About a surprising computer program of Matthias Müller

Mihai Prunescu *

Abstract

Matthias Müller published on the net a C++ code and a text describing the implemented algorithm. He claimed that the algorithm "maybe" solves the NP-complete problem 3-SAT in polynomial time. The program decided correctly so far the solvability for all instances that have been checked. This intriguing fact must be understood.
In order to achieve this goal, we introduce the graph of all possible 3-SAT clauses with edges connecting every two non-conflicting clauses. We prove that a 3-SAT instance I is satisfiable if and only if there is a maximal clique of the clause graph that does not intersect I.

Key Words: computational complexity, 3-SAT, clause table, polynomial time, P = NP.

A.M.S.-Classification: 03D15, 03D55, 03D80.

1 Introduction

Matthias Müller published on the net a C++ code and a text describing the implemented algorithm, see [1] (started in December 2013 and updated some times). He claimed that the algorithm "maybe" solves the NP-complete problem 3-SAT in polynomial time. All objects and a lot of the ideas below are suggested by Matthias' Müller program and companion text. However, objects and ideas are presented by him only intuitively, and proofs are reduced to arguments on some examples. I consider that his approach is more credible if it is described in a better founded mathematical language and with rigorous proofs. This is the goal of the present paper.

In order to achieve this goal, we introduce the graph of all possible 3-SAT instances with edges connecting every two non-conflicting clauses. We prove that a 3-SAT instance I is satisfiable if and only if there is a maximal clique of the clause graph that does not intersect I.

2 General definitions

The problem we are concerned here is 3-SAT with exactly three variables per clause, seen as decision problem. See [2].

Definition 2.1 An instance of 3-SAT is a formula written in conjunctive normal form with exactly three different variable in every disjunctive clause:

$$I = \bigwedge_{i=1}^{m} (\epsilon_{i1} x_{i1} \vee \epsilon_{i2} x_{i2} \vee \epsilon_{i3} x_{i3}),$$

where for all $i = 1, \ldots, m$, the variables x_{i1}, x_{i2}, x_{i3} are pairwise distinct, and every ϵ_{ij} means a negation or its absence. The instance is solvable if there is an assignment of the variables with truth-values 0 (false) or 1 (true), such that the whole expression evaluates 1 (true).

*Simion Stoilow Institute of Mathematics of the Romanian Academy, Research unit 5, P. O. Box 1-764, RO-014700 Bucharest, Romania. mihai.prunescu@imar.ro, mihai.prunescu@gmail.com

Definition 2.2 In Müller's notation every clause is done by a word written in the alphabet $\{-, 0, 1\}$ containing exactly three occurrences from the subset $\{0, 1\}$. The length of the word is always the total number of variables occurring in the 3-SAT instance. Zero occurring at position i means that the variable x_i occurs negated in the disjunctive clause. One occurring at position j means that the variable x_j occurs non-negated in the disjunctive clause. The letter occurring at position i $(1 \leq i \leq n)$ in clause C is denoted $C[i]$.

For example, $0----1--1$ means the clause $\neg x_1 \vee x_6 \vee x_9$ in an instance containing the variables $x_1, \ldots, x_9$. If no otherwise stated, n will always denote the total number of variables and will be a fixed number ≥ 3.

Definition 2.3 For a clause C, we call its *support* the three-element subset of $\{x_1, \ldots, x_n\}$ of the variables occurring in C. We call the set of all supports $\Sigma(n)$.

Definition 2.4 We fix a *total order* $<$ on the set $\Sigma(n)$ of all supports.

According to this order, the supports build a sequence $s_1, s_2, \ldots, s_{\binom{n}{3}}$.

Definition 2.5 Let $K(n)$ be the set of all possible 3-SAT clauses with n variables.

Every clause is completely defined by giving its support and a three-letter word $w \in \{0, 1\}^3$, which encode the information about which of the three variables is negated, and which is not negated. Consequently, $K(n)$ has $8\binom{n}{3}$ elements.

Definition 2.6 We fix a *total order* $<$ on the set $K(n)$ of all possible 3-SAT clauses. According to this order, the clauses build a sequence $C_0, C_1, \ldots, C_{8\binom{n}{3}-1}$. We say that the clause order is *compatible* with the support order, if the following condition is fulfilled:

For all $k \in \mathbb{N}$ with $0 \leq k \leq \binom{n}{3} - 1$, the set of all 8 clauses with support s_k occur in the clause sequence as the set $\{C_{8k}, C_{8k+1}, \ldots, C_{8k+7}\}$.

The following definition has been given by Matthias Müller:

Definition 2.7 Two clauses C and C' from $K(n)$ are said to *conflict* if in Müller's notation, there is an i with $1 \leq i \leq n$ such that ($C[i] = 0$ and $C'[i] = 1$) or ($C[i] = 1$ and $C'[i] = 0$). This means that at least one variable occurs in both clauses, once negated and once not negated.

Definition 2.8 On the set $K(n)$ we define the binary relation E by $E(C, C')$ if and only if C and C' *does not conflict.*

We observe that the relation E is symmetric and reflexive but not transitive. The undirected clause graph $(K(n), E)$ will be considered in the next sections.

3 Clause tables

In this section we introduce Müller's notion of a clause table. Differently from Müller, we make the definition independent from any given effective routine that generates all possible 3-SAT clauses. For the rest of the paper we fix an order of the set $\Sigma(n)$ of all supports and an order of the set $K(n)$ of all possible 3-SAT clauses, which is compatible with the order of $\Sigma(n)$. We recall that $\{0, 1\}^n$ is the set of all assignments of truth-values for the set of variables $\{x_1, x_2, \ldots, x_n\}$. This set has 2^n elements and can be alternatively seen as set of all $\{0, 1\}$-words of length n.

Definition 3.1 For any support $s \in \Sigma(n)$ we define the projection $\pi_s : \{0,1\}^n \to K(n)$ as follows: If $s = \{i_1, i_2, i_3\}$ and $\vec{a} \in \{0,1\}^n$, $\pi_s(\vec{a})$ is the word of length n containing a_{i_t} on position i_t (for $t = 1, 2, 3$) and "$-$" on all other positions. This word is interpreted as a 3-SAT clause according to Müller's notation. This set of projections can be seen as an application $\pi : \Sigma(n) \times \{0,1\}^n \to K(n)$ given by $\pi(s, \vec{a}) = \pi_s(\vec{a})$ for all $s \in \Sigma(n)$ and $\vec{a} \in \{0,1\}^n$.

Definition 3.2 The *clause table* is a matrix with $\binom{n}{3}$ columns and 2^n rows containing all possible 3-SAT clauses with n variables. The set of columns is indexed using the ordered set of supports $\Sigma(n)$. The set of rows is indexed using the set of binary words of length n, $\{0,1\}^n$. (We can see the set $\{0,1\}^n$ as lexicographically ordered, but in fact its ordering is not important.) The column s and the row $\vec{a}$ intersect in the element $\pi(s, \vec{a})$.

Lemma 3.3 *All clauses occurring in a row of the clause table are distinct and pairwise non-conflicting. Any two different rows of the clause table are different as sets of clauses. All clauses occurring in a column of the clause table are pairwise identical or conflicting. In fact, if $K(n)$ has an order that is compatible with $\Sigma(n)$, in the column indexed by $s_k \in \Sigma(n)$ occur exactly all the eight clauses $\{C_{8k}, \ldots, C_{8k+7}\}$ and every of them arises 2^{n-3} many times in the column.*

Proof: The clauses occurring in a row of the clause table are distinct because they have different supports. They are non-conflicting because they are projections of the same assignment $\vec{a}$ on different supports, so even if the supports have elements i in common, it is impossible that $C[i] \neq C'[i]$. Two different rows of the clause table are sets of projections from different assignments, and if $\vec{a} \neq \vec{b}$, there is some support s such that $\pi(s, \vec{a}) \neq \pi(s, \vec{b})$. All clauses occurring in a column of the clause table have the same support, so they are equal or differ on at least one element of the support, case in which they are different and conflicting. All clauses with support s occur in the column defined by s because all assignments are projected on this support. If we fix three booleans, there are 2^{n-3} possibilities to complete the assignment. □

Lemma 3.4 *For $k \geq 1$, if k many clauses are pairwise non-conflicting, then there is a row of the clause table such that all k clauses occur in that row.*

Proof: Denote the support of a clause C by support(C). Let $D_1, \ldots, D_k$ be the k clauses. We are looking for an assignment $\vec{a} \in \{0,1\}^n$ in order to prove that all clauses are on the row indexed by $\vec{a}$. In fact, $\vec{a}$ is uniquely determined on the set $S = \text{support}(D_1) \cup \text{support}(D_2) \cup \cdots \cup \text{support}(D_k)$, because for $i \in S$ one takes any j such that $i \in \text{support}(D_j)$ and takes $a_i = D_j[i]$. Clauses are not conflicting, so the definition is correct. If there are some values $i \in \{1, \ldots n\} \setminus S$, for these values one may take a_i arbitrarily. □

Lemma 3.5 *The clause graph $(K(n), E)$ has exactly 2^n many maximal cliques. Every maximal clique contains $\binom{n}{3}$ elements, which are the elements of some row in the clause table. For all $k \geq 0$, the 8 clauses with support s_k build a set $\{C_{8k}, \ldots, C_{8k+7}\}$ that intersects every maximal clique of $K(n)$ in exactly one vertex. Every vertex of $K(n)$ belongs simultaneously to exactly 2^{n-3} maximal cliques.*

Proof: As already seen in Lemma 3.4, a set of pairwise non-conflicting clauses can be always found as a subset of a row of the clause table. So the number of elements of a maximal clique is bounded by the length of a row of the clause table, and the number of maximal cliques is bounded by the number of clause table rows. On the other side, as already seen in Lemma 3.3, every clause table row is a clique. The last two statements are properties of the clause table (see Lemma 3.3) translated in terms of graphs. □

Recall that one can see an instance of 3-SAT as a set of clauses that must be simultaneously satisfied by some assignment.

Lemma 3.6 *An instance I of the 3-SAT problem is solvable if and only if there is some row R of the clause table such that $I \cap R = \emptyset$.*

Proof: For any assignment $\vec{a} \in \{0,1\}^n$, let $\neg\vec{a}$ be the assignment $(\neg a_1, \neg a_2, \ldots, \neg a_n)$.

Consider that $I \cap R = \emptyset$ and R is the row defined by an assignment $\vec{a} \in \{0,1\}^n$. Let $C \in I$ be some clause. We find C somewhere in the clause table. The column of C intersects the row R in a clause C' which is different of C, so by Lemma 3.3 this clause is conflicting with C but has the same support as C. Let s be their common support. There is an element $i \in s$ such that ($C[i] = 0$ and $C'[i] = 1$) or ($C[i] = 1$ and $C'[i] = 0$). In both cases $\neg\vec{a}$ is satisfying C in the literal containing the variable x_i. But C was chosen arbitrary, so $\neg\vec{a}$ satisfies I and I is solvable.

Now consider the case that for all rows R of the clause table, $I \cap R \neq \emptyset$. Consider some assignment $\vec{a}$ and its negation $\neg\vec{a}$. Let R be the clause table row corresponding to $\vec{a}$. As we have seen by proving the other direction, all clauses of I that do not belong to R are satisfied by $\neg\vec{a}$. However, no clause contained in R is satisfied by $\neg\vec{a}$, and there is at least a clause of I in R. So $\neg\vec{a}$ is not a solution of I. As $\vec{a}$ has been chosen arbitrarily, no assignment can be a solution of I, hence I is unsolvable. □

Lemma 3.7 *An instance I of the 3-SAT problem is solvable if and only if there is some maximal clique R of the clause graph $(K(n), E)$ such that $I \cap R = \emptyset$.*

Proof: This follows directly from Lemma 3.5 and Lemma 3.6.

Remark 3.8 The graph $(K(n), E)$ is a $\binom{n}{3}$-partite graph and the maximal cliques in this graph have $\binom{n}{3}$ elements, one for every partition subset. The partition subsets correspond with the set of all 3-SAT clause supports.

4 Algorithm

I present Müller's Algorithm in a modified form. I consider that the form below makes things easier to understand. Moreover, this modified form runs faster.

Bellow we presume to have already computed an array $K = K(n)$ containing all possible clauses $(C_0, C_1, \ldots, C_N)$, ordered compatibly with the fixed order of $\Sigma(n)$. The array K has length $N = 8\binom{n}{3}$. The input is a set of clauses I. Two boolean arrays $A, B : \{(i,j) \,|\, 0 \leq i < j < N\} \to \{true, false\}$ are used in the algorithm. The notation $[x]$ means the integer part of the rational number x. The pseudo-code notation "for $i = u$ to v" means that u is the first value in computation and v the last value.

Preparation:

compute K

initialize A and B

set $A \leftarrow true$ element-wise

Main Run:

for $i = 0$ to $N - 17$ do

    if $i \equiv 0 \mod 8$ set $B \leftarrow false$ element-wise

    if $C_i \notin I$ do

        for $j = 8[i/8] + 8$ to $N - 9$ do

if $\neg A(i,j)$ continue with next j
if $C_j \notin I \wedge E(C_i, C_j)$ do
for $k = 8[j/8] + 8$ to $N-1$ do
if $\neg A(i,k) \vee \neg A(j,k)$ continue with next k
if $C_k \notin I \wedge E(C_i, C_k) \wedge E(C_j, C_k)$ then set $B(j,k) \leftarrow true$

if $i \equiv 7 \mod 8$ set $A \leftarrow A \wedge B$ element-wise.

Conclusion:

for $i = N-16$ to $N-9$ do

for $j = N-8$ to $N-1$ do

if $A(i,j)$ return $true$

return $false$

Lemma 4.1 *Let n be ≥ 4. If the clause graph $(K(n), E)$ contains a maximal clique R such that $R \cap I = \emptyset$, then the algorithm returns* true.

Proof: During this proof, the following abuse of notation will be done: if i and j are the indexes of two clauses F and G, we write $A(F,G)$ or $B(F,G)$ instead of $A(i,j)$ or $B(i,j)$.

We also observe that if a boolean value $A(F,G)$ becomes false during the computation, it will be never again true, because its value can be changed only by boolean conjunction and false $\wedge\, x =$ false.

Because $n \geq 4$ there are nontrivial cliques in $K(n)$. (Observe that $K(3)$ consists of isolated vertexes because the clause table consists of a single column in this case.)

Suppose that the clause graph contains a maximal clique R such that $R \cap I = \emptyset$. By Lemma 3.5, R intersects every 8-clause set $\{C_{8k}, \ldots, C_{8k+7}\}$ in exactly one vertex F_k, and $R = \{F_0, \ldots, F_{\binom{n}{3}-1}\}$. When running the i-loop through $0 \leq i \leq 7$, the vertex F_0 is met. $F_0 \notin I$ and the array A is initialized true, so the j-loop and the k-loop work without restrictions. Along those loops it is checked that for all $0 < u < v \leq \binom{n}{3} - 1$, the clause pairs (F_0, F_u), (F_0, F_v) and (F_u, F_v) are edges in the clause graph, and because none of the clauses F_0, F_u, F_v belongs to I, $B(F_u, F_v)$ is set true. After proceeding with $A \leftarrow A \wedge B$, all boolean values $A(F_u, F_v)$ with $1 \leq u < v \leq \binom{n}{3}$ remain true.

When running the i-loop through $8 \leq i \leq 15$, the vertex F_1 is met, and $F_1 \notin I$. The boolean values $A(F_1, F_u)$, $A(F_1, F_v)$, $A(F_u, F_v)$ being true for $1 < u < v \leq \binom{n}{3} - 1$, the algorithm may check that the edges (F_1, F_u), (F_1, F_v) and finally (F_u, F_v) do exist in the clause graph and that none of the vertexes F_1, F_u, F_v are in I, and set $B(F_u, F_v)$ true. By setting $A \leftarrow A \wedge B$, the values $A(u,v)$ are reconfirmed true for $2 \leq u < v \leq \binom{n}{3} - 1$. By induction, after running the i-loop through $8k \leq i \leq 8k+7$, the boolean values $A(F_u, F_v)$ with $k+1 \leq u < v \leq \binom{n}{3} - 1$ are reconfirmed true. Finally, for $k = \binom{n}{3} - 3$, after running through $8\binom{n}{3} - 24 \leq i \leq 8\binom{n}{3} - 17$, $A(F_{\binom{n}{3}-2}, F_{\binom{n}{3}-1})$ is reconfirmed true, and the program returns *true*. □

Definition 4.2 Let (G, E) be a graph. We say that an edge AB is seen from a vertex C if edges CA and CB exist in the graph. We say that the edge AB is seen from a subset $S \subset G$ if and only if there is a vertex C in S such that AB is seen from C. This definition will be applied for S the set of all 8 clauses with given support.

We observe that the algorithm works in the following way:

Delete all edges AB such that $A \in I$ or $B \in I$.

For all supports S of 3-SAT clauses, excepting the last two supports, do:

Make the list of all edges that can be seen from S.

Delete all other edges in graph.

Delete the 8 vertices from S.

If there is an edge connecting two of the 16 remained vertices, return *true*. If not, return *false*.

The idea is quite natural in the following sense. Every maximal clique intersects a support in a one element set. If an edge cannot be seen from a support, it cannot belong to any maximal clique, so it can be deleted. Unhappily it is not clear so far, why the program should always give correct positive answers. (All negative answers are correct, but we cannot exclude false positive answers.) The algorithm is not good enough for finding maximal cliques in arbitrary k-partite graphs, but we don't know either if it does work or not for the special family of graphs ($K(n)$).

The algorithm practices at most

$$\binom{\binom{n}{3}}{3}$$

many edge checks, which is a polynomial of degree 9. An edge check can be done in logarithmic time, because we might need logarithmic time to read a name of variable like x_n. (Using Müller's notation, it could take linear time, but this would be less practical). Looking for clauses in a list I of length $O(n^3)$ can be done by binary search in time $O(3 \log n)$, because the list could have been ordered lexicographically before. So I appreciate the computation time to an order like $O(n^9 \log n)$. □

References

[1] **Matthias Müller**: *Polynomial SAT-solver.* http://vixra.org/author/matthias_mueller

[2] **Michael R. Garey and David S. Johnson**: *Computers and intractability: A guide to the theory of NP-completeness.* W. H. Freeman & Co., 1979.