Measure and Conquer for Max Hamming Distance XSAT

Gordon Hoi[1] and Frank Stephan[2]

1 **School of Computing, National University of Singapore, 13 Computing Drive, Block COM1, Singapore 117417, Republic of Singapore,** e0013185@u.nus.edu

2 **Department of Mathematics and School of Computing, National University of Singapore, 10 Lower Kent Ridge Road, Block S17, Singapore 119076, Republic of Singapore,** fstephan@comp.nus.edu.sg

Abstract

XSAT is defined as the following: Given a propositional formula in conjunctive normal form, can one find an assignment to variables such that there is exactly only 1 literal that is true in every clause, while the other literals are false. The decision problem XSAT is known to be **NP**-complete. Crescenzi and Rossi [3] introduced the variant where one searches for a pair of two solutions of an X3SAT instance with maximal Hamming Distance among them, that is, one wants to identify the largest number k such that there are two solutions of the instance with Hamming Distance k. Dahllöf [4, 5] provided an algorithm using branch and bound method for Max Hamming Distance XSAT in $O(1.8348^n)$; Fu, Zhou and Yin [14] worked on a more specific problem, the Max Hamming Distance X3SAT, and found for this problem an algorithm with runtime $O(1.6760^n)$. In this paper, we propose an exact exponential algorithm to solve the Max Hamming Distance XSAT problem in $O(1.4983^n)$ time. Like all of them, we will use the branch and bound technique alongside a newly defined measure to improve the analysis of the algorithm.

1998 ACM Subject Classification 10003624 Discrete Mathematics, 10003626 Combinatoric problems

Keywords and phrases XSAT Problem, Maximum Hamming Distance of Solutions, Exponential Time Algorithms, Measure and Conquer

1 Introduction

The Satisfiability problem has been an important part of complexity theory and continues to be to this age. Given a Boolean formula φ in conjunctive normal form (CNF) , can we find an assignment to the variables such that there are at least 1 literal in each clause that evaluates to "True". There are many variants of the satisfiability problem to date and many are shown to be at least as hard as it. One variant that we will consider in this paper is the exact satisfiability problem (XSAT). Given a boolean formula φ in CNF, can we find a satisfying assignment such that exactly 1 literal in each clause is true while all the other literals are false. If we restrict the number of literals that can appear in any clause, then the problem comes as XkSAT, where k is the maximum number of literals that appear in any clause. Both XSAT and X3SAT are known to be **NP**-complete.

There are no exact polynomial time solution for these problems unless $\mathbf{P} = \mathbf{NP}$. As such, one has to resort to designing exponential time algorithms to solve these problems in the exact manner. One common way of designing exact algorithms is to build DPLL [7, 8] style algorithms. The idea behind is to branch over a selected variable in order to decompose the problem into smaller subproblems and then solve them recursively. Every branching algorithm contains two kinds of rules: simplification and branching rules. Simplification rules are used to simplify the problem or to terminate the algorithm. Branching rules, on the other hand, are used to recursively break the problem down into smaller subproblems. For more on this topic, we refer the reader to chapter 2 of the textbook of Fomin and Kratsch [12].

The overall runtime to decide XSAT has been well-explored. Dahllöf, Jonsson and Beigel [6] gave an algorithm in $O(1.1907^n)$, improving the current state of the art from $O(1.2299^n)$;

later Byskov, Maden and Skjernaa [2] improved it to $O(1.1749^n)$. The algorithms mentioned here gives a solution to a particular XSAT instance. However, there are times when we want to consider many solutions and we want to see how "different" are these from one another. For example, consider the UNIQUE SAT problem [1], where there must be only 1 solution to this problem. On the other hand, for the usual satisfiability problem, the number of solutions for that formula can vary and the solutions can be very "different" from one another.

To make precise this notion of "different" solution, the definition of the hamming distance problem captures this combinatorial aspect naturally. The Max Hamming Distance problem asks, given a formula φ in CNF, what is the maximum number of variables that can differ between any two solutions to φ? In this paper, we'll study the Max Hamming Distance XSAT problem where all solutions must satisfy a given XSAT instance. In fact, one may note that a solution to the Max Hamming Distance XSAT problem can even yield a solution to solve the XSAT decision problem. It closely resembles a counting problem and since it is more complex than a XSAT decision problem, a larger overall runtime than $O(1.1749^n)$ may be expected.

Dahllöf [4] first gave an algorithm for Max Hamming Distance XSAT in $O^*(2^n)$ and an improved version in $O^*(1.8348^n)$, where the notation $O^*(.)$ denotes the suppression of the polynomial terms. The first algorithm enumerates all possible subset of all sizes while checking that they meet certain condition. The second algorithm uses techniques found in DPLL algorithms. Fu, Zhou and Yin [14] worked on the X3SAT problem instead and gave an algorithm to determine the Max Hamming Distance of two solutions of an X3SAT instance in time $O^*(1.6760^n)$.

In this paper, we will propose an algorithm to solve the Max Hamming Distance XSAT in $O(1.4983^n)$. Since X3SAT is a more specific version of an XSAT instance, our algorithm practically solves both problems in a shorter overall runtime. Like the above authors, our algorithm is also a DPLL algorithm which consists of branching and simplification rules. The novelty in this paper is the designing of branching cases used in combination with a non-standard measure. If we were to use the standard measure, the algorithm takes $O(1.5761^n)$ ($\tau(7,1)^2 = 1.5761$), while the use of a nonstandard measure brings the same bottleneck down to $O(1.4983^n)$. This on the other hand meant that we have to pay special attention to some cases. We'll explain them in greater detail in the next section. In addition, our algorithm does a little more than just outputting the number to compute the Max Hamming Distance XSAT. It outputs a polynomial $p = \sum_k a_k u^k$ such that the coefficient a_k is the number of pairs of solutions with Hamming distance k of the given XSAT instance. In other words, we form for each pair of solutions the polynomial u^k where k is the Hamming distance of these solutions and then p is the sum over all the so obtained polynomials in the formal variable u.

2 Preliminaries

In order to analyse the time complexity of DPLL algorithms, one can consider search trees to help us illustrate the branching rules. One can consider the root of the search tree as the whole problem, and the successive child nodes as smaller instances of the problem when applying the branching algorithm. Using the search tree generated by the DPLL algorithm, if we can bound the number of leaves in the tree, then we will know the worst case runtime of this algorithm.

Kullmann [16] is one of the authors describing a technique to analyse DPLL algorithms; furthermore, Eppstein [10, 11] dealt with this technique in algorithms on graph colouring and more generally backtracking algorithms using the method of quasiconvex analysis of algorithms. The technique is to analyze each branching rule of an DPLL algorithm as follows: Let $T(n)$ denote the time needed for n variables, or more precisely the number of leaves of the search tree. Then the runtime of that branching rule is given by $T(n) = T(n-a_1) + T(n-a_2) + \ldots + T(n-a_r)$, where r denote the number of branches (or child nodes), $r \geq 2$, generated from that node and each branch i removes a_i many variables. This can then be formulated as a linear recurrence $x^n = x^{n-a_1} + x^{n-a_2} + \ldots + x^{n-a_r}$ and $\tau(a_1, a_2, \ldots, a_r) = \min\{x \geq 1 : x^{-a_1} + x^{-a_2} + \ldots + x^{-a_r} \leq 1\}$ is then the solution to the linear recurrence and $T(n) = \tau(a_1, a_2, \ldots, a_n)^n$. This $\tau(a_1, a_2, ..., a_r)$ is known as the branching factor. Note that $a_1 > a'_1$, then $\tau(a_1, a_2, ..., a_r) < \tau(a'_1, a_2, ..., a_r)$ and $\tau(i+\epsilon, j-\epsilon) < \tau(i, j)$, for all i, j, ε with $0 < i < j$ and $0 < \epsilon < \frac{j-i}{2}$.

A more sophisticated technique of analysing the runtime of branching algorithms is measure and conquer; a typical reference to this are by Fomin, Grandoni and Kratsch [13]. See also the textbooks of Downey and Fellows [9] and of Fomin and Kratsch [12]. It focuses on designing a new measure instead of changing the algorithm and a measure is a weight assigned to a variable in our case. Typically, a measure should observe 3 properties:

- A measure should be at least 0;
- The measure of an instance of a subproblem obtained after branching should be smaller than the measure of the instance before branching;
- The measure of an instance should be bounded above by some function on the parameter of the problem.

In simple measure, every variable is given a weight of 1 and eliminated variables are given a weight 0. When measure and conquer is used, usually one designs a new nonstandard measure in the hope of bringing down the analysis of algorithm further without changing the algorithm. One may then again apply Kullmann's technique to solve the linear recurrence under the new measure to obtain the running time of the branching rule.

3 High-level description of algorithm

We give a high level description of the algorithm here before giving the whole algorithm. To know the Max Hamming Distance of a given XSAT instance, we consider each pair of solution and the number of variables that each pair differs. Therefore, to achieve this, we will take in two identical instances of φ , we call it φ_1 and φ_2 such that $\varphi_1 = \varphi_2$ and branch them individually. Through the course of branching, φ_1 and φ_2 will differ later on.

The idea here is to branch whenever cases arise that match our branching cases. Our aim is to break connected components of clauses together into smaller isolated sets of clauses; this is done by first breaking chains of length 6 or more and second by branching variables that have 8 or more neighbours. Once they are small enough, we can brute force the Max Hamming Distance from the remaining sets of small isolated cliques. Note that the Hamming Distance can only be computed once the same variable on both φ_1 and φ_2 have been branched. Suppose that the branched i-th variable $x_{1,i}$ and $x_{2,i}$ have the same value, then they will have Hamming distance 0 and represent this value as $u^0 = 1$. Otherwise, they have different values and they will have Hamming Distance 1 and we represent this as $u^1 = u$. As we build the search tree, and we traverse a path from the root to the leaf, we multiply all the polynomials along each edge of the path. Finally, we form the sum all the multiplied polynomials along each path up.

We use polynomials to represent our Hamming Distance. The degree k of the polynomial denotes the Hamming Distance k and the coefficients of degree k denote the number of pairs of solution having Hamming Distance k. The notion of polynomials allow us to add and multiply together which makes it easier to understand. The reader might also consult Example 11 below for a better understanding of the algorithm.

4 A new measure

Here, we define the measure that we will be using throughout the entire algorithm. Given any variable x_i, we give the following formulas for weight w_i of variable x_i and measure μ as

$$w_i = \begin{cases} 1, & \text{if } x_i \text{ has at least 3 neighbours;} \\ 0.905813, & \text{otherwise;} \end{cases} \qquad \mu = \sum_{i=1,\ldots,n} w_i \leq n.$$

Here a neighbour is defined as a variable that appears alongside with it in the same clause, see Definition 1 below. So most cases of weight 0.905813 are variables occuring in exactly one clause and this has 3 literals. So if the formula consists of clauses

$(a, b, c, d), (c, d, e, f), (f, g, h)$ the weight of a, b, c, d, e, f is 1 each and the weight of g, h is 0.905813 each; for example, a has the neighbours b, c, d, c has the neighbours a, b, d, e, f and g has the neighbours f, h.

The idea of choosing the reduced weight for variables with two or less neighbours is that creating such variables, by branching a node in a 4-clause, should give some savings in anticipation of the further savings done by handling 3-clauses. Furthermore, the value 0.905813 is chosen by a computer program optimising the runtime and mainly satisfies that $3^{1/(3 \cdot 0.905813)}$ and $\tau(1+4 \cdot 0.094187, 7)^2$ are both bounded by 1.4983, which are the estimate of the basis of the exponentiation in Case 2 and the bottleneck of Case 1, respectively. Here note that $0.094187 = 1 - 0.905813$.

5 The algorithm in detail

We give a few definition that will be needed throughout the algorithm.

▶ **Definition 1.** A variable x occurs in a clause iff the clause contains at least one of the literals x and $\neg x$. Two variables are neighbours iff they are different and occur jointly in at least one clause; two clauses are neighbours iff there is at least one variable occurring in both of them.

For example, if there are clauses $(a, b, c), (c, d, e), (e, f, g)$ then the neighbours of the variable c are a, b, d, e; the clause (c, d, e) is a neighbour of the clause (a, b, c), but (e, f, g) is not a neighbour of (a, b, c).

▶ **Definition 2.** We say that a clause c is isolated if for every variable x_i in the clause c does not appear in any other clauses. We say that a variable is a singleton if it appears, as negated or an unnegated literal, in exactly one clause.

▶ **Definition 3.** We say that two variable x and y are linked together if there are up to 3 clauses which allow together to derive that either $x = y$ or $x = \neg y$. If two variables are linked, we can remove one variable, say y, by replacing all occurrences of y by respectively x or $\neg x$. We write $x \sim y$. If there are two clauses overlapping with each other, we call the non-overlapping variables that appear in the two clauses as *outside variables*.

▶ **Definition 4.** A chain is a sequence $C_1, C_2, \ldots, C_k$ of clauses such that C_i, C_j share at least one variable iff $|i - j| \leq 1$ and k is the length of the chain.

For example, the chain $(\alpha, b), (b, d, a), (a, e, c), (c, \beta)$ in Figure 12 is a chain of at least length 4 and $(a, b, c), (c, d, e)$ is a chain of length 2. Figure 5 is not a chain, as all three members have the joint variable a.

Algorithm MHXSAT (Max Hamming Distance XSAT).

- **Input:** φ_1 and φ_2, where $\varphi_1 = \varphi_2$ and both are instances are XSAT; A set of polynomials $p_{i,b_i,j,b_j}(u)$ for each possible pair (x_i, b_i, y_j, b_j) of variables x_i in φ_1 and y_j in φ_2 including specific variables x_0, y_0 which only occur with value $b_0 = 0$ and $c_0 = 0$.
- **Initial Recursive Call:** Call algorithm with φ_1, φ_2 both being the initial formula and the polynomials $p_{i,b,j,b'}$ be defined for all i, j, b, b' such that (i is index of x_i and $b \in \{0, 1\}$ or $i = 0 \wedge b = 0$) and (j is index of y_j and $b' \in \{0, 1\}$ or $j = 0 \wedge b' = 0$); the initial value $p_{i,b,j,b'}$ is as follows: if $i = j$ and $b \neq b'$ then $p_{i,b,j,b'}(u) = u$ else $p_{i,b,j,b'}(u) = 1$.
- **Output:** a formal polynomial $p(u) = \sum_k a_k u^k$ where for each k are a_k pairs of satisfying assignments where the two assignments have Hamming distance k.
- **Label Start.**
- **Simplification for φ_1.**
  - **If** there is a clause α where, whatever values one chooses for the variables, the sum of the literals is not 1 **Then Return** with the polynomial $p(u) = 0$.
  - **If** there are clauses α, β with a literal using x_i occuring at least in α and a value b_i which is the only value from $0, 1$ for x_i such that α, β can be made true **Then Begin** replace x_i by the constant b_i everywhere, update $p_{0,0,j,b'} = p_{0,0,j,b'} \cdot p_{i,b_i,j,b'}$

and remove x_i from the set of possible variables and i from the possible indices of polynomials on the φ_1-side and **Goto Start**; **End**.

- **If** there is a clause α mentioning exactly two variables x_i, x_j in the literals such that one can deduce either $x_i = x_j$ or $x_i = \neg x_j$ for all possible assignments making α true and there is at least one clause which contains exactly one of x_i, x_j and also a further variable in some literal **Then Begin** replace x_i everywhere by x_j or $\neg x_j$, respectively, and update $p_{j,b,k,b'} = p_{j,b,k,b'} \cdot p_{i,b,k,b'}$ or $p_{j,b,k,b'} = p_{j,b,k,b'} \cdot p_{i,1-b,k,b'}$ in the respective case for all $b, b' \in \{0,1\}$ and indices k on the φ_2-side and remove x_i, i from the list of possible variables and indices of φ_1 /* this is called linking */ **Goto Start**; **End.** /* Note that the only case where one cannot link the variables is the one where one variable occurs in two literals in the clause, say x_i and $\neg x_i$ or two times x_i; this case is already caught by the previous case, as either $x_i, \neg x_i$ both occur and the literal containg x_j must be 0 or x_i occurs twice and x_i must be 0. */
- **If** there are clauses of form $x_i \vee \alpha$ and $\neg x_i \vee \beta$ in φ_1 and x_i does not have 10 or more neighbours of weight 0.905813 **Then Begin** update for all x_k occurring in α with $b_k = 1$ if x_k is the literal in α and $b_k = 0$ if $\neg x_i$ is the literal in α the polynomials as $p_{k,b_k,j,b'} = p_{k,b_k,j,b'} \cdot p_{i,0,j,b'}$ and for all x_k occurring in β with $b_k = 1$ if x_k is the literal in β and with $b_k = 0$ if $\neg x_k$ is the literal in β the polynomials as $p_{k,b_k,j,b'} = p_{k,b_k,j,b'} \cdot p_{i,1,j,b'}$ for all variables y_j on the φ_2-side and all $b' = 0, 1$ and replace all clauses of the form $x_i \vee \gamma$ by $\beta \vee \gamma$ and all clauses of the form $\neg x_i \vee \delta$ by $\alpha \vee \delta$ and remove x_i, i from the list of possible variables and indices of φ_1 /* this is called making a cut or also called resolving x_i */ **Goto Start**; **End**.

- **Simplication for φ_2.** Do the actions analogue to those for φ_1 given above and go back to **Start** whenever any of these actions has been performed.
- **Branching for φ_1.**
  - **If** there is a clause of form $\alpha \vee \beta$ of φ_1 such that branching $\alpha = 1$ versus $\alpha = 0$ has branching factor at most 1.4983 as indicated in Proposition 5 or List 1 (= Proposition 6) or List 2 (= Proposition 7), in this order of priority, and the subclauses α, β both contain at least one literal then compute $p = \mathbf{MHXSAT}(\varphi_1 \wedge \alpha, \varphi_2, pols) + \mathbf{MHXSAT}(\varphi_1 \wedge \beta, \varphi_2, pols)$ and go out with **Return(p) End.** /* This is called branching $\alpha = 1$ versus $\alpha = 0$; for doing the branching, the only requirement is that there is a clause in φ_1 of the form $\alpha \vee \beta$ where both α, β have at least one literal. After doing all choices by Proposition 5, one can w.l.o.g. assume by cut, branching and renaming that all clauses in φ_1 contain only positive literals; after doing all choices of List 1, there are except for isolated components of size up to six, no clauses in φ_1 with a multiple overlap; after doing all choices of List 2, there are no proper chains of length 6 or more in φ_1 and no variables with more than 7 neighbours. */
- **Branching for φ_2.** Do the actions analogue to those for φ_1 given above.
- **Now none of the above cases applies.** Note that no variable has more than 7 neighbours (unless it is in an isolated component of size 9 by List 2 item 3) and no proper chain is longer than 5 clauses; thus both formulas are split into isolated components which by Proposition 8 contain at most 1364 variables and by a more involved argument with a slightly modified algorithm in Proposition 14 in the appendix at most 67 variables.
- **Measure** φ_1 and φ_2 and do the following with the smaller task, say with φ_1.
  - **Compute** an explicit list of satisfying assignments for φ_1 only using the surviving variables x_i obtaining a list of vectors with entries b_i for each $i \neq 0$;
  - **Compute** for each assignment (b_i) an updated list of polynomials *pols* obtained by updating the previous polynomials to $p_{0,0,j,b'} = p_{0,0,j,b'} \cdot \Pi_{i \neq 0} p_{i,b_i,j,b'}$;
  - **For each** (b_i), *pols* **Do Begin**
    - **For each component** of variables and clauses C in φ_2 compute all possible assignments (b_j) of the $y_j \in C$ and update

$$p_{0,0,0,0} = p_{0,0,0,0} \cdot \Big(\sum_{\textit{assignment } (b_j) \textit{ of } C} \;\; \prod_{x_j \textit{ occurring in } C} p_{0,0,j,b_j} \Big).\mathbf{End}$$

- Let p be the sum of all the $p_{0,0,0,0}$ calculated for the above list of $(b_i), pols$ and **Return**(p).

6 The Simplifications and the Branchings from List 1 and List 2

▶ Proposition 5. If $a, \neg a$ appear both in clauses then either one can do a cut without increasing the weight or one can branch a with branching-factor below 1.31719.

Proof. For the simplifications, as they remove variables, one should not expect problems. However, there is one case, namely making the cut. In the case of a cut eliminating variable d, one replaces clauses $d \vee \alpha$ and $\neg d \vee \beta$ by $\alpha \vee \beta$ and this increases the number of neighbours of the variables in the disjunctions α and β and might therefore increase the weights of some variables from 0.905813 to 1. If there are at most 9 such variables, the saving of removing d is at least 0.905813 while the weights going up are at most $9 \cdot 0.094187 < 0.905813$ and there is no problem; here note that $1 - 0.905813 = 0.094187$. If there are at least 10 such variables, one can see that for both cases $d = 0$ and $d = 1$, some additional literals are set to 0 and therefore variables are eliminated, thus one can branch d instead of making a cut and the worst case is that the eliminated variables are distributed 2 on one side and all others on the other side, hence one has $\tau(1 + 2 \cdot 0.905813, 1 + 8 \cdot 0.905813)^2 \leq 1.31719$. For that reason, one can assume when reaching the branching case, that all variables are non-negated – if a variable d is negated everywhere, one replace $\neg d$ by d everywhere and adjusts the polynomials accordingly. ◀

Now, we begin to analyse the time needed for each rule in the algorithm. We will omit the simplification rules and instead only analyze the branching rules since the simplification rules do not increase the number of leaves of the search tree. Before we go on to analyze the runtime complexity, we give an example of the different branching cases below in List 1 and List 2. List 1 contains all the different cases that we handle when there are two or more variables that appear between two clauses. List 2 contains all the different branching cases where there is only a common variable between two clauses.

List 1.

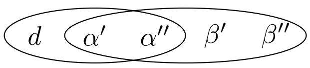

Figure 1.

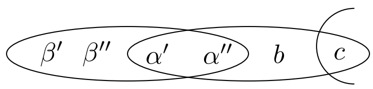

Figure 2.

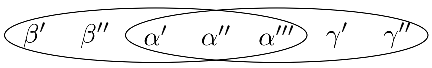

Figure 3.

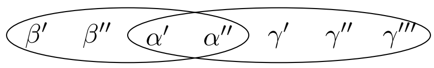

Figure 4.

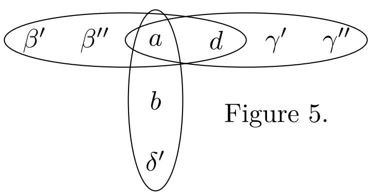

Figure 5.

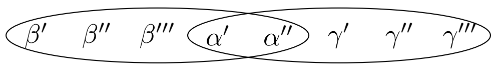

Figure 6.

This list gives how to handle overlaps between of two or more variables between two clauses. Furthermore, the variables are in each clause listed nonredundantly, so if a clause is (α, β), the two lists of variables α and β are disjoint. Lower case single letters always refer to single variables. Note that the case that an isolated component with up to six variables is not covered, where isolated means that no other clause has some but not all variables from this component. Such components are treated in Case 2. In the figures above, primed Greek letters like α', α'' denote the variables in the corresponding list of variables like α in this case. Furthermore, we indicate only those clauses which are relevant; additional clauses might connect to those given in the figures or case distinction, unless explicitly said otherwise.

1. $(d,\alpha),(\alpha,\beta)$ where α,β are lists of literals not containing d and α has at least two variables. Now one can see that $d=\beta$ and by adding $\neg d$ on both sides, one gets $1=\neg d+\beta$. Thus one has the formulas $(d,\alpha),(\neg d,\beta)$ and these allow a cut, as in the last item of the Simplification part of the algorithm. In the case that making the cut would cause more than 10 variables to be upgraded from weight 0.905813 to 1 and therefore the measure would go up, one does not make the cut but instead branches the variable d. In the case of a branching, note that there is at least one variable in β and furthermore d has weight 1; furthermore, the literals with weight 0.905813 are except a perhaps single one in β all connected to d, as $\neg d$ was just introduced and the variables in α have weight 1. This gives the branching factor $\tau(1+0.905813, 3+10\cdot 0.905813)^2 \leq 1.2900$. For example in Figure 1, $|\alpha|=2$, there are 3 outside variables.
2. $(\beta,\alpha),(\alpha,b,c),(c,\delta)$ with δ containing some variable d not in (α,b,c) and α,β have both at least two variables.
   If $\delta=b\vee d$ then the situation of Item 1 applies after some renaming, as the clauses $(d,c,b),(c,b,\alpha)$ exist. Thus this case is already handled.
   If δ contains at least three variables including b then d has weight 1 and will in the case $\alpha=0$ be set to 0 again and the branching factor is at most $\tau(3,4)^2\leq 1.4903$.
   If δ does not contain b then the case $\alpha=0$ allows to link b to c and removes the variables in α and b which all have weight 1 and $\alpha=0$ allows to remove those in β,b,c which all have weight 1 and so the branching factor is again at most $\tau(3,4)^2\leq 1.4903$.
   For example in Figure 2, we have two 4-clauses with overlapping part α consisting of two variables.
3. $(\beta,a,d),(a,d,\gamma),(a,b,\delta)$ or $(\beta,a,d),(a,d,\gamma),(a,b,d,\delta)$ and Items 1,2 do not apply and $|\beta|=|\gamma|=2$. Now every subclause containing variables of β is a subclause of (β,a,d) and every subclause containing variables of γ is a subclause of (γ,a,d) and b does not occur in a,d,β,γ.
   One branches $a+d=0$ versus $a+d=1$. If $a+d=0$ then the weight of all the variables in β,γ is reduced from 1 to 0.905813. If $a+d=1$ all variables in β,γ are set to 0 and either a can be linked to d or b can be set to 0, depending on what the third clause is. So the branching factor is at most $\tau(2+4\cdot 0.094187, 4+0.905813)^2\leq 1.4888$.
   For example in Figure 5, we have the variables a,d as the overlapping variables and we have $|\delta|=1$.
4. Not Items 1,2,3 and $|\alpha|+|\beta|+|\gamma|\geq 7$. This in particular means that α,β,γ have at least two variables. As Item 2 does not apply, if β has two exactly two variables then all clauses containing variables from β are subclauses of (α,β), similarly for γ. One branches $\alpha=0$ versus $\alpha=1$.
   Now consider the case where α has two variables and β,γ have together exactly five variables, so one of them, say β, has exactly two variables. Now consider the subcase that one or both variables from α are in a further clause containing a b not occurring in α,β,γ. Now $\alpha=1$ makes the variables in β,γ all 0 and further allows to either link the variables in α or makes $b=0$; furthermore, $\alpha=0$ makes the variable in β have a weight 0.905813 while before they had weight 1. Thus the branching factor is at most $\tau(2+2\cdot 0.094187, 5+0.905813)^2\leq 1.4498$. In the other subcase that α has exactly two variables and they do not occur in clauses with variables outside α,β,γ, one takes into account that choosing $\alpha=1$ reduces the weight of the variables of α from 1 to 0.905813. So the branching factor is at most $\tau(2+2\cdot 0.094187, 5+2\cdot 0.094187)^2\leq 1.4917$.
   If $|\alpha|=2$ and $|\beta|+|\gamma|\geq 6$ then the branching factor is at most $\tau(2,6)^2\leq 1.4656$ without any further assumptions. If $|\alpha|\geq 3$ then the branching factor is at most $\tau(3,4)^2\leq 1.4903$.
   In Figure 3, we have that $|\alpha|=3$ and we have 4 outside variables where $|\beta|+|\gamma|=4$; Figures 4 and 6 represent typical cases where $|\alpha|=2$ and $|\beta|+|\gamma|\geq 5$.

This case distinction shows that one can get rid of all multiple overlaps which are in components larger than six variables, if a component contains only $(\alpha,\beta),(\beta,\gamma)$ with α,β,γ having each two variables and perhaps further clauses only using these variables. This situation allows no simplification, but one can let it stand, as the component is already sufficiently small and deal with the other components in φ_1 and φ_2 until those are also broken down.

To see the completeness of the above case-distinction in List 1, note that Item 1 deals with the case that one of the neighbours of α is only a single variables d; in all other cases it is assumed that the basic situation is $(\beta, \alpha), (\alpha, \gamma)$ with α, β and γ each having at least two variables. Item 2 considers the case that α has a neighbour $b \vee c$ where the variable c occurs in a further clause with at least one variable different from those in α, b, c. Item 3 considers the case where both neighbours of α have two variables and the condition from Item 2 does not apply to these neighbours, but that at least one variable a of α occurs in a clause where not all variables are from α, β, γ. Item 4 considers the case where $|\alpha| + |\beta| + |\gamma| \geq 7$ and the above cases do not apply.

List 2. In this list, if a variable is branched, it is called a. Furthermore, b, c are variables in the chain which link the clauses considered to further members and these contain exactly one of b, c each. One does the first case in this list which applies. Again note that the case of one variable a in four clauses of size 3 without any further variables in the isolated component except for a and its eight neighbours does not need any further treatment. Thus in item 1 of the following list, one can assume that there is either one neighbour with weight 1 or at least 10 neighbours with weight 0.905813.

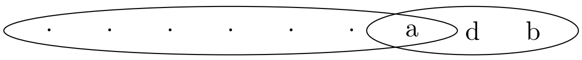

Figure 7.

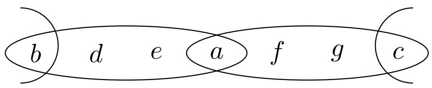

Figure 8.

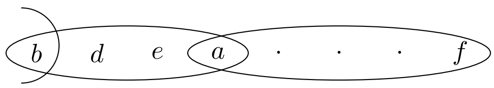

Figure 9.

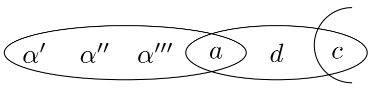

Figure 10.

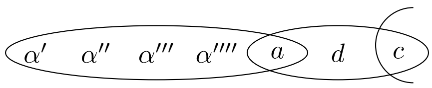

Figure 11.

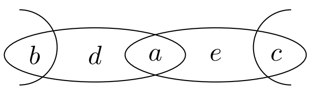

Figure 12.

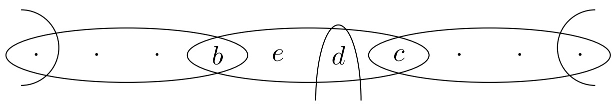

Figure 13, subsequence of 6-chain

1. One variable a with at least eight neighbours. One branches a. If there are eight neighbours and at least two of them have weight 1 or if there are at least nine neighbours then the branching factor is at most $\tau(1, 6 \cdot 0.905813 + 3)^2 \leq 1.4967$. If there are exactly eight neighbours and exactly one of them has weight 1 then the neighbours are in four 3-clauses and only one of these has a variable b shared with another clause, let d denote the other variable in this clause. If $a = 0$, b and d will be linked, if $a = 1$ all neighbours are 0. This gives $\tau(1 + 0.905813, 7 \cdot 0.905813 + 2)^2 \leq 1.3775$.
2. $(\alpha, a), (a, d, c)$ where α has at least three variables. One branches a. Note that a has at least five neighbours out of which only d can have measure 0.905813. This gives $\tau(1 + 0.905813, 5 + 0.905813)^2 \leq 1.4824$. For example in Figure 10, we have the overlapping variable as a, c appears as part of a larger chain and $|\alpha| = 3$ here. We have 5 outside variables in this case.
3. $(b, d, a), (a, e, c)$. One branches a. If $a = 0$ then one can link both d to b and e to c else all five variables are determined. Thus $\tau(1 + 2 \cdot 0.905813, 3 + 2 \cdot 0.905813)^2 \leq 1.4518$. This is exactly the case as given in Figure 12 where a is our overlapping variable and we have 4 outside variables in this case.
4. $(b, d, e, a), (a, f, g, c)$ with each of d, e, f, g being singleton variables and b, c being in

further clauses. Now one branches a and gets $\tau(1+4\cdot 0.094187, 7)^2 \leq 1.4983$, as d, e, f, g will change their measure by the branching in the case that $a = 0$. This is exactly the case as given in Figure 8 where a is the overlapping variable and we have 6 outside variables in this case.

5. $(b, d, e, a), (a, \ldots, f)$ and d is a singleton variable and the clause $(a, \ldots, f)$ having at least 5 variables. Now one branches a and has $\tau(1+0.094187, 8)^2 \leq 1.4983$. We see this example in Figure 9 where a is our overlapping variable and we have 7 outside variables in this case.

6. (b, d, e, c) and b, c are only in inner clauses of some chain and d is in a further clause. By the cases with the 3-clauses above being done first, the clauses on the other side of b and c contain at least four literals. Thus b, c have both at least six neighbours. If one of them, say b, would have further neighbours which have weight 0.905813, these would have to be in a further 3 clause and so b would have eight neighbours, so that List 2 Item 1 above applies, thus this case does not happen. Now one distinguishes $b + c = 0$ versus $b + c = 1$. In the case that $b + c = 0$, one can link d, e and eliminate three variables. In the case that $b+c = 1$, d, e are 0 and one can make a cut exploiting that $b = \neg c$ and that neither b nor c have neighbours which have weight 0.905813. So no weight-compensation is needed and 4 variables are eliminated. This gives $\tau(3, 4)^2 \leq 1.4903$. Note that we needed the fact that the neighbours at b, c are connected to further clauses in the chain only to guarantee that these clauses have at least four variables; therefore we apply this branching rule also when a 4-clause with three neighbours satisfies that two of them are disjoint and have four variables each. An example of this can be found in Figure 13.

Assume now by way of contradiction that after all actions in List 2 are done there would be a chain of length 6. Note that no variable a in the chain can have exactly eight neighbours with all having weight 0.905813, as then every variable in the chain is either a or a neighbour of a and thus the chain has at most length 2. None of the second, third, fourth and fifth member of this chain can be 3-clauses, as these are eliminated as above. The third and fourth both cannot be two 4-clauses containing two singletons. Neither the third nor the fourth can be a 4-clause with at most one singleton variable, as such clauses are also eliminated. The third and the fourth clause cannot be a 4-clause plus a clause of five or more variables, as in such situation again the connecting variable is branched. The third plus the fourth clause cannot together have 9 or more variables, as then the connecting variable a has at least eight neighbours and can be branched. Thus there is no chain of length 6 or more.

7 Analysis of Algorithm

For the verification, we divide our algorithm into two portions: Case 1 on the different simplification and branching cases and Case 2 on the brute forcing of the independent cliques of clauses. We will be using our defined measure in this analysis as given in Section 4. In addition, note that we are branching on both formulas φ_1 and φ_2. This means that we have $2n$ many variables in this case. All branchings in this proof are kept to the standard branching of a variable taking on values either 1 or 0 unless explicitly mentioned. We pay more attention to 3-literal and 4-literal clauses due to our nonstandard measure. In addition, we exploit the fact these clauses which we branch belong to a larger chain. We analyze the worst runtime needed for the first part below.

▶ Proposition 6. Let $k \geq 2$. If there are k overlapping variables between two clauses, then the worst case time complexity for branching these overlapping variables in the two clauses is $O(1.4917^n)$.

Proof. First we note that by the simplifications in the algorithm and Proposition 5, we can assume without loss of generaltiy, that all literals are positive, that is, not negated. If only $\neg a$ occurs for some variable a, we just simply replace $\neg a$ everywhere by a and adjust the polynomials accordingly.

In the following we say that the outside variables in a pair of clauses $(\beta, \alpha), (\alpha, \gamma)$ are in $|\beta|-|\gamma|$ orientation. List 1, Item 1 tells us how to handle outside variables in 1-m orientation, where $m \geq 2$. Thus we only have to deal with m-m' orientations where $m \geq 2$ and $m' \geq 2$.

When $k = 2$ and if there are 4 outside variables, then it must be in the 2-2 orientation (Figure 2). In this case, we exploit that these clauses are part of a larger chain. If not, then they are an isolated component and will be handled by Case 2 of our algorithm. Therefore, one of the variables in one of the orientation must be connected to a clause somewhere. Let α', α'' be our overlapping variables and let c be our variable that is connected to a different clause and b be a variable appearing in the clause $(\alpha' \vee \alpha'' \vee b \vee c)$. We branch $(\alpha' \vee \alpha'') = 1$ and $\alpha' = \alpha'' = 0$. Branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 4 outside variables and branching $\alpha' = \alpha'' = 0$ will allow us to remove both α', α'', and further link up the variables b and c. Therefore, we have at most $T(\mu) = T(\mu - 4) + T(\mu - 3) = O(1.2208^{2\mu}) = O(1.4903^n)$. On the other hand, suppose that the we are not allowed to link b and c together, then it must be that c appears in another clause containing b as well. This other clause that c appears in must be at least 4-literal in length else List 1 Item 1 would have handled it for us. There must be a new variable d that is different from the overlapping variables and outside variables. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 4 outside variables and branching $\alpha' = \alpha'' = 0$ will allow us to remove both α' and α'' and at least another variable d. This gives us $T(\mu) = T(\mu - 4) + T(\mu - 3) = O(1.2208^{2\mu}) = O(1.4903^n)$. This completes the case for 4 outside variables.

If there are 5 outside variables as shown in Figure 4, then we must have them in the 2-3 orientation. Let β' and β'' be the two variables in the "2" orientation, while γ',γ'' and γ''' be the three variables in the "3" orientation. We consider 4 different cases here. The first case is that one of the β' or β'' is connected to a larger chain and one of α' or α'' or both variables are in a further clause containing a new variable b that is different from the outside variables and the overlapping variables. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 5 outside variables and link $\alpha' = \alpha''$ or remove b. Branching $\alpha' = \alpha'' = 0$ will allow us to remove 2 overlapping variables and link β' to β''. This gives us $T(\mu) = T(\mu - 5.905813) + T(\mu - 3) = O(1.1757^{2\mu}) = O(1.3822^n)$. The second case is that one of the β' or β'' is connected to a larger chain and α' and α'' do not appear in further clauses. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 5 outside variables and we can also factor in the change in measure for α' and α''. Branching $\alpha' = \alpha'' = 0$ will allow us to remove both overlapping variables and link β' to β''. This will give us $T(\mu) = T(\mu - 5 - 2 \cdot 0.094187) + T(\mu - 3) = O(1.1897^{2\mu}) = O(1.4154^n)$. The third case is that both β' and β'' are not connected to a larger chain and one of α' or α'' or both variables are in a further clause containing a variable b that is different from the outside variables and overlapping variables. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 5 outside variables and link up $\alpha' = \alpha''$ or remove b. Branching $\alpha' = \alpha'' = 0$ will allow us to remove both overlapping variables and also factor in the change of measure for β' and β''. This gives us $T(\mu) = T(\mu - 5.905813) + T(\mu - 2 - 2 \cdot 0.094187) = O(1.2041^{2\mu}) = O(1.4498^n)$. Finally, the last case is that both β' and β'' are not connected to a larger chain and α' and α'' do not appear in a different clause. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 5 outside variables and allow us to factor in the change of measure for α' and α''. On the other hand, branching $\alpha' = \alpha'' = 0$ will allow us to remove both overlapping variables and factor in the change in measure for both β' and β''. This will give us $T(\mu) = T(\mu - 5 - 2 \cdot 0.094187) + T(\mu - 2 - 2 \cdot 0.094187) = O(1.2214^{2\mu}) = O(1.4917^n)$. This completes the case for 5 outside variables.

Now, for the number of outside variables j, with $j \geq 6$, regardless of the orientation of the outside variables, we have that applying the branching technique as above, we will arrive at $T(\mu) = T(\mu - 6) + T(\mu - 2) = O(1.2106^{2\mu}) = O(1.4657^n)$ regardless if they appear as part of a larger chain. Now we have that $\tau(j, 2)^2 \leq \tau(6, 2)^2 < 1.4657$. The case for $k = 2$ is therefore complete.

For $k = 3$, we consider the case that we have 4 outside variables as shown in Figure 3. Let our overlapping variables be $\alpha', \alpha'', \alpha'''$ and we will branch $(\alpha' \vee \alpha'' \vee \alpha''') = 1$ and

$\alpha' = \alpha'' = \alpha''' = 0$ and this gives us $T(\mu) = T(\mu-3)+T(\mu-4) = O(1.2207^{2\mu}) = O(1.4903^n)$ regardless if they appear in a larger chain. Now for $j \geq 4$ outside variables, our branching factor must be at most $\tau(3,j)^2 \leq \tau(4,3)^2 = 1.4903$. The case for $k = 3$ is therefore complete.

For $k > 3$ overlapping variables and for $j \geq 4$ outside variables, our branching factor must be bounded above by 1.4903. This can see from the fact that $\tau(k,j)^2 < \tau(4,3)^2 = 1.4903$. Therefore, the case for all $k \geq 2$ has been covered and will take at most $O(1.4903^n)$ time. ◀

▶ Proposition 7. The worst-case runtime complexity of the branching cases when there is exactly 1 overlapping variable between two clauses is $O(1.4983^n)$.

Proof. Let j be the number of outside variables that the overlapping variable has. Note that $j \geq 4$. For $j = 4$, we have that it must be in a 2-2 orientation as shown in Figure 12. Now, we branch the overlapped variable and this gives us $T(\mu) = T(\mu - 3 - 2 \cdot 0.905813) + T(\mu - 1 - 2 \cdot 0.905813)$. Therefore, we have $T(\mu) = O(1.2049^{2\mu}) = O(1.4518^n)$. Note that this is the worst case when we have exactly two singleton in two of the outside variable.

If $j = 5$, then we can only have them in the 2-3 orientation like in Figure 10. Therefore, we branch the overlapping variable a and we have $T(\mu) = T(\mu - 5 - 0.905813) + T(\mu - 1 - 0.905813 + 2 \cdot (0.905813 - 1)) = O(1.2083^{2\mu}) = O(1.4600^n)$. In this case, we assume that there are 3 singletons out of the 5 outside variable. If we have 2 singletons out of the 5 variables spread out in a 1-1 fashion, then we have $T(\mu) = T(\mu-5-0.905813)+T(\mu-1-0.905813-(1-0.905813)) = O(1.2128^{2\mu}) = O(1.4709^n)$. On the other hand, if both singletons are now on the 4-literal clause, then we have $T(\mu) = T(\mu-6)+T(\mu-2) = O(1.2107^{2\mu}) = O(1.4658^n)$. If we have 1 singleton out of the 5 variable, then we consider the fact that this singleton can be at the 4-literal or the 3-literal clause. If it appears on the 3-literal clause, then we have $T(\mu) = T(\mu - 5 - 0.905813) + T(\mu - 1 - 0.905813) = O(1.2176^{2\mu}) = O(1.4826^n)$. If it appears on the 4-literal clause, then we have $T(\mu) = T(\mu - 6) + T(\mu - 2 - (1 - 0.905813)) = O(1.2062^{2\mu}) = O(1.4550^n)$. Finally, if there are no singletons, then we have $T(\mu) = T(\mu - 6) + T(\mu - 2) = O(1.2106^{2\mu}) = O(1.4656^n)$. This completes the case for 5 outside variables.

If $j = 6$, then we can have it either in the 2-4 orientation or the 3-3 orientation, as shown in Figure 11 and 8 respectively. If it is the 2-4 orientation, then we branch the overlapping variable and we have $T(\mu) = T(\mu - 6 - 0.905813) + T(\mu - 1 - 0.905813) = O(1.19655^{2\mu}) = O(1.4318^n)$. For the 5-literal clause, we do not need to consider the case if there are singleton variables or not as the presence or absence of it will not change the measure. Therefore, we will only need to consider the case where there are no singletons on the 3-literal clause. We have $T(\mu) = T(\mu-7)+T(\mu-2) = O(1.1908^{2\mu}) = O(1.4181^n)$. If the outside variables appear in the 3-3 orientation, then again, we branch the overlapping variable. We will therefore have $T(\mu) = T(\mu-7)+T(\mu-1-4\cdot(1-0.905813)) = O(1.22403^{2\mu}) = O(1.22403^{2n}) = O(1.4983^n)$. In this case, we assume that all 4 out the 6 outside variables are singleton. We also need to handle the case that there are less than 4 singletons in this case. Suppose that there exist at least one of the variables such that it is not a singleton, as shown in Figure 13, then we have to change our approach in branching this problem because of the change in measure. Instead of looking at the overlapping variable, we instead look at the 4-literal clause containing the non-singleton variable. Let the variables b and c be connected to a larger chain, d be that non-singleton variable and the last variable be e. Now both b and c cannot be connected to 3-literal clauses as the earlier cases would already have handled it. In addition, if b and c appear at least 3 times, else we can branch them immediately to get a branching factor of at most $T(\mu) = T(\mu - 7 - 2 \cdot 0.905813) + T(\mu - 1) = O(1.2165^{2\mu}) = O(1.4798^n)$. Therefore both b and c must appear exactly twice. Now we branch $(b \vee c) = 1$ and $b = c = 0$. If $b = c = 0$, then we can eliminate 3 variables by linking up d and e and if $(b \vee c) = 1$, we can remove all 4 variables. Therefore, $T(\mu) = T(\mu - 3) + T(\mu - 4) = O(1.2207^{2\mu}) = O(1.4903^n)$. This completes the case for $j = 6$ outside variables.

If $j = 7$, then we can have it either in the 3-4 orientation (Figure 9) or the 2-5 orientation. Now suppose that we have the 3-4 orientation. Then we will have $T(\mu) = T(\mu - 8) + T(\mu -$

$1 - 2 \cdot (1 - 0.905813)) = O(1.2169^{2\mu}) = O(1.4809^n)$. In this case, we assume that 2 of the 7 outside variables in the 4-literal clause are singleton.

If this case does not happen, then again we have to look at the other neighbours of the 4-literal. Now, the neighbours of this 4-literal clause cannot be 3-literal or 4-literal, as they would have been handled by the earlier cases. Therefore, the neighbours of this 4-literal clause must be a 5-literal clause.

We first choose any two variables b and c that are non-singletons. If variables b and c appear at least 3 times , then we'll branch them immediately to get a branching factor of at most $T(\mu) = T(\mu - 8 - 2 \cdot 0.905813) + T(\mu - 1) = O(1.2003^{2\mu})O(1.4406^n)$. If not, then these variables appear exactly twice and and we branch them as $(b \vee c) = 1$ or $b = c = 0$. Then we will have a branching factor of $T(\mu) = T(\mu - 3) + T(\mu - 4) = O(1.2208^{2\mu}) = O(1.4904^n)$. If we have it in the 2-5 orientation, then we have $T(\mu) = T(\mu - 7.905813) + T(\mu - 1.905813) = O(1.1799^{2\mu}) = O(1.3922^n)$. Now, if there are no singletons on the 3-literal clause, then we will have $T(\mu) = T(\mu - 8) + T(\mu - 2) = O(1.1750^{2\mu}) = O(1.3807^n)$. This completes the case for 7 outside variables.

If $j = 8$, for example in Figure 7, branching the overlapping variable will give us a branching factor $\tau(j, i)^2 \leq \tau(9, 1)^2 < 1.4718$ regardless of the orientation of the outside variables. In addition for $j > 9$, we have that $\tau(j, 1)^2 < \tau(9, 1)^2 < 1.4718$. Note that this case applies to a 9-literal clause. We can just branch any variable appearing in a 9-literal clause to have a branching factor of $\tau(9, 1)^2 = 1.4718$. If there are 8 neighbours with two of them having at least weight 1, then we have a branching factor of at most $\tau(1, 6 \cdot 0.905813 + 3)^2 = 1.4967$. If there are exactly 8 neighbours and one of them having weight 1, then we have a case of an overlapping variable appearing in four 3-literal clauses. Branching the common variable will give us a branching factor of at most $\tau(1 + 0.905813, 7 \cdot 0.90513 + 2)^2 = 1.3775$. This completes the case for all overlapping variables of exactly 1 variable.

The upperbound of these branching rules is the branching rules that has the worst case time bound. Since the branchng rule of the common variable between two 4-literal clause has the worst case timebound, therefore the worst case time bound is $O(1.4983^n)$. ◀

After all the above branching rules have been applied and we come to a point where no branching rules can be further applied, we come to the second part of the algorithm where both φ_1 and φ_2 consists of small isolated components. Figure 14 shows an example of case 2. Each circle in both φ_1 and φ_2 represents an isolated group of clauses.

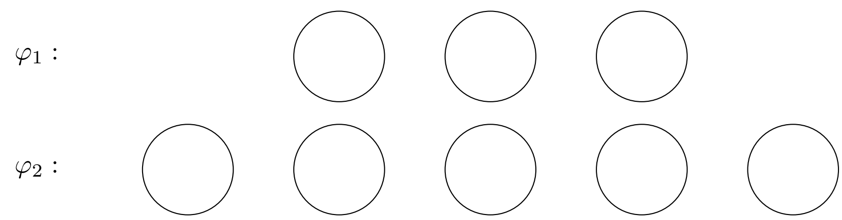

Figure 14. Case 2

The following gives a rough estimate of the size of the isolated components; a better one is found in the appendix.

▶ Proposition 8. After all actions in Case 1 is done, φ_1 and φ_2 consist of disjoint components each having at most 1364 variables.

Proof. Note that by doing the actions in List 1, every two clauses in the component intersect by at most one variable, unless the component exist of exactly two clauses of the form $(a, b, c, d), (c, d, e, f)$. By doing the actions in List 2, every chain has at most length 5 and every chain of four members has in the interior a clause of size 4 and perhaps also a clause of size 5. So one chooses a clause C_1 of size 4. Now when starting from C_1, one gives an upper bound on all nodes which are in the last clause of chains of form C_1 or C_1, C_2 or C_1, C_2, C_3 or C_1, C_2, C_3, C_4 or C_1, C_2, C_3, C_4, C_5. Here one uses that each variable in an inner C_k has at least four and at most five members and thus each $a \in C_k$ has at most four new neighbours which are not covered by chains of the form $C_1, C_2, \ldots, C_k$. So one gets the overall number of variables estimated by $4 \cdot (1 + 4 + 16 + 64 + 256) = 5 \cdot 341 = 1364$. The appendix gives an improved bound of 80 with a much more involved argument which also needs a slight generalisation of the cases in List 2. ◀

Let μ be the measure for φ_1 and ν for φ_2 at this point. Now we will apply branch and bound to the formula φ_k, where $k \in \{1, 2\}$ with $min\{\mu, \nu\}$ and then brute force the Hamming Distance from the other formula. For the brute forcing portion, we know that the size of each component is bounded above by some constant c. Therefore, the only thing we need to ensure is that the branch and bound of the formula measure is still well within the time bound of $O(1.4983^n)$.

▶ Proposition 9. The branch and bound of the formula with lower measure has worst case run time of $O(1.4983^n)$.

Proof. Note that when we are in this case, we no longer have $2n$ variables but only n variables from one of the formula. In addition, we can safely assume that we will not have a variable with 8 other neighbours as it will be handled by the branching case above. We consider all possible cases here.

First we consider standalone clauses with length < 9. Suppose that we have a clause of length 2. Then we branch the entire clause by the values $(1, 0)$ and $(0, 1)$. This will incur $T(\mu) = T(\mu - 2 \cdot 0.905813) + T(\mu - 2 \cdot 0.905813) = O(1.4662^n)$. Suppose that we have a length of length 3, then again we branch the clause with values $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$. This will incur $T(\mu) = 3T(\mu - 3 \cdot 0.905813) = O(1.4983^n)$. Now let $n \in \mathbb{N}$. We know that $n^{(\frac{1}{n})}$ is decreasing for $n \geq 3$ and hence for any length of any standalone clauses with $3 < j < 9$, we have that the branching factor with j branches and removal of j variables for each branch, $\tau(j, j, ..., j) < \tau(3 \cdot 0.905813, 3 \cdot 0.905813, 3 \cdot 0.905813) < 1.4983$. This completes the case for all standalone clauses.

We next consider clauses with at exactly 1 overlapping variable with < 8 neighbours. Now the worst case that we can have is an overlapping variable with 4 outside variables in a 2-2 orientation. In this case, we will have $T(\mu) = T(\mu - 1 - 4 \cdot 0.905813) + T(\mu - 1) = O(1.3433^n)$. Again, let j be the number of neighbours in this case with $4 < j < 8$. Then $\tau(1 + j \cdot 0.905813, 1) < 1.3433$. This completes the case for all clauses with exactly 1 overlapping variable. For two or more overlapping variables, we can treat it as a similar case by just branching only 1 of the overlapping variable. This completes the case for all clauses with overlapping variables. ◀

▶ **Theorem 10.** *The algorithm takes $O(1.4983^n)$ time.*

Proof. To know the worst case runtime of our algorithm, we have to consider the branching rule which generates the most number of leaves. For branching rules in Case 1, we have that the runtime is bounded above by $O(1.4983^n)$ as given by Proposition 7 and 6. For Case 2, as shown in Proposition 9, the runtime is again bounded above by $O(1.4983^n)$. So the overall complexity is $O(1.4983^n)$. ◀

▶ Example 11. Consider φ_1, φ_2 to contain the following clauses : $x_1 \vee x_2 \vee x_3$, $x_1 \vee x_4 \vee x_5$, $x_1 \vee x_6 \vee x_7$, $x_2 \vee x_8 \vee x_9 \vee x_{10}$. For variable x_k, the initial values of the polynomials are $p_{k,a,k,b} = u$ in the case that $a \neq b$ and 1 in the case that $a = b$; furthermore, all polynomials involving mixed variables are 1 and also the polynomials with $k = 0$ on either side are 1.

Now the algorithm branches x_1, first in φ_1. Now let x_1 take the value a_1 in φ_1.

If $a_1 = 1$ then the variables x_h with $h = 2, 3, 4, 5, 6, 7$ take the value $a_h = 0$. Furthermore, we update the polynomials as follows: $p_{0,0,k,b} = \prod_{h=1,...,7} p_{h,a_h,k,b}$ for all k, b and after that we let $p_{h,a,k,b} = 0$ for $h = 1, 2, \ldots, 7$ and all a, b, k. The remaining formula in φ_1 is $x_8 \vee x_9 \vee x_{10}$.

If $a_1 = 0$ then one can conclude that $x_2 = \neg x_3$, $x_4 = \neg x_5$ and $x_6 = \neg x_7$. However, only the first of these 3 possible equalities will be realised, as x_2 appears also in a further clause. So we do the update $x_3 = \neg x_2$ in φ_1 and $p_{3,a,k,b} = p_{3,a,k,b} \cdot p_{2,1-a,k,b}$ for all a, b, k and after that $p_{2,a,k,b} = 0$ for all a, b, k. The remaining formulas for φ_1 are in this case $x_4 \vee x_5$, $x_6 \vee x_7$, $x_2 \vee x_8 \vee x_9 \vee x_{10}$.

After that, one does the analogous updates in φ_2.

The descent will result in 4 subcases where one has on each side either 3 variables and one clause or 8 variables distributed over 3 disjoint and unconnected clauses.

Now consider the example case where φ_1 has one clause (so one has branched $x_1 = 1$ previously for φ_1) and φ_2 has 3 clauses. Now one considers the 3 cases of (x_8, x_9, x_{10}) taking the values $(0, 0, 1)$, $(0, 1, 0)$ and $(1, 0, 0)$ in φ_1 and one subbranches into these cases which will set the remaining variables accordingly. Then only the polynomials $p_{0,0,k,b}$ are non-zero and we will process the 3 components of variables (x_2, x_8, x_9, x_{10}), (x_4, x_5) and (x_6, x_7) for φ_2 accordingly.

So for each $(b, b', b'') \in \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}$, we do separate computations where each of them starts with the same version of the polynomials and first updates for all applicable k and all $a \in \{0, 1\}$ the polynomials $p_{0,0,k,a} = p_{0,0,k,a} \cdot p_{8,b,k,a} \cdot p_{9,b',k,a} \cdot p_{10,b'',k,a}$ and once, this is done, one updates the polynomial $p_{0,0,0,0}$ 3 times as follows: First for 4 possible vectors in $U = \{(0, 0, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0,), (1, 0, 0, 0)\}$ of (x_2, x_8, x_9, x_{10}), we update $p_{0,0,0,0} = p_{0,0,0,0} \cdot (\sum_{(c,c',c'',c''')\in U} p_{0,0,2,c} \cdot p_{0,0,8,c'} \cdot p_{0,0,9,c''} \cdot p_{0,0,10,c'''})$ and then we deal with the two solutions for $x_4 \vee x_5 = 1$ by updating $p_{0,0,0,0} = p_{0,0,0,0} \cdot (p_{0,0,4,1} \cdot p_{0,0,5,0} + p_{0,0,4,0} \cdot p_{0,0,5,1})$ and afterwards we do the same for $x_6 \vee x_7 = 1$ by updating $p_{0,0,0,0} = p_{0,0,0,0} \cdot (p_{0,0,6,1} \cdot p_{0,0,7,0} + p_{0,0,6,0} \cdot p_{0,0,7,1})$ and we receive for each of the 3 starting vectors (b, b', b'') a sum polynomial and these 3 sum polynomials are added up to the return value of this branch. The other 3 cases arising from different branchings of x_1 in either formula are handled analogously.

8 Conclusion

In this paper, we introduced the concept of finding the most number of variables that can differ between a pair of solution and called it the Max Hamming Distance problem and we focused on the Max Hamming Distance XSAT problem. We introduced a DPLL algorithm with a nonstandard measure to bring the complexity down to $O(1.4983^n)$, and therefore beating both Dahllöf's state of the art algorithm to solve Max Hamming Distance XSAT and Fu, Zhou and Yin's algorithm to compute the Max Hamming Distance X3SAT.

Here is a possible direction where interested readers can take our work further. Our current nonstandard measure gave us a huge improvement in performance from $O(1.5761^n)$ if we were to use the standard measure. Can a more creative and cleverly designed measure bring the complexity of the algorithm down further?

An anonymous referee also pointed out that the large constant of Proposition 8 is large and that this constant goes in exponentiated form into the runtime; thus the algorithm is only of theoretical nature and not implementable in practice. Proposition 14 in the appendix gives a better value, but there is still room for improvement.

One might ask whether there are heuristics using known methods which might beat our algorithm. One such approach would be to enumerate all solutions of φ_1 at the beginning and then to solve for each of this solution φ_2. In the general case, we note that the number of solutions of φ_1 is a badly conditioned function. Say if there are $n = 3m + 1$ variables consisting on m clauses with 4 variables where always the first variable is the same and the other 3 are uniquely to the clause, then the number of solutions is $1 + 3^m$ which is least $\Omega(1.4422^n)$. For each of these solutions, though not in this specific case, one has to solve on the other side a variable-weighted maximum XSAT formula which actually takes longer than solving XSAT; Porschen [17] solved this in $O(2^{0.244n})$ which is $O(1.184^n)$. So combining known algorithm might give only $O(1.7075^n)$, as $1.184 \cdot 1.4422 \geq 1.7075$. Thus even if the first bound can be improved, as the current bound requires an easy structure, we do not expect this method to give our bounds.

Hoi, Jain and Stephan [15] provide a better algorithm for computing the maximum hamming distance of X3SAT; however, this better bound exploits several properties of X3SAT which do not hold in general XSAT and this method does not generalise here. It is a often observed phenomenon that algorithms for X3SAT have a better time performance than their counterparts for the more general XSAT problem.

Acknowledgments. The authors would like to thank the anonymous referees of ISAAC 2019 for useful suggestions. Furthermore, the authors would like to thank internet companies for putting services like Wolfram Alpha Equation Solver, Firefox Scratchpad and Google Scholar for free onto the internet. F. Stephan was supported in part in part by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019 / R146-000-234-112.

References

1. Andreas Blass and Yuri Gurevich. On the unique satisfiability problem. *Information and Control*, 55(1–3):80–88, 1982.
2. Jesper Makholm Byskov, Bolette Amitzbøll Madsen and Bolette Skjernaa. New algorithms for exact satisfiability. *Theoretical Computer Science*, 332(1–3):515–541, 2005.
3. Pierluigi Crescenzi and Gianluca Rossi. On the Hamming distance of constraint satisfaction problems. *Theoretical Computer Science*, 288(1):85–100, 2002.
4. Vilhelm Dahllöf. Algorithms for Max Hamming Exact Satisfiability. *International Symposium on Algorithms and Computation*, ISAAC 2005, *Springer Lecture Notes in Computer Science*, 3827:829–383, 2005.
5. Vilhelm Dahllöf. *Exact Algorithms for Exact Satisfiability Problems*, Ph.D. dissertation, Department of Computer and Information Science, Linköping University, 2006.
6. Vilhelm Dahllöf, Peter Jonsson and Richard Beigel. Algorithms for four variants of the exact satisfiability problem. *Theoretical Computer Science*, 320(2–3):373–394, 2004.
7. Martin Davis, George Logemann and Donald W. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.
8. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.
9. Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Springer, Berlin, Heidelberg, 2013.
10. David Eppstein. Small maximal independent sets and faster exact graph coloring. *Proceedings of the Seventh Workshop on Algorithms and Data Structures. Springer Lecture Notes in Computer Science*, 2125:462–470, 2001.
11. David Eppstein. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Transactions on Algorithms*, 2(4):492–509, 2006.
12. Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, Heidelberg, 2010.
13. Fedor V. Fomin, Fabrizio Grandoni and Dieter Kratsch. A measure and conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5), article no 25, 2009.
14. Linlu Fu, Junping Zhou and Minghao Yin. Worst case upper bound for the maximum Hamming distance X3SAT problem. *Journal of Frontiers of Computer Science and Technology*, 6(7):664–671, 2012.
15. Gordon Hoi, Sanjay Jain and Frank Stephan. A fast exponential time algorithm for Max Hamming X3SAT. *Foundations of Software Technology and Theoretical Computer Science*, FSTTCS 2019, to appear.
16. Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1–2):1–72, 1999.
17. Stefan Porschen, On variable-weighted exact satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 51(1):27–54, 2007.

Appendix

An anonymous referee pointed out to the authors, that the size of the components after Case 1 is critical to the performance of the algorithm when implemented, as it is multiplicative constant in the runtime which is exponentiated, so the overall runtime multiplies with approximately $2^{m/2} \cdot Poly(n)$ which is an estimate to count all the weighted number of solutions of an component of size m with the weights being polynomials in u of degree n. One can go for this through all possible solutions by descent. Due to the exponentiation of m, it is important that one gets a bound on m to be as small as possible. Though a large m does not influence the theoretical analysis of the algorithm in terms of its asymptotic complexity, it still makes the algorithm useless for practical implementations. The following relaxation from chains to pseudochains is critical for the improved bound on m.

▶ **Definition 12.** A pseudochain is a sequence of clauses $C_1, C_2, \ldots, C_k$ such that two neighbouring clauses C_h, C_{h+1} overlap by exactly one variable and for two subsequent pairs of neighbouring clauses C_h, C_{h+1} and C_{h+1}, C_{h+2}, the variables in the overlaps are different.

In this definition, the second condition implies that there are no pseudochains with a subsequence C_h, C_{h+1}, C_h, so that one cannot go back and forth between two clauses. Now for the following, the conditions in List 2 will be slightly generalised, in the sense that they can also apply to pseudochains and not only proper chains. So in Item 2, it is allowed that the neighbouring clause of c has a joint variable with the first clause, but this needs to be different from c. In Item 3, b, c are in further clauses and there are no constraint on what these further clauses are, they could even be the same clause; however, $b \neq c$ is required. In Items 4 and 5, it is not required that b, c are in further clauses and if they are, there are no constraints on what they are. In Item 6, it is only required that the two additional clauses where b and c are in have at least four variables, no further requirement is there. The two clauses have to be different, but they can have a joint variable. Note that b and c are then exactly in two clauses, as they have already six neighbours and as they cannot have eight neighbours out of which six have weight 1 by Item 1 to be done first when it applies. These relaxations do not influence the branching factors.

▶ Proposition 13. Every component of the formulas after Case 1 with the more modified conditions in List 2 as above does not contain pseudochains of length six or more; furthermore, it contains no circular pseudochains like C_1, C_2, C_3, C_1.

Proof. Note that Item 1 enforces that no variable a in a component has eight or more neighbours, except in the case that the whole isolated component of the consists the variable a and eight neighbours which are all in clauses $(a, ., .)$ consisting of a and two further variables. Furthermore the Simplification Rules enforce that there are no clauses of 1 or 2 variables, except the case of an isolated component consisting of a single 2-variable-clause. Thus no member of a pseudochain has more than six variables, as its neighbour must have at least three variables. Items 2 and 3 in a pseudochain enforce that no inner member of a pseudochain has three variables except in the case of a pseudochain C_1, C_2, C_3 where all three clauses have exactly three variables. If now a pseudochain is of the form $C_1, C_2, C_3, C_4, C_5, C_6$, the clauses C_2, C_3, C_4, C_5 have all at least four variables and it cannot be that both C_3, C_4 have both four variables, as then either Item 4 applies or one of them has a variable connecting to a further clause and Item 6 applies, causing a further reduction of the variables. Furthermore, it cannot be that one of C_3, C_4 has four variables and the other one five or more variables, as then either Item 5 or Item 6 applies. Furthermore, it cannot be that both C_3, C_4 have at least five variables, as then the connecting variable can be branched by Item 1. Thus pseudochains of six or more clauses do not survive until all branchings or cuts which can apply by List 2 are done.

Note that for a circular pseudochain $C_1, C_2, \ldots, C_k, C_1$, it is required that C_2 and C_k connect to C_1 by different variables, as the writing should not depend on where one breaks the circle. Thus one can also view it as a long pseudochain $C_1, C_2, \ldots, C_k, C_1, C_2, \ldots, C_k$ and, as $k \geq 3$, this psuedochain has at least six members and does not exist after all branchings of the modified List 2 are done. ◀

▶ Proposition 14. Every component of the formulas after Case 1 with the more modified conditions in List 2 as above has at most 80 variables.

Proof. As seen in the preceding proposition, there are no circular pseudochains and no pseudochains of length six or more. So let $C_1, C_2, \ldots, C_k$ be a pseudochain of maximal length, therefore k is at most five.

Now consider the case $k = 5$. Let a be the common variable of C_2, C_3 and b be the common variable of C_3, C_4. If there is a clause of distance 3 from C_3, it is connected by clauses C_3, C_6, C_7, C_8. Either C_6 does not connect to C_3 through a or does not connect to C_3 through b, say the first. Now $C_1, C_2, C_3, C_6, C_7, C_8$ is a pseudochain of length six which does not exist. Thus all clauses are either neighbours of C_3 or neighbours of neighbours of C_3. So assume that C_6 is a neighbour of C_3 with common variable c which has a further neighbour C_7 which is not a neighbour of C_3. Then C_6 has at least four and at most five variables. Thus c can only be in the clauss C_3, C_6 as these have together already at least seven variables and being in a further clause would cause c to have eight or more neighbours. So there are either three or four variables in C_6 other than c and these have each at most $8 - |C_6|$ neighbours which are not in C_6. So if C_6 has four variables these are 3 neighbours of c outside C_3 plus $3 \cdot 4$ neighbours of neighbours; if C_6 has five variables these are 4 neighbours of c outside C_3 plus $4 \cdot 3$ neighbours of neighbours; in total c contributes to at most 16 neighbours and neighbours of neighbours. Note that 16 will only be reached, if C_6 has five variables and therefore C_3 four. So one has at most $\max\{4 \cdot (1+16), 5 \cdot (1+15)\} = 80$ variables in the isolated component.

If $k = 4$ then every clause would have at most distance two from C_2 which can be seen as follows: If there would be a clause of distance three then it would be a pseudochain C_2, C_5, C_6, C_7 which can be either extended to C_1, C_2, C_5, C_6, C_7 or C_3, C_2, C_5, C_6, C_7 and does not exist. Similarly, any clause has distance at most three from C_3. If now a clause C_7 is of distance 2 from both C_2, C_3 then this is witnessed by C_2, C_5, C_7 and C_3, C_6, C_7 and C_5, C_6 do not connect to C_2, C_3, respectively, through the common variable a of C_2 and C_3. Thus C_7, C_5, C_2, C_6, C_7 would be pseudochain of length five with does not exist by assumption. Thus, every clause other than C_2 and C_3 is a neighbour of exactly one of these two clauses, but not of both. At most one of C_2, C_3 is of size five and both have at least size four. Thus a has up to 15 neighbours and neighbours of neighbours from one side and up to 16 from the other side, so the overall number of variables is at most $16 + 15 + 1 = 32$.

If $k = 3$ then every clause is a neighbour of C_2. Each variable in C_2 has at most $8 - |C_2|$ neighbours which are not in C_2 and so the overall number of nodes is variables in the connected component is bounded by $|C_2| + |C_2| \cdot (8 - |C_2|) = |C_2| \cdot (9 - |C_2|)$ which is maximised at $|C_2| \in \{4, 5\}$ and is 20.

If $k = 2$ then every clause contains the connecting variable of C_1 and C_2 so that the overall number of variables is at most 9. If $k = 1$ then the clause has at most size 8 and there are no neighbouring clauses. ◀

It might be worth to mention that when dealing in a brute-force way with a component of size 80, one can break it down by taking, in the case of $k = 5$, first the central clause C_3 and branch into the subcases according to which of the variabls in it is 1, each of these branches splits the component into subcomponents of size at most 16 which are easy to handle, as all variables of C_3 are set to constants. This would then make the case of dealing with isolated components to become more treatable from an implementation perspective. In the case of $k = 4$, one can branch the variable connecting C_2 and C_3 and get a similar breakdown of the component into two smaller, isolated components of up to 16 variables. The cases $k = 3, k = 2, k = 1$ have already very small components.