

# The Size and Depth of Layered Boolean Circuits

Anna Gál<sup>a,1</sup>, Jing-Tang Jang<sup>a,2</sup>

<sup>a</sup>*Dept. of Computer Science, University of Texas at Austin,  
Austin, TX 78712-1188, USA*

---

## Abstract

We consider the relationship between size and depth for layered Boolean circuits and synchronous circuits. 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 synchronous circuits of size  $s$ , we obtain simulations of depth  $O(\sqrt{s})$ . The best known result so far was by Paterson and Valiant [17], 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.

*Keywords:* Computational complexity, 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  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , 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.

Pippenger and Fischer [18] showed that for  $t(n) \geq n$ ,  $DTIME(t(n))$  can be simulated by logspace uniform families of circuits of size  $O(t(n) \log t(n))$ .

---

*Email addresses:* [panni@cs.utexas.edu](mailto:panni@cs.utexas.edu) (Anna Gál), [keith@cs.utexas.edu](mailto:keith@cs.utexas.edu) (Jing-Tang Jang)

<sup>1</sup>Supported in part by NSF Grants CCF-0830756 and CCF-1018060

<sup>2</sup>Supported in part by MCD fellowship from Dept. of Computer Science, UT Austin

Borodin [4] showed that for  $s(n) \geq \log n$ , languages computed by logspace uniform families of circuits of depth  $s(n)$  are contained in  $SPACE(s(n))$ , and  $SPACE(s(n))$  can be simulated by logspace uniform families of circuits of depth  $O(s^2(n))$ . Furthermore, circuit depth is related to parallel computation time [22]. These 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 [17], and later proved by Dymond and Tompa [6] using another method.

**Theorem A.** [17, 6] *Given a Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , we have  $D(f) = O(C(f)/\log C(f))$ .*

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 [14] showed that every Boolean function depending on  $n$  variables has circuit depth at most  $n + 1$ . There is an even stronger result by Gaskov [8] 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  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  such that  $C(f)$  is exponential in  $n$ , [14] and [8] 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 [14] and [8] only when  $C(f) = o(n \log n)$ . Improving Theorem A would yield improvements over [14] and [8] 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 [11] proved the following analogous theorem about sequential time and space, and Adleman and Loui [1] later gave an alternative proof.

**Theorem B.** [11, 1]  *$DTIME(t(n)) \subseteq SPACE(t(n)/\log t(n))$ .*

By the results of [18] and [4] mentioned above, improving Theorem A by at least a polylog factor in a uniform setting immediately improves Theorem B.

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 [21] 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 [17, 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 [10]. Planar circuits were introduced by Lipton and Tarjan [13]. 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 [15] (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 [23] 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.

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 can obtain slightly better bounds for synchronous 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 [13]. In [7] we showed that planar circuits of size  $s$  can be simulated by planar circuits of depth  $O(\sqrt{s})$ . For planar circuits, we used the fact that every planar circuit of size  $s$  has a separator of size  $O(\sqrt{s})$  [12]. Informally, the separator of a graph is a subset of the nodes whose removal yields two subgraphs of comparable sizes. This allowed us to use a simple divide-and-conquer strategy. Graphs with small separators include trees, planar graphs [12], graphs with bounded genus [9], 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 [20] 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 [17] 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  $O(s^2)$  [25]. 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  $\{x_1, \dots, x_n\}$ , 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 [25] for more background on Boolean circuits.

**Definition 1.** [10] **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 maximum depth of an output gate, over all outputs of  $C$ .

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, “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  $g_2$  as the output.

**Lemma 3.** *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. [16, 5]. See [19] 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 choosing to challenge 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 choose either the currently challenged node or a node that was pebbled in the pebbler’s most recent move. The game continues until at the beginning of the pebbler’s move, all immediate 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  $t$* . Note that pebbles are never removed: once a node is pebbled it remains pebbled during the rest of the game.

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 the underlying graph of  $C$  can be two-person pebbled with  $t$  pebbles, then there exists a tree-like circuit of depth  $2t + 1$  that also computes  $f$ .*

Paul, Tarjan, and Celoni [15] 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 4.** *Let  $C$  be a layered circuit of size  $s$ . Then the underlying graph of  $C$  can be two-person pebbled in time  $O(\sqrt{s \log s})$ .*

*Proof.* In our proof, the pebbler will always place pebbles only on nodes with larger height than the currently challenged node. Since the challenger can choose only either the currently challenged node or a node that was pebbled in the

pebbler's most recent move, 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. Furthermore, our strategy forces the challenger to choose vertices with increasing height within  $C$ , except when rechallenging the currently challenged node.

Let  $L(0), \dots, L(d)$  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, in each round the pebbler places pebbles on *every* node of a small layer  $S_\alpha$  with  $\alpha > \beta$ , where  $S_\beta$  is the layer where the challenged node resides in that round. The pebbler continues the first phase until the small layer  $S_\alpha$  with  $\alpha > \beta$  closest to the challenged node is pebbled, or until there are no more such small layers. The pebbler chooses the next small layer 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  $S_0, \dots, S_m$  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  $j$ th small layer  $S_j$ . We define the strategy inductively.

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

Suppose that for  $r \geq 1$  at the beginning of round  $r$  the challenger chooses a node  $w \in S_j$ . 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. According to our strategy, a small layer either has a pebble on all of its nodes (in this case we say that the layer is pebbled), or has no pebbles on any of its nodes. 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 chooses node  $w$  in the beginning of the  $k$ th round. 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 chooses 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 places pebbles on 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.  $\square$

PROOF OF THEOREM 1. Theorem D and Lemma 4 together give a tree-like circuit of depth  $O(\sqrt{s\log s})$ . Since tree-like circuits are layered, this proves the theorem.  $\square$

#### 4. Size versus Depth for Synchronous Circuits

The following simple lemma was given in [25]. The results by McColl and Paterson [14], and Gaskov [8] 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 5.** [25] *For every function  $f : \{0,1\}^n \rightarrow \{0,1\}$ , 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.  $\square$

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

Note that the notation  $L_V$  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  $LV(0), \dots, LV(d)$  be the levels in  $C$ , where  $d$  is the depth of  $C$ , and  $LV(0)$  contains the inputs. Let  $y$  be an integer to 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  $g_1, \dots, g_m$  be the gates in  $LV(k_0)$ , and let  $\gamma_i$  be the function computed at  $g_i$ . As noted above,  $f$  is a function of  $\gamma_1, \dots, \gamma_m$ . Let  $f' = f'(\gamma_1, \dots, \gamma_m)$ . Then by Lemma 5, given  $\gamma_1, \dots, \gamma_m$  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  $g_1, \dots, g_m$ . That is,  $C'$  consists of the levels  $LV(0), \dots, LV(k_0)$  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  $LV(0), \dots, LV(k_0 - 1)$  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  $O(\sqrt{s})$ .  $\square$

#### Acknowledgements

A preliminary version of this paper appeared in the proceedings of LATIN 2010 [7]. We thank the anonymous referees for helpful comments.

#### References

- [1] L. Adleman and M. Loui: Space-bounded Simulation of Multitape Turing Machines. *Theory of Computing Systems* V.14 (1), 215–222 (1981)
- [2] N. Alon, P. Seymour, R. Thomas: A Separator Theorem for Graphs with an Excluded Minor and its Applications. *STOC*, 293–299 (1990)
- [3] E. Belaga: Locally Synchronous Complexity in the Light of the Trans-Box Method. *STACS, Lecture Notes in Computer Science* V.166, 129–139 (1984)
- [4] A. Borodin: On Relating Time and Space to Size and Depth. *SIAM Journal on Computing* V.6 (4), 733–744 (1977)
- [5] S. Cook: An Observation on Time-Storage Trade Off. *Journal of Computer and System Sciences* V.9 (3), 308–316 (1974)

- [6] P. Dymond and M. Tompa: Speedups of Deterministic Machines by Synchronous Parallel Machines. *J. Comp. and Sys. Sci.* V.30 (2), 149–161 (1985)
- [7] A. Gál, J. T. Jang: The Size and Depth of Layered Boolean Circuits. *Proceedings of LATIN 2010, LNCS 6034*, pp. 372–383, 2010.
- [8] Gaskov: The Depth of Boolean Functions. *Probl. Kiber.* V.34, 265–268 (1978)
- [9] 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)
- [10] L.H. Harper: An  $n \log n$  Lower Bound on Synchronous Combinational Complexity. *Proc. AMS* V.64 NO.2, 300–306 (1977)
- [11] J. Hopcroft and W. Paul and L. Valiant: On Time Versus Space. *Theory of Computation* V.24 NO.2, 332–337 (1977)
- [12] R. Lipton and R.E. Tarjan: A Separator Theorem for Planar Graphs. *SIAM J. Appl. Math.* V.36, 177–189 (1979)
- [13] R. Lipton and R.E. Tarjan: Applications of a Planar Separator Theorem. *SIAM Journal on Computing* V.9 NO.3, 615–627 (1980)
- [14] W.F. McColl and M.S. Paterson: The Depth of All Boolean Functions. *SIAM J. on Comp.* V.6, 373–380 (1977)
- [15] W. Paul and R.E. Tarjan and J. Celoni: Space Bounds for a Game on Graphs. *Mathematical Systems Theory* V.10, 239–251 (1977)
- [16] M.S. Paterson and C.Hewitt: Comparative Schematology. MIT AI Memo 464(1978)
- [17] M.S. Paterson and L.G. Valiant: Circuit Size is Nonlinear in Depth. *Theoretical Computer Science* V.2 NO.3, 397–400 (1976)
- [18] N. Pippenger and M. Fischer: Relations Among Complexity Measures. *Journal of the ACM* V.26, 361–381 (1979)
- [19] N. Pippenger: Pebbling. *Math. Foundations of Computer Science* (1980)
- [20] A. Rosenberg and L. Heath: *Graph Separators with Applications* (2001)
- [21] P.M. Spira: On Time-Hardware Complexity Tradeoffs for Boolean Functions. In *Proc. 4th Hawaii Symp. on System Sciences*, 525–527 (1971)
- [22] L. Stockmeyer and U. Vishkin: Simulation of Parallel Random Access Machines by Circuits. *SIAM Journal on Computing* V.13 NO.2, 409–422 (1984)
- [23] G. Turán: On Restricted Boolean Circuits. *Fund. of Comp. Theory*, 460–469 (1989)
- [24] J. D. Ullman: *Computational Aspects of VLSI*. (1984)
- [25] I. Wegener: *The Complexity of Boolean Functions*. (1987)