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 k-SAT clauses with edges connecting every two non-conflicting clauses. We prove that a k-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 k-SAT instances with edges connecting every two non-conflicting clauses. We prove that a k-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 k-SAT with exactly k variables per clause, seen as decision problem. The most important case is 3-SAT, because $k = 3$ is the smallest value of k for which it is known that the corresponding problem is NP-complete. See [2].

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

$$I = \bigwedge_{i=1}^{m} (\epsilon_{i_1} x_{i_1} \vee \epsilon_{i_2} x_{i_2} \vee \cdots \vee \epsilon_{i_k} x_{i_k}),$$

where $k \geq 1$ and for all $i = 1, \ldots, m$, the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ are pairwise distinct, and every ϵ_{i_j} means a negation or its absence. In fact, it is useful to consider that $\epsilon_{i_j} \in \{0, 1\}$ such that

*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

$\epsilon_{i_j} = 0$ means negation. The set of all occurring variables is considered to be $\{x_1, \ldots, x_n\}$, where $n \geq k$ is part of the input. The number n will be considered the measure of the complexity of the instance, because the number of all possible clauses for fixed k is a polynomial in n. 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).

It is known that 1-SAT is trivially solvable in linear time, because the instance is solvable if and only if no variable occurs both negated and positive in the instance. 2-SAT is also solvable in linear time, see [3].

Definition 2.2 For a clause C, we call its *support* the k-element subset of $\{x_1, \ldots, x_n\}$ consisting of the variables occurring in C. We call the set of all supports $\Sigma(k, n)$.

Definition 2.3 For a clause C, we say that $C[i] = 0$ if the variable x_i occurs negated in C and that $C[i] = 1$ if the variable x_i occurs positive in C. If x_i does not occur in C, we write $C[i] = -$.

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

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

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

Every clause is completely defined by giving its support and a k-letter word $w \in \{0, 1\}^k$, that encodes the information about which of the k variables is negated, or positive. Consequently, $K(k, n)$ has $2^k \binom{n}{k}$ elements.

Definition 2.6 We fix a *total order* $<$ on the set $K(k, n)$ of all possible k-SAT clauses. According to this order, the clauses build a sequence $C_0, C_1, \ldots, C_{2^k\binom{n}{k}-1}$. We say that the clause order is *compatible* with the support order, if the following condition is fulfilled:

For all $t \in \mathbb{N}$ with $0 \leq t \leq \binom{n}{k} - 1$, the set of all 2^k clauses with support s_t occur in the clause sequence as the set $\{C_{2^k t}, C_{2^k t+1}, \ldots, C_{2^k t+2^k-1}\}$.

We call the set $\{C_{2^k t}, C_{2^k t+1}, \ldots, C_{2^k t+2^k-1}\}$, the clause type set corresponding to the given support.

The following definition has been given by Matthias Müller:

Definition 2.7 Two clauses C and C' are said to conflict if there is a variable x_i present in both clauses, once negated and once positive.

Definition 2.8 On the set $K(k, n)$ 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 $(K(k, n), E)$ 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 of any given effective routine that generates all possible k-SAT clauses. For the rest of the paper we fix an order of the set $\Sigma(k, n)$ of all supports and an order of the set $K(k, n)$ of all possible k-SAT clauses, which is compatible with the order of $\Sigma(k, n)$. 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 $s \in \Sigma(k,n)$ we define the projection $\pi_s : \{0,1\}^n \to K(k,n)$ as follows: If $s = \{i_1, i_2, \ldots, i_k\}$ and $\vec{a} \in \{0,1\}^n$, $\pi_s(\vec{a})$ is the clause

$$a_{i_1}x_{i_1} \vee a_{i_2}x_{i_2} \vee \ldots \vee a_{i_k}x_{i_k},$$

where $a_{i_j} = 0$ is interpreted as a negation. This set of projections can be seen as an application $\pi : \Sigma(k,n) \times \{0,1\}^n \to K(k,n)$ given by $\pi(s,\vec{a}) = \pi_s(\vec{a})$ for all $s \in \Sigma(k,n)$ and $\vec{a} \in \{0,1\}^n$.

Definition 3.2 The *clause table* is a matrix with $\binom{n}{k}$ columns and 2^n rows containing all possible k-SAT clauses with n variables. The set of columns is indexed using the ordered set of supports $\Sigma(k,n)$. 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 $K(k,n)$ has an order that is compatible with $\Sigma(k,n)$, in the column indexed by $s_t \in \Sigma(k,n)$ occur all 2^k many clauses $\{C_{2^k t}, \ldots, C_{2^k t + 2^k - 1}\}$ and every one of them arises 2^{n-k} 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 x_i in common, it is impossible that x_i occurs negated in C and positive in C', or vice-versa. 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 - in this case 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 k booleans, there are 2^{n-k} possibilities to complete the assignment. □

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

Proof: Denote the support of a clause C by $\text{support}(C)$. Let $D_1, \ldots, D_t$ be the t 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 $S = \text{support}(D_1) \cup \text{support}(D_2) \cup \cdots \cup \text{support}(D_t)$, 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 $i \in \{1, \ldots n\} \setminus S \neq \emptyset$, for these values one may choose a_i arbitrarily. □

Lemma 3.5 *The clause graph $(K(k,n), E)$ has exactly 2^n many maximal cliques. Every maximal clique contains $\binom{n}{k}$ elements, which are the elements of some row in the clause table. For all $t \geq 0$, the 2^k clauses with support s_t build a set $\{C_{2^k t}, \ldots, C_{2^k t + 2^k - 1}\}$ that intersects every maximal clique of $K(k,n)$ in exactly one vertex. Every vertex of $K(k,n)$ belongs simultaneously to exactly 2^{n-k} 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 k-SAT as a set of clauses that must be simultaneously satisfied by some assignment.

Lemma 3.6 *An instance I of the k-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 when 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 one 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, and so I is unsolvable. □

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

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

Remark 3.8 The graph $(K(k,n), E)$ is a $\binom{n}{k}$-partite graph and the maximal cliques in this graph have $\binom{n}{k}$ elements, one for every partition subset. The partition subsets correspond with the set of all k-SAT clause supports and every one contains exactly 2^k many clauses with the same support.

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.

Definition 4.1 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 to S, the set of all 2^k clauses with support $s \in \Sigma(k,n)$. We recall that such a set has been called a clause type set.

We observe that the algorithm works in the following way:

Consider $(K(k,n), E)$ and an order of edges compatible with the lexicographic order of $\Sigma(k,n)$.

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

For all supports $s \in \Sigma(k,n)$, excepting the last two supports, do:

> Make the list of all edges that can be seen from the corresponding clause type set S.
>
> Delete all other edges in graph.
>
> Delete the 2^k vertices in S.

If there is an edge connecting two of the 2^{k+1} remained vertices, return *true*. If not, return *false*.

The idea is quite natural in the following sense. Every maximal clique intersects a clause type set in a one element set. If an edge cannot be seen from a clause type set, 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) t-cliques in arbitrary t-partite graphs, but we don't know either if it does work or not for the special family of graphs $(K(k,n) \setminus I)$. Matthias Müller tried the program successfully for a lot of 3-SAT instances.

If $k = 3$, 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. 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 for 3-SAT to an order like $O(n^9 \log n)$.

Lemma 4.2 *Let n be $\geq k+1$. If the clause graph $(K(k,n), E)$ contains a maximal clique R such that $R \cap I = \emptyset$, then the algorithm returns* true.

Proof: Indeed, all edges in this maximal clique R are seen from the clause where the maximal clique intersects the first clause type set. After deleting this first clause type set, all edges of the maximal clique R, which are still in the graph, are seen from a vertex of the second clause type set, so they will not be deleted. And so on. □

5 A counterexample to order issues

Despite the fact that in Müller's argumentation the order of supports does not play any role, I will show here that his algorithm is sensitive to order issues.

Consider the following instance I of the problem 3-SAT. I has 20 clauses and contains 8 variables:

$$(\neg x_2 \vee \neg x_5 \vee \neg x_8) \wedge (\neg x_3 \vee x_7 \vee x_8) \wedge (x_3 \vee x_7 \vee x_8) \wedge (x_2 \vee \neg x_5 \vee \neg x_6) \wedge (x_4 \vee \neg x_6 \vee x_8) \wedge (x_4 \vee x_6 \vee x_8) \wedge (\neg x_3 \vee x_5 \vee x_6) \wedge$$
$$(x_2 \vee \neg x_4 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (\neg x_5 \vee x_6 \vee x_8) \wedge (x_5 \vee \neg x_6 \vee \neg x_8) \wedge (\neg x_3 \vee x_6 \vee \neg x_8) \wedge (\neg x_4 \vee \neg x_6 \vee \neg x_7) \wedge (x_2 \vee x_7 \vee \neg x_8) \wedge$$
$$(x_2 \vee x_6 \vee \neg x_8) \wedge (x_1 \vee \neg x_7 \vee \neg x_8) \wedge (x_1 \vee x_7 \vee \neg x_8) \wedge (\neg x_2 \vee x_6 \vee x_7) \wedge (x_3 \vee x_5 \vee x_8) \wedge (\neg x_2 \vee \neg x_4 \vee \neg x_7)$$

Lemma 5.1 *The instance I introduced above is unsolvable. There is an order A of the* 3-SAT *supports such that Müller's Algorithm returns a false* true *for this problem. For the lexicographic order of the supports and for another order B, Müller's Algorithm correctly returns* false.

The proof can be done only by dirrect computation. Let s_1, s_2, ..., s_{16} be the supports of all clauses in I, *in the order of their occurrence* in I, according to the list above. Consider the order A consisting of: all 40 supports for 8 variables that do not occur in I, put in an arbitrary order (e. g. the lexicographic one), followed by the 16 supports occurring in I, in the order in which they occur in I. If run over this order of supports, Müller's Algorithm returns a false *true*. This fact also answers another possible question: Is it important that all supports are present, and not only those which contain the clauses of the instance? The answer is yes. It is also important to notice the following lemma:

Lemma 5.2 *Consider an order of $\Sigma(3,n)$ such that the first m supports do not contain any clause from I. As long as the algorithm runs over those m supports, no edge connecting clauses of the remaining supports is deleted.*

Proof: Indeed, this does not happen by the first support, because all maximal cliques pass through this support, and all edges which connect vertexes from the remaining supports are visible from the first support. This happens exactly so during the second support and up to support m. □

So, running the algorithm only with supports intersecting I or running the algorithm with the order A produce the same false positive result.

The order B is constructed in a way opposite to which the order A has been constructed. First we write down the 16 supports of clauses occurring in I, and then all other 40 supports lexicographically ordered. This order returns the correct negative answer. Even if the 40 supports do not contain missing vertexes, there are lots of edges which are deleted when running over them. This is the effect of the edges deleted when running through the first 16 supports: some later edges are no more seen from some later supports. □

In general, in order to prevent that elements from I are too close to the end, and do not express their whole power, I recommend that one always build a support order like order B and runs the algorithm over this order. I also recommend that after every support, one checks the stop condition. If I is such that it contains elements in all supports, or in almost all supports and very close to the end, one can introduce a dummy variable more and consider I as an instance of $n+1$ variables. For this "new" I, construct order B and run Müller's Algorithm. The improved algorithm is presented below:

Order the set of supports $\Sigma(k,n)$ such that the supports containing clauses from I are put before.

Optional: Permute the supports in the decreasing order of the number of clauses from I contained in the corresponding clause type sets.

Optional: If at least $\binom{n}{k} - 2$ clause type sets contain clauses from I, consider I as instance in variables $\{x_1, \ldots, x_n, x_{n+1}\}$ and restart the algorithm.

Consider $(K(k,n), E)$ and an order of edges compatible with the order of $\Sigma(k,n)$.

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

For all supports $s \in \Sigma(k,n)$, excepting the last two supports, do:

Make the list of all edges that can be seen from the corresponding clause type set S.

Delete all other edges in graph.

Delete the 2^k vertices in S.

If there are no more edges between the last two clause type sets, stop and return *false*.

If there is an edge connecting the last two clause type sets, return *true*. If not, return *false*.

I believe that, for the problem 3-SAT, the algorithm improved this way always returns *false* if all maximal cliques intersect I. It looks to be so, that missing elements produce holes that become bigger and bigger along the former maximal cliques, if they have enough place to do this. Unhappily, I could not prove (or disprove) this so far, and the problem remains open.

6 Obsevations

This section contains some partial results.

Lemma 6.1 *Both algorithms given above correctly solve* 1-SAT.

Proof: A 1-SAT instance is unsolvable if and only if a variable x occurs both negated and positive in the instance. On the other hand, the algorithm deletes some edge if and only if both clauses x and $\neg x$ are absent in the clause type set of x. In this case, no further edge can be seen from x and all following edges are deleted. The algorithm stops and returns a right *false*. □

Observation 6.2 For all $k \geq 1$ is true that if an instance I of the problem k-SAT contains a whole clause type set (corresponding to some support), then I is unsolvable. Of course, from this clause type set one cannot see any further edge.

Definition 6.3 A 2-SAT clause is denoted AB, where A and B are literals, i. e. negated or positive variables. An edge of the graph $K(2, n)$ connecting clauses AB and CD is denoted $AB - CD$. A 3-SAT clause is denoted ABC, where again A, B and C are literals. An edge of the graph $K(3, n)$ connecting clauses ABC and DEF is denoted $ABC - DEF$.

Lemma 6.4 *When running both algorithms given above:*

If the algorithms are applied for 2-SAT*: If a clause AB is missing in a given clause type set, all edges $Au - Bv$ there are still in the graph will be deleted when the clause type set of AB is processed.*

If the algorithms are applied for 3-SAT*: If a clause ABC is missing in a given clause type set, all edges $Xuv - YZw$ there are still in the graph will be deleted when the clause type set of ABC is processed, where $\{A, B, C\} = \{X, Y, Z\}$.*

Proof: In both cases, the edges mentioned above can be seen from the clause type set of the clause AB (respectivlely, ABC) only from the corresponding clauses, because they conflict with all otherclauses in the given clause type set. □

If one checks in detail how the algorithms work, one sees that the edges mentioned in the Lemma 6.4 are only a small part of the edges deleted by the algorithms. Of course, there are a lot of edges which are no more seen from some clause type set, because at least one edge between each vertex in the clause type set and the given edge has been deleted before. One can introduce a notion of rank for the edges deleted by the algorithms, such that the edges mentioned in Lemma 6.4 are the deleted edges of rank one.

Definition 6.5 Only deleted edges get a rank.

Edges $C_1 - C_2$ which are deleted because $C_1 \in I$ or $C_2 \in I$, get rank 0.

Suppose that an edge $C_1 - C_2$ is deleted after proceeding a clause type set S.

Start with the rank value $r = 0$. For each vertex $C \in S$:

If the edge $C - C_1$ is still present and $C - C_2$ has rank a, replace r with $\max(r, a + 1)$.

If the edge $C - C_2$ is still present and $C - C_1$ has rank b, replace r with $\max(r, b + 1)$.

If edges $C - C_1$ and $C - C_2$ have ranks a and b, replace r with $\max(r, \min(a, b) + 1)$.

The last value of the rank reached during this procedure will be the rank of $C_1 - C_2$.

Experiments show that the rank ist most probably unbounded. In some 3-SAT instances with 14 variables, the author meat edges of rank 12. The fact that the rank is not bounded seems to increase the difficulty of the problem.

We finish with an illustrated example. The following instance consists of eight clauses in five variables. The instance is minimally unsolvable, in the sense that all proper subsets are solvable, and all five variables occur in the clauses.

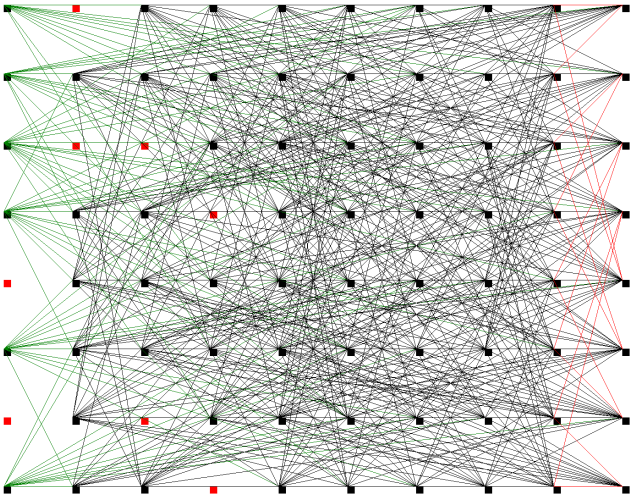

Figure 1: Edges seen from the first clause type set.

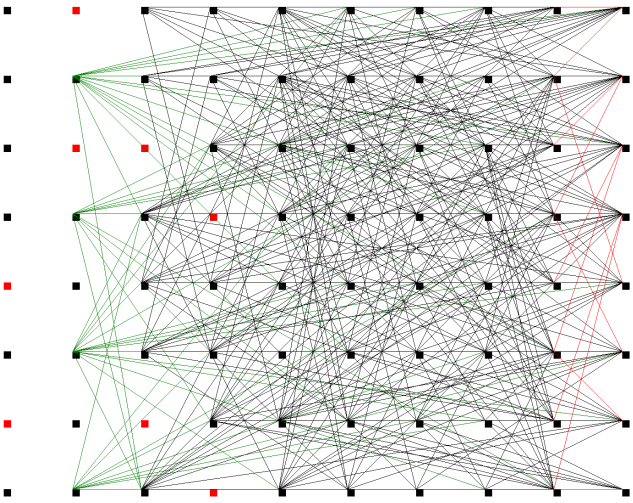

Figure 2: Edges seen from the second clause type set.

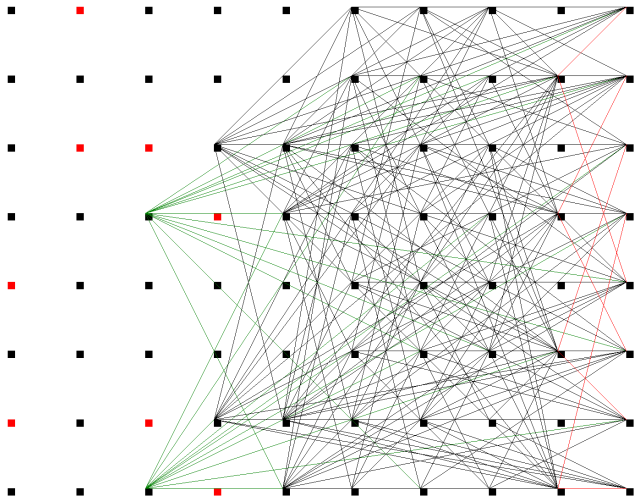

Figure 3: Edges seen from the third clause type set.

$$I = (\neg x_2 \vee \neg x_4 \vee x_5) \wedge (\neg x_2 \vee x_4 \vee x_5) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_5) \wedge (\neg x_2 \vee x_3 \vee \neg x_5) \wedge$$
$$\wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

The set of supports is ordered as follows: $(2,4,5)$, $(2,3,5)$, $(1,2,4)$, $(1,2,3)$, $(1,2,5)$, $(1,3,4)$, $(1,3,5)$, $(1,4,5)$, $(2,3,4)$, $(3,4,5)$. Only the first four supports contain clauses from I, the remaining six supports are ordered lexicographically. The images below represent the successive graphs produced by the second algorithm. The clause type sets corresponding to each support contain eight elements and are displayed vertically. The graph is 10-partite. Clauses from I are marked by red vertexes.

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.

[3] **Bengt Aspvall, Michael F. Plass, Robert E. Tarjan**: *A linear-time algorithm for testing the truth of certain quantified boolean formulas.* Information Processing Letters 8 (3), 121123, 1979.

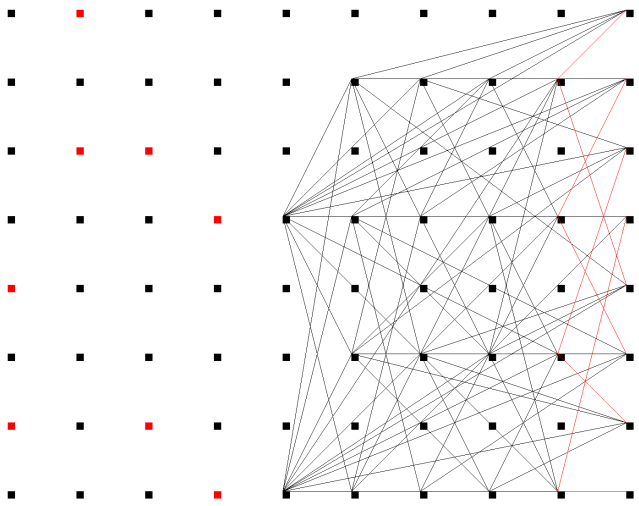

Figure 4: No edge is seen from the fourth clause type set anymore.

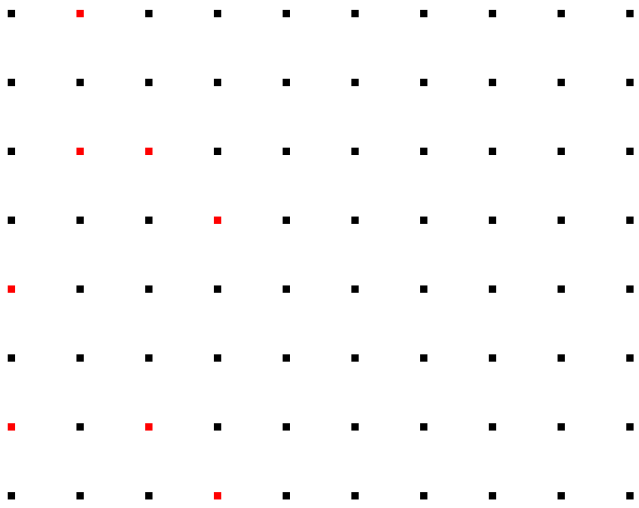

Figure 5: After processing the fourth clause type set, all edges vanished.