The Potential of the Approximation Method[1]

Kazuyuki Amano　　Akira Maruoka

Graduate School of Information Sciences, Tohoku University
Aoba05 Aramaki Aoba-ku, Sendai 980-8579, JAPAN
Phone: +81–22–217–7149, Fax: +81–22–263–9414

E-mail:{ama,maruoka}@ecei.tohoku.ac.jp

[1] A preliminary version of this paper appeared in *proc. 37th IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 431–440

Abstract Developing certain techniques for the approximation method, we establish precise versions of the following statements concerning lower bounds for circuits that detect cliques of size s in a graph with m vertices: For $5 \leq s \leq m/4$, a monotone circuit computing CLIQUE(m, s) contains at least $(1/2)1.8^{\min(\sqrt{s-1}/2, m/(4s))}$ gates: If a non-monotone circuit computes CLIQUE using a "small" amount of negation, then the circuit contains an exponential number of gates. The former is proved by using so called bottleneck counting argument within the framework of approximation, and the latter is verified by introducing a notion of restricting negation in circuits and generalizing these arguments to non-monotone cases.

1 Introduction

Since Razborov introduced the approximation method[R1] and used it successfully to prove a superpolynomial lower bound on the size of monotone circuits computing the clique function, much effort has been devoted to explore the method and to derive good lower bounds using it[K, NM, R1, R2, RR, W]. Alon and Boppana[AB] obtained an exponential lower bound on the size of monotone circuits that compute the clique function, through a clever use of the approximation method. Using an apparently different argument, called *bottleneck counting*, Haken[H] derived an exponential lower bound on the size of monotone circuits that compute a variant of the clique function. Razborov, in subsequent work, showed that the approximation method can, in principle, provide tight lower bounds for non-monotone circuits[R2]. In spite of these advances, it remains a challenging problem to apply the method successfully to obtain good lower bounds on the size of circuits computing a given problem. This is especially hard when we deal with non-monotone circuits.

In this paper, we extend our knowledge of the method in two ways. First, we show how to use the bottleneck counting argument within the framework of the approximation method. This allows us to present better lower bounds for a certain clique problem, and to simplify considerably the proof both for the clique problem and for a problem that Alon and Boppana addressed. Second, we extend the method for non-monotone circuits for the clique problem, although we must restrict ourselves to circuits with a restricted amount of negation.

More precisely, the results obtained in these ways are as follows. First, denoting by CLIQUE(m,s) the clique function detecting cliques of size s in a graph with m vertices, we obtain a lower bound on the size of monotone circuits computing CLIQUE(m,s), which is given as $(1/2)1.8^{\min(\sqrt{s-1}/2, m/(4s))}$ for $5 \le s \le m/4$. This lower bound should be contrasted with the fact that the best lower bound obtained so far for the clique function is $(1/8)(m/(4s^{3/2}\log m))^{\sqrt{s+1}/2}$ for $3 \le s \le (1/4)(m/\log m)^{2/3}$ which is due to Alon and Boppana[AB]. So as for the largest monotone lower bound for the clique function, our bound is $\exp(\Omega(m^{1/3}))$ for $s = \lceil m^{2/3} \rceil$, whereas the one due to Alon and Boppana is $\exp(\Omega((m/\log m)^{1/3}))$ for $s = (1/4)(m/\log m)^{2/3}$. To derive these lower bounds, we define approximators, instead of relying on the sunflower contraction, in terms of DNF and CNF formulas such that the size (or the length) of terms and clauses in the formulas is limited appropriately. In this way, we succeeded to simplify greatly the proofs to obtain these bounds, using only elementary combinatorics. The similarity between the method of approximations and the bottleneck counting arguments were independently found by Berg and

Ulfberg[BU], Jukna[J] and Simon and Tsai[ST]. More recently, Harnik and Raz[HR] proved a higher lower bound of $2^{\Omega((n/\log n)^{1/3})}$ on the monotone size of an explicit function by introducing the *monotone switching lemma* between DNF and CNF approximators.

Second, we extend the method discussed in the first part of this paper so as to apply for the case of non-monotone circuits, provided that an amount of negation in circuits is restricted somehow. To do so, we introduce a notion of restricting negation used in a circuit. For circuit C with input variables $x_1, \ldots x_n, \bar{x}_1, \ldots \bar{x}_n$, let its monotone analogue, denoted C_{ex}, be the monotone circuit obtained by replacing each negated variable $\bar{x}_i$ in C with a new input variable y_i. There exists an obvious correspondence between minterms of the function computed by a circuit C and those of the function computed by the corresponding circuit C_{ex}. When we deal with circuit C_{ex} we only pay attention to such minterms. Then we consider the maximum number of variables $y_1, \ldots, y_n$ appearing in such a minterm computed by C_{ex} as the parameter indicating an amount of negation used in C. So setting the parameter to an integer between 0 and n, we can get a variety of negation-restriction ranging from monotone circuits to general circuits with no restriction. Based on the notion of restricting negation, we verify that if a non-monotone circuit C computes CLIQUE(m, m^c) with the parameter at most $m^{c/2-\epsilon}$, then the size of the circuit C is given by $\exp(\Omega(m^{c/2}))$, where c and ϵ are any constants such that $0 < c \leq 2/3$ and $0 < \epsilon < c$. One might think of the result as showing the extent to which an argument based on bottleneck counting can be generalized to apply in non-monotone cases.

2 Preliminaries

A *Boolean circuit* is a directed acyclic graph. The nodes of indegree 0 are called input nodes, and are labeled with x_i or its negation $\bar{x}_i$. The nodes of indegree 2 are called gates and are labeled with the Boolean functions AND and OR. AND and OR gates are also called $\wedge$ and $\vee$ gates, respectively. A circuit represents a Boolean function computed at a node in the circuit designated as the output node. In particular, a Boolean circuit without input nodes labeled with negated variables is called *monotone*. The *size* of a circuit C is the number of gates in the circuit C. For ease of arguments, in Section 3 a monotone circuit is further assumed to satisfy the following conditions: Any input of an $\vee$ gate is connected to either an output of $\wedge$ gate or an input node, and any input of an $\wedge$ gate is connected to either an output of $\vee$ gate or an input node; The output gate is an $\wedge$ gate. Any monotone circuit can be easily converted to the one satisfying these conditions by replacing, if necessarily, a line connecting gates by a dummy gate(that can be thought of as an $\vee$ gate or an $\wedge$ gate, appropriately) with their two inputs being connected to an output of the same gate, and hence by at most doubling the size of the circuit. The *circuit complexity*(*monotone circuit complexity*) of a function f is the size of the smallest circuit(monotone circuit) computing f.

3 Lower bounds for monotone circuit complexity based on CNF and DNF based approximators

In this section, we derive lower bounds for the clique problem and also those for a problem defined in terms of polynomials. The clique function, denoted CLIQUE(m, s), of $m(m-1)/2$ variables $\{x_{i,j} \mid 1 \leq i < j \leq m\}$ is defined to take value 1 if and only if the undirected graph on m vertices represented in the obvious way by the input contains a clique of size s. The graphs will be identified with the assignments to the variables specifying the graphs.

Theorem 1 If $5 \leq s \leq m/4$, then any monotone circuit that computes the function CLIQUE(m, s) contains at least $(1/2)1.8^{\min(\sqrt{s-1}/2, m/(4s))}$ gates. In particular, the monotone circuit complexity of CLIQUE$(m, \lceil m^{2/3} \rceil)$ is $\exp(\Omega(m^{1/3}))$. □

We now proceed to proving Theorem 1 in the framework of approximation. We define good and bad graphs which are used as test inputs to compare the circuit's behavior with the behavior of the clique function. A graph is called *good* if it consists of a clique on some set of s vertices, and no other edges. A graph is called *bad* if there exists a partition of the vertices into m mod $(s-1)$ sets of size $\lceil m/(s-1) \rceil$ and $s-1-(m \bmod (s-1))$ sets of size $\lfloor m/(s-1) \rfloor$ such that any two vertices chosen from different sets have an edge between them, and no other edges exist. Note that the function CLIQUE(m, s) outputs value 1 on every good graph and outputs 0 on every bad graph.

Fact 1 There are

$$\frac{m!}{s!(m-s)!}$$

good graphs and

$$\frac{m!}{(\lceil m/(s-1) \rceil !)^w (\lfloor m/(s-1) \rfloor !)^{s-1-w} w!(s-1-w)!}$$

bad graphs, where $w = m \bmod (s-1)$. □

Let t be a term or a clause. The *endpoint set* of t is a set of all endpoints of the edges corresponding to variables in t. The *size* of t is the cardinality of the endpoint set of t.

We are ready to define an approximator circuit to approximate a monotone circuit. An approximator circuit is just the same as a Boolean circuit except that $\vee$ and $\wedge$ gates are replaced with $\overline{\vee}$ and $\overline{\wedge}$ gates, respectively, which will be defined below. Approximators computed at the nodes in an approximator circuit are defined in bottom-up fashion, starting from the input nodes and then working up. The approximate gates, denoted $\overline{\vee}$ gate and $\overline{\wedge}$ gate, are defined to behave as follows. The approximator corresponding to input variable x_i is defined to be x_i itself. The functions of an $\overline{\vee}$ gate and an $\overline{\wedge}$ gate are defined as follows. Integers l and r in the definition are chosen later.

- $\overline{\vee}$: Let f_1^D and f_2^D be two approximators, represented by monotone DNF formulas, feeding into an $\overline{\vee}$ gate. The approximate OR of these approximators is the monotone CNF formula obtained by transforming the monotone DNF formula $f_1^D \vee f_2^D$ into the monotone CNF formula and then taking away all the clauses whose sizes exceed r.

- $\overline{\wedge}$: Let f_1^C and f_2^C be two approximators, represented by monotone CNF formulas, feeding into an $\overline{\wedge}$ gate. The approximate AND of these approximators is the monotone DNF formula obtained by transforming the monotone CNF formula $f_1^C \wedge f_2^C$ into the monotone DNF formula and then taking away all the terms whose sizes exceed l.

Since we assumed that no output of an $\vee$ (resp. $\wedge$) gate is connected to an $\vee$ (resp. $\wedge$) gate, an approximator computed at an $\overline{\vee}$ gate is given by a monotone CNF formula, and an approximator computed at an $\overline{\wedge}$ gate is given by a monotone DNF formula, where both formulas satisfy the size requirements. For Boolean functions f and g, let us denote $f \leq g$ if and only if $f(x) \leq g(x)$ holds for any input vector x. Note that by the definition $(f_1 \overline{\vee} f_2) \geq (f_1 \vee f_2)$ holds for any monotone DNF formulas f_1 and f_2. Similarly, $(f_1 \overline{\wedge} f_2) \leq (f_1 \wedge f_2)$ holds for any monotone CNF formulas f_1 and f_2.

Let C be a monotone circuit computing CLIQUE(m, s), and let $\overline{C}$ (called the approximator circuit corresponding to C) denote the circuit obtained by replacing all of the $\vee$ and $\wedge$ gates in C by $\overline{\vee}$ and $\overline{\wedge}$ gates, respectively. Since the computation in $\overline{C}$ proceeds from bottom to top, it is easy to see that, for any good graph that makes approximator circuit $\overline{C}$ output 0, there exists an $\overline{\wedge}$ gate that outputs 0 for the good graph because of taking the terms away in defining its output approximator. So, an $\wedge$ gate feeded the same input approximators as those to the $\overline{\wedge}$ gate outputs 1. Similarly, for any bad graph that makes approximator circuit $\overline{C}$ outputs 1, there exists an $\overline{\vee}$ gate that outputs 1 for the bad graph because of taking the clauses away in defining its output approximator.

The proof of Theorem 1 proceeds as follows: First, show that either of the number of good graphs that are classified wrongly by the approximator circuit or the number of bad graphs classified wrongly is large (Lemma 1). Second, show that the number of bad graphs for which an approximate gate $\vee$ (i.e., $\overline{\vee}$ gate) behaves differently from an $\vee$ gate is small (Lemma 2) and similarly the number of good graphs for which an approximate $\wedge$ gate (i.e., $\overline{\wedge}$ gate) behaves differently from a $\wedge$ gate is small (Lemma 3). Finally, calculate the numbers obtained by dividing the numbers of bad and good graphs classified wrongly by an approximator circuit by the numbers of graphs for which the corresponding approximate gates behaves differently, respectively, and show that the larger of the two numbers calculated becomes large, completing the proof of Theorem 1. It is worthwhile to note that it is straightforward to extend our proofs to arbitrary fan-in circuit without changing the lower bounds we obtained.

The parameters l and r are put as $l = \lfloor \sqrt{s-1}/2 \rfloor$ and $r = \lfloor m/(4s) \rfloor$.

Lemma 1 An approximator circuit either outputs identically 0 or outputs 1 on at least one half of the bad graphs.

Proof Let $\overline{f}$ be the approximator function that an approximator circuit computes. Because of the assumption that the output gate of an approximator circuit is $\wedge$ gate, $\overline{f}$ can be represented by a monotone DNF formula consisting of terms of size at most l. If $\overline{f}$ is identically 0 then the first conclusion holds. If not, then there is a term t whose size is less than or equal to l such that $\overline{f} \geq t$ holds. In what follows, bad graphs are represented as one to one mappings from vertex set $\{v_1, \ldots, v_m\}$ to $\{(1,1), \ldots, (1, \lceil m/(s-1) \rceil), \ldots, (w,1), \ldots, (w, \lceil m/(s-1) \rceil), (w+1, 1), \ldots, (w+1, \lfloor m/(s-1) \rfloor), (w+2, 1), \ldots, (s-1, \lfloor m/(s-1) \rfloor)\}$. Such a mapping specifies a bad graph in the obvious way: Two vertices in the graph have an edge between them if and only if the mapping assigns to the vertices a pair with different first components. Note that there exist many mappings corresponding to one bad graph. It is easy to see that the ratio of mappings that satisfy the condition that there is a variable x in the term t such that two vertices incident to x are assigned a pair with the same first component, i.e., the term t outputs 0 on bad graphs specified by such mappings, is at most $(l(l-1)/2)(\lceil m/(s-1) \rceil / m)$. Recalling $l = \lfloor \sqrt{s-1}/2 \rfloor$, above quantity is bounded from above by $1/2$. Therefore, the ratio of bad graphs such that $\overline{f}$ outputs 1 on them is at least $1/2$. □

Lemma 2 Suppose that an $\vee$ gate and an $\bigtriangledown$ gate are given as input the same monotone DNF formulas such that the size of terms in the formulas is equal to or less than l. Then the number of bad graphs for which the $\vee$ and $\bigtriangledown$ gates produce different outputs (the $\vee$ gate produces 0, whereas the $\bigtriangledown$ gate produces 1) is at most

$$\frac{(m/2)^{r+1}(m-r-1)!}{(\lceil m/(s-1)\rceil!)^{w}(\lfloor m/(s-1)\rfloor!)^{s-1-w}w!(s-1-w)!}. \tag{1}$$

Proof Suppose that an $\vee$ gate and an $\bigtriangledown$ gate are given as input the same monotone DNF formulas, denoted f_1^D and f_2^D, such that the size of any term in the formulas is equal to or smaller than l. Let $f_1^D \vee f_2^D$ and $f_1^D \bigtriangledown f_2^D$ be denoted by f^D and f^C, respectively. Let $t_1, \ldots, t_q$ be the complete list of terms in f^D. Then each t_i contains at most $l(l-1)/2$ variables. We shall count the number of bad graphs x such that both $f^D(x) = 0$ and $f^C(x) = 1$ hold. As in the proof of Lemma 1, bad graphs are represented as mappings described there. Instead of counting bad graphs directly, we count mappings corresponding to bad graphs. The number in question is bounded from above by the number of the mapping corresponding to bad graphs x such that $f^D(x) = 0$ and $f^C(x) = 1$ divided by the number of mappings corresponding to one bad graph. (The latter number is independent of a bad graph chosen.) Since the latter is given by the denominator of (1), it suffices to estimate the former. The former is the number of mappings corresponding to bad graphs x that do not satisfy any of $t_1, \ldots, t_q$ but satisfy all clauses in f^C. We count how many ways one could choose variables each from terms t_1 up to t_q and assign pairs of integers to the endpoints associated with the variables chosen so that the corresponding bad graphs x satisfy $f^D(x) = 0$ and $f^C(x) = 1$.

Suppose that we proceed to term t_i and hence some of the endpoints associated with variables from terms t_1 to t_{i-1} are already assigned distinct pairs of integers. This partial assignment assigns 0 and 1 to some of variables in the way mentioned above. We first consider the extreme cases. If there exists a variable in term t_i already assigned 0 by the partial assignment, we skip to the next term t_{i+1}. The other extreme case occurs when all the variables in term t_i are assigned 1 so far. In this case term t_i will never take value 0, hence we don't need to consider the case.

If neither of these extreme cases happens, choose a variable from term t_i such that at least one of the vertices associated with the variable is not assigned a pair of integers. There are two cases to consider: If exactly one of the vertices is assigned so far, then assign to the remaining vertex a pair whose first component is identical to the first component of the pair of integers assigned to the other vertex so that the variable associated with the two vertices is assigned 0. In this case, there are at most $\lceil m/(s-1)\rceil - 1 \leq m/(s-1)$ ways of assigning pairs of integers to the vertex. On the other hand, if both of the vertices are not chosen so far, assign to these vertices pairs of integers with their first components being the same so that the variable associated with the two vertices is assigned 0. So, for the two vertices, there are at most $(s-1)(\lceil m/(s-1)\rceil)(\lceil m/(s-1)\rceil - 1) \leq 2m^2/(s-1)$ ways of assigning pairs of integers.

Suppose that there exist k variables in term t_i such that exactly one of the vertices corresponding to the variables is assigned a pair of integers so far. Then there exist at most $l(l-1)/2 - k$ variables in term t_i such that none of the vertices associated with the variables is assigned a pair of integers so far. So the number of ways of choosing an unassigned vertex in the endpoints of variables in t_i and assigning a pair of integers to the chosen vertex is bounded from above by

$$\max_k (km/(s-1) + \sqrt{(l(l-1)/2-k)(2m^2/(s-1))}), \tag{2}$$

where integer k ranges from 0 to $l(l-1)/2$. This is because doing something to two vertices in i ways can be regarded as doing something to a vertex appropriately in $\sqrt{i}$ ways twice successively. Because of $l \leq \sqrt{s-1}/2$, a simple calculation shows that the quantity above is maximized when $k = 0$, and is bounded from above by $m/2$.

By the definition of $\overline{\vee}$ gate, it is easy to see that the graphs x corresponding to the mappings specified in this way satisfy $f^D(x) = 0$ and $f^C(x) = 1$ only if there exist more than r vertices assigned pairs of integers in the above procedure. So the number of mappings corresponding to such bad graphs is bounded from above by the number of mappings such that $r+1$ vertices are assigned pairs of integers multiplied by the number of ways of assigning arbitrarily distinct pairs of integers to the remaining $m-r-1$ vertices. The resulting number is given by the numerator of (1), completing the proof. $\square$

Lemma 3 Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same monotone CNF formulas such that the size of terms in the formulas is equal to or less than r. Then the number of good graphs for which the $\wedge$ gate and the $\overline{\wedge}$ gate produce different outputs (the $\wedge$ gate produces 1, whereas the $\overline{\wedge}$ gate produces 0) is at most

$$\frac{(m/2)^{l+1}(m-l-1)!}{s!(m-s)!}.$$

Proof The proof is similar to that of Lemma 2. Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same monotone CNF formulas, denoted f_1^C and f_2^C, as in the lemma. Let $f^C = f_1^C \wedge f_2^C$ and $f^D = f_1^C \overline{\wedge} f_2^C$. Let $c_1, \ldots, c_q$ be the complete list of clauses in f^C, where each c_i contains at most $r(r-1)/2$ variables. The number in question is equal to the number of good graphs x such that $f^C(x) = 1$ and $f^D(x) = 0$ hold.

Instead of the mappings from vertices to the integer pairs in the case of Lemma 2, we consider one to one mappings from the vertex set $\{v_1, \ldots, v_m\}$ to the integer set $\{1, \ldots, m\}$. Such a mapping is thought of specifying a good graph such that the set of vertices assigned with integers from 1 to s forms a clique. The number mentioned in the lemma is at most the number of mappings corresponding to good graphs x that satisfy all clauses of $c_1, \ldots, c_q$ but do not satisfy any term in f^D divided by the number of mappings corresponding to one good graph.

As in the case of Lemma 2, we count how many ways one could choose variables each from clauses of $c_1, \ldots, c_q$ and assign integers to the vertices associated with the variables chosen so that all of the clauses are satisfied but the disjunctive normal form formulas $f_1^C \overline{\wedge} f_2^C$ is not satisfied. Suppose that we proceed to clause c_i and hence some of the vertices are assigned integers from 1 to s so that all of the clauses from c_1 to c_{i-1} are satisfied. Such an assignment is called a partial assignment. If there exists a variable in c_i assigned 1 by the partial assignment, we skip to the next clause c_{i+1}.

The variable $x_{j,k}$ is said to be incident to vertices v_j and v_k. To count the number of ways of making clause c_i take value 1, there are two cases to consider: Choose a variable in c_i incident to two vertices, one is assigned an integer by the partial assignment and the other is not; Choose a variable in c_i incident to two vertices that are not assigned integers so far.

As in the proof of Lemma 2, the number of ways of choosing unassigned vertices and assigning integers to the vertices so as to make clause c_i take value 1 is bounded from above by

$$r(s-1) + \sqrt{(r(r-1)/2)s(s-1)} < 2rs. \tag{3}$$

Since $r \leq m/(4s)$, this quantity is bounded from above by $m/2$. The rest of the proof is similar to that of Lemma 2. $\square$

Proof of Theorem 1 In view of Fact 1, Lemmas 1, 2 and 3 the size of a monotone circuit that computes CLIQUE(m, s) is at least

$$\min\left(\frac{m!}{2(m/2)^{r+1}(m-r-1)!}, \frac{m!}{(m/2)^{l+1}(m-l-1)!}\right),$$

which is bounded from below by

$$\min\left(\frac{(m-r)^{r+1}}{2(m/2)^{r+1}}\ ,\ \frac{(m-l)^{l+1}}{(m/2)^{l+1}}\right). \tag{4}$$

Since $5 \leq s \leq m/4$, we have $r \leq m/10$ and $l \leq m/10$. Hence $m - r \geq 9m/10$ and $m - l \geq 9m/10$ hold. Therefore, (4) is bounded from below by

$$\min(1.8^{r+1}/2\ ,\ 1.8^{l+1}).$$

Thus, noticing $r+1 \geq m/(4s)$ and $l+1 \geq \sqrt{s-1}/2$, the proof is completed. □

One might expect that the definitions of approximate operations in the proof of Theorem 1 can be simplified by replacing "size" with "length" which is the number of variables appearing in terms and clauses. However, it turns out that such a simplification yield a weaker lower bound although it is still exponential.

Consider that we define the approximate operations based on "length" with suitable choices of l and r and apply the same argument as the proof of Theorem 1. By the same counting argument as in the proof of Lemmas 2 and 3, the upper bound of the number of choices corresponds to (2) in the proof of Lemma 2 is

$$\max_k\{km(s-1) + \sqrt{(l-k)(2m^2/(s-1))}\}, \tag{5}$$

and corresponds to (3) in the proof of Lemma 3 is

$$\max_k\{k(s-1) + \sqrt{(r-k)s(s-1)}\}. \tag{6}$$

It is easy to see that if $r = \lfloor m/(4s) \rfloor$ and $l = \lfloor (s-1)/8 \rfloor$, then the quantities of (5) and (6) are both smaller than $m/2$. It is also easy to see that the Lemma 1 holds for such l. However, Lemmas 2 and 3 may not hold in this time. For example, in the proof of Lemma 2, the condition that there exist more than r vertices assigned pairs of integers in the procedure described in the proof is not necessary condition to satisfy $f^D(x) = 0$ and $f^C(x) = 1$. Because if $r'(r'-1)/2 > r$, then there is a clause with more than r variables corresponding to the assignment that only r' vertices assigned pairs of integers. So we must replace the variable r in the statement of Lemma 2 with r' such that $r'(r'-1)/2 \leq r$. By a similar reason, we also have to replace the variable l in the statement of Lemma 3 with l' such that $l'(l'-1)/2 \leq l$. By using these lemmas and the similar argument as in the proof of Theorem 1, we can obtain the $\exp(\Omega(\min(r', l'))) = \exp(\Omega(\min(\sqrt{m/s}, \sqrt{s})))$ lower bound for the size of a monotone circuit that computes CLIQUE(m, s). This lower bound is an exponential in m for a suitable choice of s, however, is slightly weaker than the $\exp(\Omega(\min(m/s, \sqrt{s})))$ lower bound that Theorem 1 says.

We now consider another problem to show that our CNF and DNF based approximators make lower bound proofs simple.

Let $n = q^2$ and $x = \{x_{i,j} \mid 1 \le i, j \le q\}$, where q is a prime. Let $G(x)$ be the bipartite graph on $V = \{v_1, \ldots, v_q\}$ and $W = \{w_1, \ldots, w_q\}$ with edge set $\{(v_i, w_j) \mid x_{i,j} = 1\}$. Function POLY$(q, s)(x)$ of q^2 variables $\{x_{i,j}\}$ is defined as POLY$(q, s)(x) = 1$ if and only if there is a polynomial p over the field Z_q of degree at most $s - 1$ such that $G(x)$ includes all edges $(v_i, w_{p(i)})$ for $1 \le i \le q$.

The next theorem gives the same lower bound as the one due to Alon and Boppana[AB].

Theorem 2 [AB] If $s \le (1/2)\sqrt{q/\ln q}$, then any monotone circuit that computes the function POLY(q, s) contains $q^{\Omega(s)}$ gates. □

To prove Theorem 2, we need a few lemmas. A graph is called *good* if there exists a polynomial p of degree at most $s - 1$ such that the set of edges of the graph is given as $\{(v_i, w_{p(i)}) \mid 1 \le i \le q\}$. Notice that the number of good graphs is q^s. For $0 \le \epsilon \le 1/2$, let NG$_\epsilon$ be the probability distribution on bipartite graphs with each edge appearing independently with probability $1 - \epsilon$. We choose $\epsilon = (2s \ln q)/q$. It is easy to see that

$$\begin{aligned} \Pr_{x \in NG_\epsilon}[\mathrm{POLY}(q, s)(x) = 1] &\le q^s (1 - \epsilon)^q \\ &\le q^s e^{-\epsilon q} \\ &= 1/q^s \le 1/4. \end{aligned} \quad (7)$$

So x chosen according to NG$_\epsilon$ becomes a negative graph with probability at least 3/4. These graphs serve as the bad graphs in the previous case.

Instead of defining approximators in terms of the size of terms and clauses, we define approximators this time in terms of the length of terms and clauses, i.e., the number of variables appearing in terms and clauses. Approximate operations of $\bar{\vee}$ gate and $\bar{\wedge}$ gate are defined exactly the same way as the previous case except that "size" is replaced with "length". As in the previous case, approximators are defined in bottom-up fashion, taking the approximator corresponding to input variable x_i to be x_i itself. The parameters l and r are put this time as $l = s$ and $r = \lceil \sqrt{q \ln q} \rceil$.

To prove Theorem 2, we need Lemmas 4, 5 and 6, which correspond to Lemmas 1, 2 and 3, respectively.

We now proceed to the proof of the theorem.

Lemma 4 Let $\overline{f}$ be a function computed by an approximator circuit. Then

$\overline{f}$ is identically 0 or

$$\Pr_{x \in NG_\epsilon}[\mathrm{POLY}(q,s)(x) = 0 \text{ and } \overline{f}(x) = 1] \geq 1/4.$$

Proof If $\overline{f}$ is identically 0, then the first conclusion holds. If not, then there exists a term t of length at most l such that $\overline{f} \geq t$. Recall that the output gate is assumed to be an $\wedge$ gate, and hence $\overline{f}$ is given as a monotone DNF formula. Since $\epsilon = (2s \ln q)/q$, $l = s$ and $s \leq (1/2)\sqrt{q/\ln q}$, we have $\epsilon l \leq 1/2$. Since $0 \leq \epsilon 1/2$, we therefore have

$$\begin{aligned} \Pr_{x \in NG_\epsilon}[\overline{f}(x) = 1] &\geq (1-\epsilon)^l \\ &\geq (1/4)^{\epsilon l} \\ &\geq (1/4)^{1/2} = 1/2. \end{aligned} \tag{8}$$

Thus the lemma follows from (8) and (7). □

Lemma 5 Let f_1^D and f_2^D be monotone DNF formulas such that the length of each term in them is at most l. Then

$$\Pr_{x \in NG_\epsilon}[(f_1^D \vee f_2^D)(x) = 0 \text{ and } (f_1^D \,\overline{\vee}\, f_2^D)(x) = 1] \leq (1/2)^r.$$

Proof Let f^D denote the monotone DNF formula $f_1^D \vee f_2^D$. We construct a decision tree computing f^D as follows. We start with a vertex associated with f^D, which becomes the root of the decision tree we shall construct. Take a term whose length is, say i, out of terms in f^D, and construct the path consisting of i vertices labeled with variables in the term and a leaf labeled 1. Furthermore, all the edges on the path is labeled with 1. Draw i edges out of all the vertices except the leaf together with distinct endpoints added and label these edges with 0. Each path from the root to one of the i vertices added specifies in the obvious way an assignment of 1 and 0 to the variables on the path except the endpoint of the path. Associate these endpoints with the functions obtained by substituting the Boolean values to the variables of f^D according to the corresponding assignments. Then repeat the procedure mentioned above with the vertices added until all the vertices are assigned with the constant functions.

Let $path_x$ denote the path from the root to a leaf specified in the obvious way by assignment x. Let (u,v) be an arbitrary edge labeled with 0 on $path_x$. Clearly, the probability that the $path_x$ doesn't pass any edge labeled with 0 after (u,v) on the condition that $path_x$ passes edge (u,v) is at least $(1-\epsilon)^l$. This is because $path_x$ has at most l consecutive edges labeled with 1. Therefore the probability that $path_x$ passes more that r edges labeled with 0 is at most

$(1-(1-\epsilon)^l)^r$. Clearly, if both $(f_1^D \vee f_2^D)(x) = 0$ and $(f_1^D \overline{\vee} f_2^D)(x) = 1$ hold, then $path_x$ passes more than r edges labeled with 0. So the probability mentioned in the lemma is bounded from above by $(1-(1-\epsilon)^l)^r$. From the inequality $(1-\epsilon)^l \geq 1/2$ in (8), this quantity is at most $(1-1/2)^r = (1/2)^r$. This completes the proof. □

Lemma 6 Let an $\wedge$ gate and an $\overline{\wedge}$ gate be given as input the same monotone CNF formulas such that the length of clauses in the formulas is equal to or less than r. Then the number of good graphs for which the $\wedge$ gate and the $\overline{\wedge}$ gate produce different outputs is at most r^l.

Proof The proof is similar to that of Lemma 2. Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same monotone CNF formulas, denoted f_1^C and f_2^C, such that the size of any clause in the formulas is equal to or less than r. Let $f_1^C \wedge f_2^C$ be denoted by f^C and let $c_1, \ldots, c_q$ be the complete list of clauses of f^C. Let f^D denote the DNF formula equivalent to f^C. Clearly, f^D is the Boolean sum of terms of distinct variables, each being chosen from c_1 up to c_p. function as that by f^C. Since $f_1^C \overline{\wedge} f_2^C$ is equal to the DNF formula obtained from f^D by taking away all the terms whose sizes exceed l and $l(=s)$ variables $x_{i,j}$ with distinct first indices i's specify a polynomial with degree $s-1$, hence a good graph, it is easily seen that the number of good graph x's such that $(f_1^C \wedge f_2^C)(x) = 1$ and $(f_1^C \overline{\wedge} f_2^C)(x) = 0$ is bounded from above by the number of ways of choosing a variable out of r variables (appearing in a clause) l times. Thus the number of such good graph is at most r^l, completing the proof. □

Proof of Theorem 2 In view of Lemmas 4, 5 and 6, it is concluded that the size of the monotone circuit computing POLY(q, s) is at least $\min((q/r)^l, (1/4)2^r)$. Thus, equations $l = s$ and $r = \lceil \sqrt{q \ln q} \rceil$ give the desired bound, completing the proof. □

4 Lower bounds for non-monotone circuit complexity based on negation-based approximators

For two vectors v and v' in $\{0,1\}^n$, we denote $v' \lneq v$ if $v \neq v'$ and $v'_i \leq v_i$, for any $1 \leq i \leq n$, where v_i denotes the ith component of v. A vector v is called a *minimal true vector* for a Boolean function f if $f(v) = 1$ and, for any $v' \lneq v$, $f(v') = 0$. Let $Min(f)$ be the set of all minimal true vectors of f. Similarly, when circuit C computes a function f, $Min(C)$ denotes $Min(f)$. Given a non-monotone circuit C with input literals $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$, let C_{ex} denote the monotone circuit obtained from C by simply replacing each negated input variable $\bar{x}_i$ in C with a new input variable y_i for $1 \leq i \leq n$ without making any further modification. In what follows C_{ex} will be called the monotone analogue of the original circuit C. Let the input vectors to circuits C and C_{ex} be denoted by $(x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n)$ and $(x_1, \ldots, x_n, y_1, \ldots, y_n)$, respectively. Obviously these circuits produce the same output values for inputs represented as $(x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n)$.

Suppose that a circuit C computes a function f. When an input vector is given, all the wires in C are assigned 0 or 1 in the obvious way. We call a wire assigned the value 1 by an *activated* wire. Given a positive vector, i.e., a vector that makes circuit C output 1, we have the collection of activated wires in the circuit C which contains at least one path from an input node to the output node. On the other hand, in the case of a negative vector, i.e., a vector that makes C output 0, such a collection of wires does not contain any path from an input node to the output node.

Now suppose that a positive vector x is given to the circuit C. In the non-monotone case, not only the wires connected to inputs x_i with $x_i = 1$, but also the ones connected to inputs $\bar{x}_i$ with $x_i = 0$, which do not appear in monotone circuits, are activated. Since the circuit C contains only AND and OR gates in its inside, the entire collection of activated wires increases monotonically as the collection of activated wires at the input level increases. Hence viewing the computation of a circuit like this, one might imagine that the more negated variables we are allowed to use, the more economically the collections of activated wires can reach the output node. These considerations lead us to the following definition, which formalizes a measure of the amount of negation used in the process of computation in a circuit.

Definition 1 Let C be a circuit computing a monotone function f on n variables and let C_{ex} be the monotone analogue of C. For $0 \leq k \leq n$, we define $Min_k(C)$ as follows.

$$Min_k(C) = \left\{ v \in Min(f) \;\middle|\; \begin{array}{c} \exists y \in \{0,1\}^n \text{ with } |y|_1 \leq k \\ v \circ y \in Min(C_{ex}) \end{array} \right\}.$$

For v and u in $\{0,1\}^n$, $|v|_1$ denotes the number of 1's among v, and $v \circ u$ denotes the concatenation of v and u. In other word, a minterm v in f is in $Min_k(C)$ when the minimum number of variables $y_1, \ldots, y_n$ appearing in the minterm of the function computed by C_{ex}, which corresponds to the minterm v, is at most k.

□

Note by the definition

$$Min_0(C) \subseteq Min_1(C) \subseteq \cdots \subseteq Min_n(C) = Min(f).$$

In particular, if C is monotone, then $Min_0(C) = Min(f)$ holds. We can think of the minimum i satisfying $Min_i(C) = Min(f)$ as a sort of measure indicating an amount of negation used in computing f. The condition $Min_{m^{c/2-\epsilon}}(C) = Min(\mathrm{CLIQUE}(m, m^c))$ in the theorem below can be thought of as saying that C uses a "small" amount of negation in the sense of Definition 1 because $m^{c/2-\epsilon}$ is much smaller than $n = m(m-1)/2$, namely the number of variables of CLIQUE. So roughly, the theorem says that if a non-monotone circuit computes the clique function using a small amount of negation, then the circuit has an exponential number of gates.

Theorem 3 Let c and ϵ be constants such that $0 < c \leq 2/3$ and $0 < \epsilon < c/2$. Suppose that a circuit C computes the function $\mathrm{CLIQUE}(m, m^c)$. If $Min_{m^{c/2-\epsilon}}(C) = Min(\mathrm{CLIQUE}(m, m^c))$ holds, then the size of C is $\exp(\Omega(m^{c/2}))$. $\square$

The proof of the theorem will be done by a generalized argument based on the approximation method.

Let f be the function $\mathrm{CLIQUE}(m, s)$, where $s = m^c$. Let C be a circuit computing f whose monotone analogue is denoted by C_{ex}. Without loss of generality, we assume that m is divisible by s. Note that C_{ex} is assumed to be layered with alternating AND and OR layers, i.e., every AND gate is connected to an OR gate and vice versa, and the output gate of C_{ex} is an AND gate. Let $n = \binom{m}{2}$ be the number of input variables of CLIQUE. A vector $v \circ y$ of length $2n$ is called a *good* vector if $v \in Min(f)$, $|y|_1 \leq k$ and $v \circ y \in Min(C_{ex})$, where k is an integer whose value will be determined later. By the definition of $Min(C_{ex})$, if $v \circ y$ is a good vector then $v \circ y' \notin Min(C_{ex})$ for any $y' \lneq y$. Let V be the set of m vertices of the graph associated with CLIQUE. A one to one mapping from V to the set $\{0, 1, \ldots, s-1\} \times \{1, \ldots, m/s\}$ is called a *bad* mapping. The definition of the bad mapping is the same to that for the monotone case except that we added the value 0 to the range of the first component of the bad mapping. We consider a bad mapping ϕ as the graph that have an edge between u and v whenever the first elements of $\phi(u)$ and $\phi(v)$ are different and both of them are not 0. For each bad mapping ϕ, the bad vector associated with ϕ is the vector $u \circ \bar{u}$, where $\bar{u} = (\bar{u_1}, \bar{u_2}, \ldots, \bar{u_n})$. Note that every good vector makes the circuit C_{ex} output 1, whereas every bad vector makes the circuit C_{ex} output 0. The following fact is obvious.

Fact 2 There are at least $|Min_k(C_{ex})|$ good vectors and $m!$ bad mappings. $\square$

We are now ready to define approximators as well as approximate operations. Put $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$. Let $\mathcal{N}(k)$ denote the collection of subsets of Y of size at most k. For a set of variables $L \subseteq X$, let $\lfloor L \rfloor$ denote the set of all endpoints of the edges corresponding to variables in L. For $W \subseteq V$, the set of variables whose both endpoints are in W is denoted by $\lceil W \rceil$. $\lceil W \rceil$ can be thought of as the subset of X which corresponds to the clique on W. As in the previous section, the *size* of a term or a clause t is the cardinality of the endpoint set of variables in t. In what follows, for a set of literals L, the product of all literals in L is denoted by $\wedge L$.

An approximator is defined to be a set of monotone functions denoted $\{f_Y\}_{Y \in \mathcal{N}(k)}$, where all of f_Y's are either monotone DNF formulas on X or monotone CNF formulas on X. We interpret the approximator $\{f_Y\}_{Y \in \mathcal{N}(k)}$ to represent the function written as $\vee_{Y \in \mathcal{N}(k)} (\wedge Y \cdot f_Y)$.

$\overline{\vee}$: Let $\{f^D_{1,Y}\}_{Y \in \mathcal{N}(k)}$ and $\{f^D_{2,Y}\}_{Y \in \mathcal{N}(k)}$ be two approximators, each being sets of monotone DNF formulas. The approximate OR of these approximators, denoted $\{f^D_{1,Y}\}_{Y \in \mathcal{N}(k)} \overline{\vee} \{f^D_{2,Y}\}_{Y \in \mathcal{N}(k)}$, is defined as the set of monotone CNF formulas, denoted $\{f^C_Y\}_{Y \in \mathcal{N}(k)}$ where, for each $Y \in \mathcal{N}(k)$, f^C_Y is the monotone CNF formula obtained by transforming $f^D_{1,Y} \vee f^D_{2,Y}$ into the equivalent monotone CNF formula and then taking away all the clauses whose size exceed r.

$\overline{\wedge}$: Let $\{f^C_{1,Y}\}_{Y \in \mathcal{N}(k)}$ and $\{f^C_{2,Y}\}_{Y \in \mathcal{N}(k)}$ be two approximators, each being sets of monotone CNF formulas. The approximate AND of these approximators, denoted $\{f^C_{1,Y}\}_{Y \in \mathcal{N}(k)} \overline{\wedge} \{f^C_{2,Y}\}_{Y \in \mathcal{N}(k)}$, is defined as the set of monotone DNF formulas, denoted $\{f^D_Y\}_{Y \in \mathcal{N}(k)}$, where, for each $Y \in \mathcal{N}(k)$, f^D_Y is defined as follows :

1. For each $Y_1, Y_2 \in \mathcal{N}(k)$, let $f^D_{Y_1,Y_2}$ be the DNF formula obtained from the CNF formula $f^C_{Y_1} \wedge f^C_{Y_2}$ by transforming into the equivalent monotone DNF formula and then taking away all the clauses whose size exceed l.
2. For each $Y \in \mathcal{N}(k)$, set f'^D_Y to
$$f'^D_Y = \bigvee_{Y_1, Y_2 : Y = Y_1 \cup Y_2} f^D_{Y_1,Y_2}.$$

3. For each $Y \in \mathcal{N}(k)$, let f_Y^D be the monotone DNF formula obtained by replacing each term $\wedge L$ of $f_Y'^D$ by the term $\wedge\lceil\lfloor L\rfloor\rceil$.

In what follows an approximator $\{f_Y\}_{Y \in \mathcal{N}(k)}$ will be simply denoted by $\{f_Y\}_Y$. Note that $(\{f_{1,Y}^D\}_Y \overline{\vee} \{f_{2,Y}^D\}_Y) \geq (\{f_{1,Y}^D\}_Y \vee \{f_{2,Y}^D\}_Y)$ holds for any approximators $\{f_{1,Y}^D\}_Y$ and $\{f_{2,Y}^D\}_Y$ and that $(\{f_{1,Y}^C\}_Y \overline{\wedge} \{f_{2,Y}^C\}_Y) \leq (\{f_{1,Y}^C\}_Y \wedge \{f_{2,Y}^C\}_Y)$ holds for any approximators $\{f_{1,Y}^C\}_Y$ and $\{f_{2,Y}^C\}_Y$.

Now that the approximate operations are defined for non-monotone circuits, approximators computed by gates are determined from the bottom to the top. In order to get approximators determined in this way, we need to specify the approximator corresponding to input variables: The approximator corresponding to x_i is simply defined to be x_i itself.

The proof of Theorem 3 proceeds in a similar way to those in the previous section. We first prove that the total error in an approximator circuit is large (Lemma 7), and then prove that the local error at any single gate is small (Lemmas 8 and 9). Using these lemmas, the theorem follows from a simple calculation. Let $s = m^c$, $l = m^{c/2}/4$, $k = m^{c/2-\epsilon}$ and $r = m^{1-c}/4$.

Lemma 7 Let $\overline{f}$ be a function computed by an approximator circuit. Then $\overline{f}$ is identically 0 or the number of bad mappings for which $\overline{f}$ outputs 1 on the corresponding bad vector is at least $(1/2)(1/2s)^k$.

Proof

If $\overline{f}$ is identically 0, then the first conclusion of the lemma holds. If not, because of the assumption that the output gate of an approximator circuit is $\wedge$ gate, $\overline{f} \geq \wedge Y\lceil W\rceil$ for some $Y \in \mathcal{N}(k)$ and for some $W \subseteq V$ with $|W| \leq l$. Then a bad mapping ϕ satisfying the following conditions makes the approximator $\overline{f}$ take the value 1 : (i) For every variable $y \in Y$, at least one endpoint of y has value 0 on the first element of ϕ, and (ii) the first elements of $\phi(v)$ for $v \in W$ are all distinct and none of them are 0. The number of such bad mappings is at least

$$\begin{aligned}\binom{s-1}{l}\left(\frac{m}{s}\right)^l\binom{m/s}{k}(m-k-l)! &= m!\frac{\binom{s-1}{l}\left(\frac{m}{s}\right)^l\binom{m/s}{k}(m-k-l)!}{m!} \\ &\geq \prod_{i=0}^{l-1}\left(\frac{(m/s)(s-1-i)!}{m-i}\right)\prod_{i=0}^{k-1}\left(\frac{m/s-i}{m-k-i}\right) \\ &\geq \left(\frac{(m/s)(s-l)}{m-l+1}\right)^l\left(\frac{m/s-k+1}{m-l-k+1}\right)^k\end{aligned}$$

$$
\begin{aligned}
&\geq \left(\frac{(m/s)(s-l)}{m}\right)^l \left(\frac{m/s-k+1}{m-l-k+1}\right)^k \\
&\geq \left(1-\frac{l}{sm}\right)^l \left(\frac{m/s-k+1}{m-l-k+1}\right)^k \\
&\geq \left(1-\frac{1}{m}\right)^l \left(\frac{m/s-k+1}{m-l-k+1}\right)^k \geq \frac{1}{2}\left(\frac{1}{2s}\right)^k
\end{aligned}
$$

□

Lemma 8 Suppose that an $\vee$ gate and an $\overline{\vee}$ gate are given as input the same pair of approximators. Then the number of bad mappings for which the $\vee$ and the $\overline{\vee}$ gates produce different outputs on the corresponding bad vectors is at most

$$m^{2k}(m/2)^{r+1}(m-r-1)!$$

Proof Suppose that an $\vee$ gate and an $\overline{\vee}$ gate are given as input the same pair of approximators, denoted $\{f^D_{1,Y}\}_Y$ and $\{f^D_{2,Y}\}_Y$, respectively. Let $\{f^D_{1,Y}\}_Y\overline{\vee}\{f^D_{2,Y}\}_Y$ be denoted by $\{f^C_Y\}_Y$.

Let $Y \in \mathcal{N}(k)$ be fixed arbitrary. We first estimate the number of bad mappings for which $f^D_{1,Y}\vee f^D_{2,Y}$ outputs 0 and f^C_Y outputs 1 on the corresponding bad vectors by following the arguments in the proof of Lemma 2. The condition $l \leq \sqrt{s-1}/2$ in the proof of Lemma 2 is satisfied. Thus we can estimate the number of such bad mappings by using the same argument to the proof of the lemma except for replacing all the formulas $(s-1)$ in that proof by s. This modification is needed since the size of the ranges of the first components of the bad mappings is changed from $s-1$ to s. Thus we can verity that, for each $Y \in \mathcal{N}(k)$, the number of bad mappings for which $f^D_{1,Y}\vee f^D_{2,Y}$ outputs 0 and f^C_Y outputs 1 on the corresponding bad vectors is at most $(m/2)^{r+1}(m-r-1)!$. Thus, since the number of elements in $|\mathcal{N}(k)|$ is at most m^{2k}, we obtain the desired bound. □

Lemma 9 Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same approximators. Then the number of good vectors for which the $\wedge$ and the $\overline{\wedge}$ gates produce different outputs is at most

$$\frac{m^{4k}(m/2)^{l+1}(m-l-1)!}{s!(m-s)!}.$$

Proof Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same approximators, denoted $\{f^C_{1,Y}\}_Y$ and $\{f^C_{2,Y}\}_Y$, respectively. Let $\{f^D_Y\}_Y = \{f^C_{1,Y}\}_Y\overline{\wedge}\{f^D_{2,C}\}_Y$. It is easy to observe that steps 2 and 3 in the definition of the approximator $\overline{\wedge}$ introduce no error on good vectors. Thus we only have to estimate the amount of error on good vectors introduced by replacing the product of two CNF formulas $f^C_{1,Y_1} \wedge f^C_{2,Y_2}$ by a DNF formula $f^D_{Y_1,Y_2}$ for each $Y_1, Y_2 \in \mathcal{N}(k)$.

Let $Y_1, Y_2 \in \mathcal{N}(k)$ be arbitrary fixed. We estimate the number of good vectors for which $f^C_{1,Y_1} \wedge f^C_{2,Y_2}$ outputs 1 and $f^D_{Y_1,Y_2}$ outputs 0 by following the arguments in the proof of Lemma 2. Note that the condition $r \leq m/(4s)$ in the proof of Lemma 2 holds. Thus we can verify that the number of such good vectors is at most $(m/2)^{l+1}(m-l-1)!/\{(s!(m-s)!\}$ by the same argument to the proof of Lemma 2. Thus, since the number of choices of Y_1 and Y_2 is at most $|\mathcal{N}(k)|^2 \leq m^{4k}$, we obtain the desired bound. $\square$

We now proceed to the proof of Theorem 3. Instead of proving Theorem 3, we prove Theorem 4 which claims a stronger statement than Theorem 3. Intuitively, Theorem 4 says that the condition "$Min_k(C)$ is identical to $Min(\text{CLIQUE}(m,s))$" which appears in Theorem 3 can be replaced by the somewhat weaker condition "the cardinality of $Min_k(C)$ is not so small".

Theorem 4 Let c and ϵ be constants such that $0 < c \leq 2/3$ and $0 < \epsilon < c/2$. Suppose that a circuit C computes the function $\text{CLIQUE}(m, m^c)$ and suppose that $h = h(m) = o(m^{c/2-\epsilon})$. If

$$|Min_{m^{c/2-\epsilon}}(C)| > \frac{1}{m^h}|Min(\text{CLIQUE}(m, m^c))|$$

holds, then the size of C is $\exp(\Omega(m^{c/2}))$.

Proof Recall that $s = m^c$, $l = m^{c/2}/4$, $k = m^{c/2-\epsilon}$ and $r = m^{1-c}/4$. Let C be a circuit which computes the function $\text{CLIQUE}(m,s)$ and let $\overline{f}$ be the function computed by the approximator circuit of the monotone analogue of the circuit C. Suppose that $h = h(m) = o(m^c)$ and $|Min_k(C)| > \frac{1}{m^h}|Min_k(\text{CLIQUE}(m,s))| = \frac{1}{m^h}\frac{m!}{s!(m-s)!}$. By Fact 2, Lemma 8 and 9, the size of the circuit C is at least

$$\min\left(\frac{m!}{m^h m^{4k}(m/2)^{l+1}(m-l-1)!}, \frac{m!}{2(2s)^k m^{2k}(m/2)^{r+1}(m-r-1)!}\right)$$

This is lower bounded by

$$\begin{aligned}\min\left(\frac{(m-l)^{l+1}}{m^{5k}(m/2)^{l+1}}, \frac{(m-r)^{r+1}}{m^{3k}(m/2)^{r+1}}\right) &= \frac{1}{m^{5k}}\exp(\Omega(\min(l,r))) \\ &= \exp(\Omega(m^{c/2} - 5k\log m)) = \exp(\Omega(m^{c/2})).\end{aligned}$$

The second equality holds since $1 - c \geq c/2$ if $c \leq 2/3$. This completes the proof. $\square$

Before closing this section we give some more words on how our results relate to the Razborov's results[RR] pointing out the limitations of the approximation method. His result says that $\omega(n^2)$ lower bounds for non-monotone circuits could not be obtained as long as we follow the approximation method. Our results which establish exponential lower bounds for non-monotone circuits apparently conflicts with the results due to Razborov. From these conflicting result we may conclude that the amount of negations in the circuits we deal with in this paper is too small to make Razborov's arguments valid. It is an interesting open problem to show a bound on the amount of negation beyond which we have to go to make Razborov's arguments valid.

5 Acknowledgement

We would like to thank anonymous referees for making the many suggestions for improving the presentation of the paper.

References

[AB] N. Alon and R. B. Boppana, *The Monotone Circuit Complexity of Boolean Functions*, Combinatorica, Vol. 7, No. 1 (1987), pp. 1–22.

[BU] C. Berg and S. Ulfberg, *Symmetric Approximation Arguments for Monotone Lower Bounds without Sunflowers*, Comput. Complexity, Vol. 8, No. 1 (1999), pp. 1–20.

[H] A. Haken, *Counting Bottlenecks to Show Monotone $P \neq NP$*, Proc. 36th FOCS (1995), pp. 36–40.

[HR] D. Harnik and R. Raz, *Higher Lower Bounds on Montone Size*, Proc. 32nd STOC (2000), pp. 378–387.

[J] S. Jukna, *Finite Limits and Monotone Computations over the Reals*, Proc. 12th Conf. Comput. Complexity (1997), pp. 302–313.

[K] M. Karchmer, *On Proving Lower Bounds for Circuit Size*, Proc. 8th Structure in Complexity Theory (1993), pp. 112–118. also *Feasible Mathematics II* (1995), pp. 245–255.

[NM] K. Nakayama and A. Maruoka, *Loop Circuits and Their Relation to Razborov's Approximation Model*, Inform. and Comput., Vol. 119, No. 2 (1995), pp. 154–159.

[R1] A. A. Razborov, *Lower Bounds on the Monotone Complexity of Some*

Boolean Functions, Sov. Math. Doklady, Vol. 31 (1985), pp. 354–357.

[R2] A. A. Razborov, *On the Method of Approximations*, Proc. 21st STOC (1989), pp. 167–176.

[RR] A. A. Razborov and S. Rudich, *Natural Proofs*, J. Comput. System Sci., Vol. 55, No. 1 (1997), pp. 24–35. (Preliminary versions are in Proc. 26th STOC (1994), pp. 204–213.)

[ST] J. Simon and S.C. Tsai, *A Note on the Bottleneck Countint Argument*, Proc. 12th Conf. on Comput. Complexity (1997), pp. 297–301.

[W] A. Wigderson, *The Fusion Method for Lower Bounds in Circuit Complexity*, Paul Erdős is Eighty (Volume 1), Keszthely, Hungary (1993), pp. 453–467.