Записки научных
семинаров ПОМИ
Том 475, 2018 г.

A. A. Lialina

ON THE COMPLEXITY OF UNIQUE CIRCUIT SAT

Abstract. We consider the Circuit SAT problem for a circuit with at most one satisfying assignment. We present an algorithm running in time $O(2^{\cdot 374589m})$, where m is the number of internal gates of the circuit. In order to make the exposition self-contained, we also describe the algorithm for the general case of Circuit SAT with running time $O(2^{\cdot 389667m})$ obtained by Savinov in [10].

§1. Introduction

The Boolean circuit satisfiability problem (Circuit SAT), along with its particular case of satisfiability of a Boolean formula in CNF (SAT), are among the central problems of theoretical computer science. Many upper bounds for various cases of SAT have been proven; however, no approaches are known for proving Circuit SAT upper bounds of the form c^n, where n is a number of variables and $c < 2$ is a constant.

The first nontrivial upper bounds for SAT have been proved using branching heuristics, which we also apply in our algorithm. This approach was widely used in "moderately exponential" time algorithms: Kullmann and Luckhardt proved the $O(2^{m/3})$ and $O(2^{l/9})$ bounds [5], where l is the number of literals and m is the number of clauses. Both these bounds were improved by Hirsch in [3] to $O(2^{\cdot 30897m})$ and $O(2^{\cdot 10299l})$. Later the second bound was improved to $O(2^{\cdot 0926l})$ in [13].

Despite the success of exact algorithms for SAT and especially k-SAT, where nontrivial upper bounds are known since [6] and [1], there are few works studying the (exponential) time complexity of Circuit SAT for more general circuit classes. In [9] Santhanam considers cn-size arbitrary depth formulas over the basis $U_2 = B_2 \backslash \{\oplus, \equiv\}$, where B_2 is the set of all binary Boolean functions. He shows the bound $|C|2^{(1-1/c^{O(1)})}$. Later Seto and Tamaki extend his result to the basis B_2 [11]. They get the bound $2^{(1-\mu_c)n}$ for formulas of size at most cn, where $\mu_c > 0$ is a constant only depending on c (roughly $\mu_c = 2^{-\Theta(c^3)}$). An exponential-time #SAT-algorithm over

Key words and phrases: unique circuit SAT, circuit SAT, branching heuristics.
This research is supported by Russian Science Foundation (project 18-71-10042).

U_2 and B_2 has been recently presented in [2], this algorithm works faster than 2^n for "small" circuits; however, no such algorithms for arbitrary large circuits are known.

Unique k-SAT is the variant of k-SAT problem where the input CNF formula has a unique or no satisfying assignment. Valiant and Vazirani [12] give a randomized polynomial-time reduction from SAT to its instances of with unique solutions, which demonstrates that faster algorithms for Unique SAT give faster algorithms for the general case. Although their reduction is not enough to give an improvement in the case of exponential-time algorithms, still it demonstrates that solving the unique case of SAT is also important. Some exponential-time algorithms for the case of unique solutions are known: for example, a randomized algorithm for Unique 3-SAT that runs in time $O(2^{.386n})$ was presented in [8], where n is the number of variables of the formula, and it is still much simpler than the treatment of the general case.

If the size of the circuit is relatively small compared to the number of variables, then one is naturally interested in upper bounds of the form c^m, where m is the size of the circuit (that is, the number of internal gates). In [7] Nurk presents a branching algorithm, which achieves an upper bound $O(2^{.4058m})$. Some of the circuit transformations we apply in our algorithm originate from [7], they are described in Section 3. Some of the branching rules originate from the master thesis of Savinov [10], we explain them in Section 4. In Section 5 we present Savinov's algorithm, which improves Nurk's bound to $O(2^{.389667m})$. The main result for Unique Circuit SAT is presented in Section 6.

§2. Preliminaries

A circuit is an acyclic directed graph, in which the incoming degree of every node is equal to two or zero. The nodes of incoming degree zero are called inputs or variables and are labeled by Boolean variables. We denote the number of variables by n. The internal nodes are called gates and are labeled by one of the sixteen Boolean functions $f : \{0,1\} \times \{0,1\} \to \{0,1\}$. The function in every gate G is applied to the values obtained in G's parents. One gate of outdegree zero is designated as the output. We will always identify the gate name with its output. The size of a circuit is the number of internal gates (denoted by m). We say that a Boolean circuit has a satisfying assignment if there exists a substitution of the inputs by constants that forces the circuit to output the value one.

Among the Boolean functions of two variables there are two constant functions and four functions that depends only on one of the two variables. We can get rid of such functions by substituting a constant (or the identity function or the negation) to the descendants of the corresponding gate. Hence, we can assume that a function in a gate can be represented in one of the two following ways: $f(x, y) = (a \oplus x)(b \oplus y) \oplus c$ for some constants $a, b, c \in \{0, 1\}$ or $f(x, y) = x \oplus y \oplus a$ for some constant $a \in \{0, 1\}$. In the former case, we say that a gate which computes such function is of $\wedge$-type. In the latter case, the corresponding gate is called a $\oplus$-type gate. We will say that a circuit C can be simplified if it is possible to construct a smaller circuit C' in polynomial time, the new circuit satisfying the following conditions:

- The number of satisfying assignments of C' does not exceed the number of satisfying assignments of C.
- C' does not have a satisfying assignment only if C does not have any satisfying assignments.
- We can construct a satisfying assignment for C in polynomial time, knowing any satisfying assignment for C'.

Algorithms presented in this paper are based on the branching heuristic. The idea of this method is to choose a variable or gate and substitute it with 0 and 1. After such substitution we get two branches of the algorithm: two subcircuits (hopefully with much fewer gates). This substitution is called splitting and is denoted by $\{C[x = 0], C[x = 1]\}$, where x is a variable or gate chosen for substitution and C is the initial circuit. Finding a satisfying assignment for at least one of them solves the problem for the initial circuit as well.

For real numbers a, b, c, we say that splitting is not worse than $\{a, b\}$ if the given substitution splits our circuit of size c into two circuits: one that has the size at most $c - a$ and the other that has the size at most $c - b$.

Let A be the output gate. When we say that we substitute some gate G with a constant c it means that we add input x_G and reattach all outgoing wires of G to x_G. After that we add gate G' to our circuit computing the function $(G \equiv c) \wedge A$ and mark it as a new output gate (as we always have only one output gate, A is no longer an output gate). Then we substitute x_G to c. Note, that this substitution increases the number of gates by one.

We apply multiple splitting in some cases of the algorithms. That means that we substitute n variables or gates at once, which gives us 2^n branches. For example, we make splittings $C[x_1 = 0], C[x_1 = 1]$ and

$C[x_2 = 0], C[x_2 = 1]$. It is obvious that the result does not depend on the order of these splittings, so to estimate the multiple splitting we can assume that we first split by the variable x_1 and then by x_2. If the splitting by x_1 is not worse than $\{a_1, b_1\}$ and the splitting by x_2 is not worse than $\{a_2, b_2\}$ we say that the multiple splitting by x_1 and x_2 is not worse than $\{a_1, b_1\} * \{a_2, b_2\}$.

§3. Simplification rules and circuit transformations

In order to work with a circuit, we are going to formulate several transformation rules that we will apply at every step of the algorithm. Most of these rules are commonly used in circuit algorithms and their correctness is clear from the description, so we will not provide all formal proofs.

Rule (1)**.** *If G has no outgoing edges and is not marked as the output, then it can be removed.*

Rule (2)**.** *If G computes a constant operation a, we can remove G and "embed" this constant to the descendants of G changing the function computed in them.*

Rule (3)**.** *If there is a gate G that computes an operation depending only on one of its inputs, it is possible to remove G by reattaching its outgoing wires to that input. If G computes the negation, we also change the function $f(x, y)$ in its descendant to $f(\bar{x}, y)$.*

Lemma (4)**.** *If G (where G is a gate or an input of the circuit) has exactly two descendants G_1 and G_2 then either G_1 has a descendant different from G_2 (and vice versa) or we can simplify the circuit.*

Proof. Assume that G_1 is the only descendant of G_2, and denote the other parent of G_2 by H. The result computed in G_2 affects only gate G_1, which computes a function of G and H. Hence we can replace G_1, G_2 with a single gate computing this function. □

Lemma (5)**.** *If the input x feeds exactly one gate G, and G has type $\oplus$, then the circuit can be simplified.*

Proof. The input x affects only gate G and we can obtain any value in G regardless of the other parent of G. Therefore, we can reduce the number of gates by replacing G with a new input. □

Lemma (6)**.** *If there exists an input x with exactly two descendants of type $\oplus$, then the circuit can be transformed into another circuit with fewer such inputs and the same or smaller number of gates.*

Proof. This transformation was originally described in [7]. Denote two descendants of x by P_0 and R_0. Let us build two chains P and R. At the first step we add gate P_0 to P, after that we add a gate to P only if it is the only descendant of a previously added gate and has type $\oplus$. The same we do for R and R_0. We get two chains $P_0, \dots, P_p$ and $R_0, \dots, R_r$. Denote by $L_0, \dots, L_p$ and $T_0, \dots, T_r$ the parents of gates in P and R respectively.

Suppose that P and R do not intersect. Assume without loss of generality that there is no directed path from R_r to P_p. Clearly $P_p = x \oplus L_0 \oplus \dots \oplus L_p \oplus a$. Hence $x = P_p \oplus L_0 \oplus \dots \oplus L_p \oplus a$. Now we reverse all the edges in P, make P_p the new input and x the new gate. Before applying the transformation P_p could not have exactly one descendant of type $\oplus$ (otherwise we would extend the chain). Therefore, after applying this transformation P_p cannot have exactly two descendants of type $\oplus$. Hence we decreased the number of inputs from the lemma statement.

Suppose now that $P_k = R_m$ for some $0 \leqslant k \leqslant p$ and $0 \leqslant m \leqslant r$. It is easy to see that $P_k = x \oplus L_0 \oplus \dots \oplus L_{k-1} \oplus a \oplus x \oplus T_0 \oplus \dots \oplus T_{m-1} \oplus b$. Therefore, x can be eliminated because it does not affect the output. Hence we decreased the number of inputs with exactly two descendants of type $\oplus$. □

Suppose we have found a chain of gates S in our circuit. It starts with gate G that is the most remote from the output gate (the longest path between this gate and the output is the longest possible one). Suppose G has two parents x and y, which are both the inputs of the circuit. Then at every step we add a gate to S only if it is the only descendant of a previously added gate and has type $\oplus$.

Lemma (7)**.** *If there is a gate from S that has a parent, which is an internal gate and does not belong to S, then either the circuit can be immediately satisfied or the initial circuit C can be transformed into another circuit C' satisfying the following conditions:*

- *The set of parents of any chain in C constructed as described above consists on inputs and gates from this chain.*
- *The depth of C' is larger than the depth of C.*
- *The number of gates in C' does not exceed the number of gates in C.*

- *The number of inputs that have exactly two $\oplus$-type descendants in C' does not exceed the number of such inputs in C.*

Proof. Denote by G_l the first gate in S that has a parent A, which is an internal gate and does not belong to S. We swap A and y, as all gates in chain S have type $\oplus$ and we can always change the order of summation. This increases the depth of the circuit. We apply this transformation as long as there exist a chain dissatisfying the first condition. If the depth of the circuit becomes equal to its size, the circuit can be immediately satisfied. Obviously, the third and the fourth conditions hold after every such transformation. □

§4. Splitting cases

In this section we describe splittings which we apply if the circuit satisfies certain conditions. These splittings were suggested by Savinov in [10]. They are applied recursively in both algorithms.

Case (1)**.** The circuit contains a variable x feeding more than two gates. Then we make the splitting $\{C[x = 0], C[x = 1]\}$. In any case all x's descendants will be eliminated, therefore our splitting is not worse than $\{3, 3\}$.

Case (2)**.** The circuit contains a variable x feeding exactly two gates G_1 and G_2 of type $\wedge$.
We will call a descendant of x zero-type if it becomes constant by assigning x to 0 and one-type otherwise.
Note that if either G_1 or G_2 is the output gate, we can immediately satisfy the circuit or learn the value of x depending on the output type.

Case (2.1)**.** G_1 and G_2 are of different types.
Then we make the splitting $\{C[x = 0], C[x = 1]\}$ which is not worse than $\{3, 3\}$ because in any case we eliminate G_1, G_2, a descendant of zero-type gate (in case $x = 0$) or a descendant of one-type gate (in case $x = 1$).

Case (2.2)**.** G_1 and G_2 are of the same type.

Case (2.2.1)**.** The set consisting of G_1, G_2 and all their descendants contains at least four elements.

Then we make the splitting $\{C[x = 0], C[x = 1]\}$ which is not worse than $\{2, 4\}$. In one case we learn the value of G_1 and G_2, therefore we can eliminate them and their descendants. In the other case we eliminate just two gates G_1 and G_2.

Case (2.2.2)**.** G_1 and G_2 feed only one common gate H.
Then we make the splitting $\{C[x=0], C[x=1]\}$ which is not worse than $\{2,4\}$. In one case we learn the value of G_1, G_2 and also the value of H, therefore we eliminate them and at least one descendant of H. In the other case we eliminate just two gates G_1 and G_2.
Note that if H is the output gate, then in one case we determine the output of the circuit.

Case (3)**.** There is $\wedge$-gate G fed by two inputs x and y.

Case (3.1)**.** G is the only descendant of x and y.
Then by substituting some constant for the variables x and y, we can obtain any value in G. Therefore, we can replace G by a new input and eliminate x and y.

Case (3.2)**.** x and y both have exactly two descendants: H of type $\oplus$ and G.
Then we make the splitting $\{C[G=0], C[G=1]\}$ which is not worse than $\{3,3\}$. In one case we learn the values of x and y, therefore, we eliminate G, H and the descendant of G. In the other case we can obtain in H any value (by Lemma 5), thus H can be replaced by a new variable. It means that we can eliminate at least three gates: G, H and the descendant of G. Although we split by a gate we do not need to add any extra gates to our circuit because parents of G do not affect any gates after this simplification.

Case (3.3)**.** x and y together have two different descendants (except G) of type $\oplus$.

Then we make the splitting $\{C[x=0], C[x=1]\}$ which is not worse than $\{2,4\}$. In one case we eliminate two descendants of x, a descendant of G and $\oplus$-type descendant of y (by Lemma 5). In the other case we eliminate just two descendants of x.

Case (3.4)**.** x has two descendants: H of type $\oplus$ and G; y has outdegree 1.
We make the splitting $\{C[G=0], C[G=1]\}$ which is not worse than $\{3,3\}$. In one case we learn the values of x and y, therefore, we eliminate G, H and the descendant of G. In the other case we can assume that the value of G depends only on y, therefore, we eliminate H, G and the descendant of G. We do not add any extra gates to the output similarly to the Case 3.2.

Case (4)**.** The circuit contains a variable x feeding two gates: G of type $\wedge$, A of type $\oplus$. Another parent of G denoted by H has outdegree 1.
Then we make the splitting $\{C[x=0], C[x=1]\}$ which is not worse than

$\{2, 4\}$. In one case we eliminate G and A. In the other case we learn the value of G, therefore, we eliminate G, A, H, and the descendant of G.

Case (5)**.** The circuit contains a variable x feeding two gates: G of type $\wedge$, A of type $\oplus$. Gate G has outdegree more than one.

Case (5.1)**.** G has at least two descendants different from A.
Then we make the splitting $\{C[x = 0], C[x = 1]\}$ which is not worse than $\{2, 4\}$. In one case we eliminate G and A. In the other case we learn the value of G, therefore, we eliminate G, A and at least two descendants of G.

Case (5.2)**.** G has two descendants, one of them is A.
Again we make the splitting $\{C[x = 0], C[x = 1]\}$ which is not worse than $\{2, 4\}$. In one case we eliminate G and A. In the other case G becomes trivialized, therefore, we eliminate G, A, another descendant of G and at least one descendant of A.

Case (6)**.** In this case we will build a chain S. It starts with gate G that is the most remote from the output gate (the longest path between this gate and the output is the longest possible one). We denote G's parents by x and y, they are both the inputs of the circuit, therefore, G is of type $\oplus$, otherwise Case 3 would occur. Then at every step we add a gate to S only if it is the only descendant of a previously added gate and has type $\oplus$.
The set of parents of all the gates in S consists on the inputs and gates from S by Lemma 7. We denote the gates in S by $G_1, \ldots, G_k$ and their input parents by $x_1, \ldots, x_k$, where $G_1 = G$ and $x_0 = y, x_1 = x$.

Case (6.1)**.** A descendant of the last gate in S marked as the output.

Case (6.1.1)**.** The output gate has type $\wedge$.
Depending on the output type, we can immediately satisfy the circuit or learn the value of the output's parents. This means that we can eliminate G_k (the last added gate in S). Since G_k is fed by x_k and $\bigoplus_{i=1}^{k-1} x_i + a$ we can substitute x_k with $\bigoplus_{i=1}^{k-1} x_i + b$.

Case (6.1.2)**.** The output gate has type $\oplus$.
Then the other parent of H is a variable, because otherwise we could apply Lemma 7. Therefore the output is equal to $x_0 \oplus \cdots \oplus x_k \oplus x_{k+1}$, where x_{k+1} is a parent of H. Hence the circuit has a satisfying assignment.

Case (6.2)**.** G_k has at least two descendants.
Gate G_k is fed by x_k and $\bigoplus_{i=1}^{k} x_i + a$. We make the splitting

$$\{C[G_k = 0], C[G_k = 1]\}$$

which is not worse than $\{3, 3\}$. In other words, we substitute x_k with $\bigoplus_{i=1}^{k-1} x_i + b$ which eliminate G_k and at least two its descendants.

Case (6.3)**.** G_k has one descendant H. It follows that H has type $\wedge$, otherwise we would include H to S.

Case (6.3.1)**.** H has at least two descendants.
We make the splitting $\{C[G_k = 0], C[G_k = 1]\}$ which is not worse than $\{2, 4\}$. In one case we eliminate H and G_k (G_k by substituting x_k with $\bigoplus_{i=1}^{k-1} x_i + b$). In the other case we learn the value of H, therefore, we eliminate H, G_k, and two descendants of H.

Case (6.3.2)**.** H has exactly one descendant. Parents of H are $G_k \in S$ and the circuit input z with exactly one descendant.
We make the splitting $\{C[H = 0], C[H = 1]\}$ which is not worse than $\{3, 3\}$. In one case we eliminate H, its descendant, and G_k (the latter, by substituting x_k with $\bigoplus_{i=1}^{k-1} x_i + b$). In the other case we eliminate H, its descendant, and all gates in S, because H computes an operation depending only on z.

Case (6.3.3)**.** H has exactly one descendant. The input z feeds H and A. Then the splitting $C[z = 0], C[z = 1]$ will work not worse than $\{2, 4\}$. In one case we eliminate A and H. In the other case we eliminate A, H, its descendant, and also G_k like in the previous case. *Note that* $G_k \neq A$ *by Lemma* 4.

Case (6.3.4)**.** H has exactly one descendant. Parents of H are $G_k \in S$ and gate F with exactly one descendant.
We make the splitting $\{C[G_k = 0], C[G_k = 1]\}$ which is not worse than $\{2, 4\}$. In one case we eliminate G_k and H. In the other case we eliminate G_k, H, its descendant and F.

§5. Circuit SAT algorithm

In this section we present an algorithm solving Circuit SAT in time $O(2^{.389667m})$. This algorithm was suggested by Savinov [10]. We present it for completeness as we use parts of it in our algorithm for Unique Circuit SAT in Section 6.

The algorithm gets circuit C as its input and outputs a satisfying assignment if it exists or says "No" if it does not.

Step (0)**.** If the size of the circuit is less than some constant c then we check all the possible assignments in order to find one that satisfies the circuit.

Step (1)**.** Simplify the circuit using Rules 1–3 and Lemmas 5–7 if possible.

Step (2)**.** Go through all the cases listed in a Section 4 and exhaustively apply the first case satisfying the current situation.

Step (3)**.** If none of the cases applies to our circuit, then there is a chain S consisting of $\oplus$-type gates $G_1, \ldots, G_k$. Let x_i be a circuit input feeding G_i. We denote by F_i the other descendant of x_i. Such gate F_i exists and has type $\wedge$ for every i, otherwise we could apply Lemma 5 or Lemma 6. Gate H is the descendant of the last gate in S, has type $\wedge$ and exactly one descendant T, otherwise we could apply Case 6.2, Case 6.3 or add it to S. We denote by A the other parent of H (which is not in S). We will also denote by L_i the unique descendant of F_i (it is the only descendant because otherwise Case 5 would occur).
Then the following is true:

- $F_i \neq F_j$ for $i \neq j$ or we would apply Case 3.
- $F_i \neq L_j$ or we would apply Case 4.
- $H \neq F_i$ for all i or we would apply Case 6.3.2
- $H \neq L_i$ for all i or we would apply Case 6.3.3

Now we should divide our strategy between the two following cases, we will explain the motivation for this later.

Case ($k \leqslant 4$)**.** In this case we make multiple splitting by $x_k, x_{k-1}, \ldots, x_0$. The splitting $C[x_i = 0], C[x_i = 1]$ by itself is no worse than $\{2, 3\}$, because in both of these cases we eliminate G_i, F_i and in one of these cases L_i. Furthermore, in any case we will still get a similar chain, just of length $i - 1$.

After we make the splittings for all x_i except x_0 all gates in chain S will be eliminated. Therefore we will be able to apply Case 2 which is not worse than $\{2, 4\}$. Hence, the whole multiple splitting is not worse than $\{2, 3\}^k * \{2, 4\}$.

Case ($k > 4$)**.** We make the splitting $\{C[A = 0], C[A = 1]\}$. In one case we eliminate two descendants of A (they exists because otherwise we would

apply Case 6.3.4). However, in the other case we eliminate two descendants of A, T and all the gates $G_1, \ldots, G_k$, because they do not affect the value of H anymore. Note that we should add one gate to the circuit output because we split by gate. Hence, the splitting is not worse than $\{1, k+2\}$.

Now to find the running time of the algorithm we use the method of analyzing recursive branching algorithms. Let us denote by $l(m)$ the maximum number of leaves in the tree of recursive calls of the described algorithm among all the circuits of size m. The splitting $\{a_1, a_2\}$ brings us to the following recurrence relation:

$$l(m) \leqslant l(m - a_1) + l(m - a_2).$$

To estimate $l(m)$ we use the method suggested by Kullmann and Luckhardt ([4, 5]). Let us construct an equation which corresponds to the recurrence relation above:

$$1/x^{a_1} + 1/x^{a_2} = 1.$$

It always has a single positive root. Then following [4] and [5] we get the bound $l(m) \leqslant poly(m)\tau^m$, where τ is the biggest positive root among all the splittings we make in our algorithm.

Now let us explain the motivation to divide our strategy between these two cases.

Theorem 1. *The number of leaves in the tree of recursive calls of the splitting* $\{2,3\}^k * \{2,4\}$ *increases with* k.

Proof. Denote $f(k,x) = (\frac{1}{x^2} + \frac{1}{x^3})^k(\frac{1}{x^2} + \frac{1}{x^4})$, $k > 0$. This expression for estimating multiple splitting is explained in [5]. Let $a(k)$ be the maximal root of the equation $f(k,x) = 1$. If $x > 1$ then $(\frac{1}{x^2} + \frac{1}{x^4}) < (\frac{1}{x^2} + \frac{1}{x^3})$, therefore, $(\frac{1}{a(k)^2} + \frac{1}{a(k)^2}) > 1$. Hence, $f(k+1, a(k)) > 1$. The function $f(k+1,x)$ is continuous and approaches 0 as x approaches infinity. Therefore, $a(k+1) > a(k)$. □

Theorem 2. *The maximum number of leaves in the tree of recursive calls of the splitting* $\{1, k+2\}$ *decreases with* k.

Proof. Denote $g(k,x) = \frac{1}{x} + \frac{1}{x^k}, k > 0$. Let $a(k)$ be the largest root of equation $g(k,x) = 1$. If $a > 1$ then $g(k+1, a(k)) < g(k, a(k))$. Function $g(k,x)$ decreases monotonely with $x > 1$. Therefore, $a(k+1) < a(k)$. □

Now it is easy to check that $k = 4$ is the best value to change the strategy at. The largest root we get solving the corresponding equation is

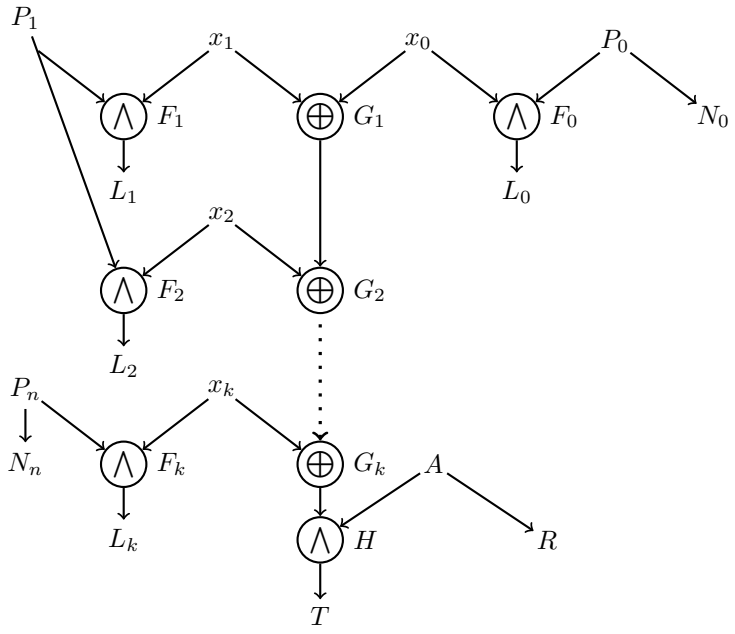

Figure 1. Unique Circuit SAT algorithm: Step 3.

1.31009. Hence, the algorithm decides Circuit-SAT in time $O(2^{.389667m})$, where m is the number of gates of the input circuit.

§6. Unique Circuit-SAT

In this section we present an algorithm that decides Circuit SAT for a circuit with at most one satisfying assignment.

Proposition (1)**.** *If the initial circuit has at most one satisfying assignment, then the number of satisfying assignments in every branch of the algorithm does not exceed one.*

Proof. Indeed when we make a splitting we get several subcircuits. And any subcircuit of Unique Circuit also has at most one satisfying assignment. □

Algorithm gets as input circuit C with at most one satisfying assignment and outputs the satisfying assignment if it exists, or the message that the given circuit cannot be satisfied in the opposite case.

Step (0)**.** If the size of the circuit is less than some constant c then we check all the possible assignments in order to find one that satisfies the circuit.

Step (1)**.** Simplify the circuit using Rules 1–3 and Lemmas 4–7 if possible.

Step (2)**.** Go through all the cases listed in a Section 4 and exhaustively apply the first splitting satisfying the current situation.

Step (3)**.** We will use all the definitions we gave in the algorithm for Circuit SAT. Additionally, let us denote by $P_1, \ldots, P_n$ all parents of F-gates that are different from x_i.

Note that some of F-gates can share a common parent, so n is not necessarily equal to k, where k is the size of chain S. We denote by N_i the other descendant of P_i (it exists because Case 4 does not occur). This construction is shown in Figure 1.

Again we have two different strategies depending on k. We will say that gate M kills $\wedge$-type gate N in some branch of the algorithm if M feeds N and takes a value according to this branching which uniquely defines the value of N.

Case ($k = 1$)**.** In this case we make a multiple splitting exactly like in Circuit SAT algorithm: $C[x_1 = 0], C[x_1 = 1]$ and $C[x_0 = 0], C[x_0 = 1]$. Hence we get a bound no worse than $\{2, 3\} * \{2, 4\}$.

Case ($k > 1$)**.** This case is the main difference between the two algorithms. We make the splitting $\{C[A = 0], C[A = 1]\}$ (again adding one extra gate to the output). In one case we eliminate two descendants of A. However, in the other case we get a lot more trivialized gates. Gate H now depends only on the value of A. Hence, we can eliminate at least one descendant of H (T), two descendants of A (H, R), all gates $G_1, \ldots, G_k$, because they do not affect the output anymore. It is only $k + 2$ gates, so no difference from the previous algorithm so far. Now let us consider two cases.

Case ($A \neq P_i$ for any i)**.** What happens if at least one of the gates $P_0, \ldots, P_n$ kills its descendant F_i? The circuit input x_i does not affect values of its descendants. Hence the number of satisfying assignments in the current branch either equals zero or exceeds one. In the case of Unique

Circuit SAT it means that in this branch we will not get a solution, so we can drop it. Therefore, we additionally get the information about all values of $P_0, \ldots, P_n$. Now we can eliminate $k+1$ F-gates, but the price for it is n gates that we must add to the output to make sure that we have chosen the values of P-gates correctly.

Now let us denote the number of P-gates that feed exactly one F-gate by s. For any of these gates there exists another descendant N_i, because Case 4 does not occur. (If for some i and j gate N_i coincide with gate N_j or another eliminated gate, we know the values of both N_i's parents. Hence we can eliminate at least one descendant of N_i. Therefore, we can assume that for every such P_i we can find its own N_i.) On the other hand, the number of gates P_i feeding more than two F-gates is equal to $n-s$. Hence, $k+1 \geqslant 2(n-s)+s$. Therefore, $\lfloor \frac{k+1}{2} \rfloor \geqslant n$. Consequently, the number of the eliminated gates (all F-gates and all N-gates) minus the number of the added ones is at least $k+1+s-n \geqslant k+1-n \geqslant \lceil \frac{k+1}{2} \rceil$. Therefore, the splitting is not worse than $\{1, k+2+\lceil \frac{k+1}{2} \rceil\}$.

Case $(A = P_i)$**.** Let A feed H and F_i. If H and F_i have the same type, i.e., they become trivial together when substituting some constant to A, then the value of A is fixed. Because if A kills both H and F_i, the input x_i does not affect value of its descendants. Hence the number of satisfying assignments in the current branch either equals zero or exceeds one. Therefore, we can assume that the "right" value of A is known and simplify the circuit without splitting (by eliminating at least two A's descendants and adding one more gate to the output).

If H and F_i have different types, then we learn the values of all P_i, because now they cannot kill their F-descendants or we get the same zero-or-more-than-one situation. Therefore, we can assume that the "right" values of all P_i's are known and simplify the circuit without splitting.

Theorem 3. *The algorithm described above decides Unique Circuit-SAT in time* $O(2^{.374589m})$*, where* m *is the number of gates in the input circuit.*

Proof. We split our strategy in the two cases using the same logic as in the Circuit SAT algorithm above. In the case $k = 1$ we make the splitting $\{2,3\} * \{2,4\}$, which gives the root 1.29647. In the case $k > 1$ we make the splitting $\{1, k+2+\lceil \frac{k+1}{2} \rceil\}$, in which the maximum number of leaves in the tree of recursive calls decreases by k. In the worst case of $k = 2$ it turns into $\{1,6\}$ with the root 1.2852. Then the largest root is 1.29647. Therefore, after applying the logarithm we get the upper bound $O(2^{.374589m})$. □

References

1. E. Ya. Dantsin, *Two tautologihood proof systems based on the split method.* — Zap. Nauchn. Sem. LOMI **105** (1981), 24–44.
2. A. Golovnev, A. S. Kulikov, A. V. Smal, S. Tamaki, *Circuit Size Lower Bounds and #SAT Upper Bounds Through a General Framework.* — In: Proceedings of 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016). LIPIcs 58, Schloss Dagstuhl – Leibniz–Zentrum fuer Informatik, pp. 45:1–45:16, (2016).
3. E. A. Hirsch, *New worst-case upper bounds for SAT,* J. Automated Reasoning **24**, No. 4 (2000), 397–420.
4. O. Kullmann, *New methods for 3-SAT decision and worst-case analysis.* — Theor. Computer Sci. **223** No. 1-2 (1999) 1–72.
5. O. Kullmann, H. Luckhardt, *Deciding propositional tautologies: Algorithms and their complexity.* Technical report, Fachbereich Mathematik, Johann Wolfgang Goethe Universität 1997.
6. B. Monien, E. Speckenmeyer, 3-*satisfability is testable in* $O(1.62^r)$ *steps.* Bericht Nr. 3/1979, Reihe Theoretische Informatik, Universität-Gesamthochschule-Paderborn, 1979.
7. S. Nurk, *An* $O(2^{.4058m})$ *upper bound for Circuit SAT* PDMI Preprint, 2009.
8. R. Paturi, P. Pudlák, M. E. Saks, F. Zane, *An improved exponential-time algorithm for k-SAT.* — J. ACM **52**, No. 3 (2005), 337–364.
9. R. Santhanam, *Fighting perebor: New and improved algorithms for formula and QBF satisfiability.* — In: Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS), (2010), 183–192.
10. S. V. Savinov, *Upper bound for Circuit SAT,* MSc Thesis. St. Petersburg Academic University RAS, 2014.
11. K. Seto, S. Tamaki, *A satisfiability algorithm and average-case hardness for formulas over the full binary basis.* — Comput. Complexity **22**, No. 2 (2013), 245–274.
12. L. G. Valiant, V. V. Vazirani, *NP is as easy as detecting unique solutions.* — Theor. Computer Sci. **47** (1986), 85–93.
13. M. Wahlström, *An algorithm for the SAT problem for formulae of linear length.* — In: Proceedings of the 13th Annual European Symposium on Algorithms, ESA 2005, volume 3669 of Lecture Notes in Computer Science (2005), 107–118.

С.-Петербургское отделение
Математического института
им. В. А. Стеклова РАН;
С.-Петербургский государственный университет,
С.-Петербург, Россия

E-mail: lyalina.albina@mail.ru

Поступило 5 декабря 2018 г.