of Comp. Theory, 460–469 (1989)
23. Jeffrey D. Ullman: Computational Aspects of VLSI. (1984)
24. Ingo Wegener: The Complexity of Boolean Functions. (1987)