Faster Exact Solving of SAT Formulae with a Low Number of Occurrences per Variable

Magnus Wahlström[⋆]

Department of Computer and Information Science,
Linköping University,
SE-581 83 Linköping, Sweden
magwa@ida.liu.se

Abstract. We present an algorithm that decides the satisfiability of a formula F on CNF form in time $O(1.1279^{(d-2)n})$, if F has at most d occurrences per variable or if F has an average of d occurrences per variable and no variable occurs only once. For $d \leq 4$, this is better than previous results. This is the first published algorithm that is explicitly constructed to be efficient for cases with a low number of occurrences per variable. Previous algorithms that are applicable to this case exist, but as these are designed for other (more general, or simply different) cases, their performance guarantees for this case are weaker.

1 Introduction

The boolean satisfiability problem, and its restricted variants, is one of the most well studied classes of NP-complete problems. Since no algorithm for general formulae on conjunctive normal form (CNF) with a worst-case running time of $O(c^n)$ for $c < 2$ is known, or even believed to exist (the currently fastest algorithms for SAT run in $O(2^{n(1-1/\log_2 2m)})$ time [6, 15] where m is the number of clauses and expected time $O(2^{n-c\sqrt{n}})$ for a constant c [5]), a large amount of work has been done on restricted variants of the problem that are easier to solve. Most notable of these restricted problems is k-SAT where a clause may have at most k literals. This is polynomial for $k = 2$ and NP-complete for $k > 2$ [9]. The best results for 3SAT are a probabilistic algorithm which runs in time $O(1.3238^n)$ [11] and a deterministic one which runs in time $O(1.473^n)$ [1], and for general k-SAT a probabilistic algorithm with running time in $O((2 - 2/k)^n)$ [14] and a deterministic algorithm in time $O((2 - 2/(k + 1))^n)$ [4].

If every variable is limited to at most d occurrences in a formula, SAT is solvable in linear time when $d = 2$ and NP-complete when $d \geq 3$, and k-SAT where every clause has exactly k literals is trivial when $d \leq k$ (every such formula is satisfiable) and NP-complete otherwise. If shorter clauses are allowed, 3SAT is NP-complete when $d \geq 3$ [17]. Previous comparable algorithms include an

⋆ This research work was funded by CUGS (the national graduate school in computer science).

unpublished result by Kullmann (cited in [13]), where he achieves $O(3^{n/9}) \approx O(1.1299^n)$ for SAT instances where $d = 3$, and two algorithms with running times characterised by measures other than n that give non-trivial results for some of these cases. Hirsch has given algorithms [10] that run in time $O(1.2389^K)$ and $O(1.0740^L)$ for a general SAT formula with K clauses or total length L, which would give $O(1.2389^n)$ for $d = 3$ and $O(1.3305^n)$ for $d = 4$, and Szeider has given a fixed-parameter tractable algorithm with a running time characterised by the maximum deficiency of a formula F [16]. If $K(F)$ is the number of clauses in F and $N(F)$ the number of variables, the maximum deficiency of F is $D = \max_{F'}(K(F') - N(F'))$ over every subformula F' of F, and Szeider's algorithm runs in time $O(2^D n^4)$. For a k-SAT instance where every clause has exactly k literals, and with $d \geq k$, this would give us a running time of $O(2^{(d/k-1)n})$, or $O(1.2600^n)$ when $d = 4$ and $k = 3$ (or indeed for any formula F where $d = 4$ and where no clause has fewer than three literals). When 2-clauses are allowed, the result is not as strong. In this paper, we present an algorithm that runs in time $O(1.1279^{(d-2)n})$ for any CNF formula with at most d occurrences per variable, *i.e.* $O(1.1279^n)$ when $d = 3$ and $O(1.2721^n)$ when $d = 4$, regardless of the lengths of the clauses. In a slight variation of this, we get the same running time if a formula F is free of singletons (variables with one occurrence) and the average number of occurrences per variable is d. Hirsch's results hold for this case as well. This latter case is perhaps of less complexity theoretical interest, but might occur in practice, if one has an application that creates formulae with few high-degree variables. The reason that we cannot allow singletons in the latter case is that they lower the average degree too much. While all singletons disappear in reductions without adding any significant extra running time, the value of the function used as an upper bound on the running time, which is a function of $(d - 2) \cdot n$, actually decreases when singletons are added–in the extreme case, one could add singletons until $d < 2$ and the upper bound would collapse. It would be possible to compensate for this by introducing extra terms in the function, but this would make the entire analysis more complex. We feel that the restriction, which only applies when one is counting the average number of occurrences per variable, is not severe enough to warrant this.

The results in this paper are achieved by a compact recursive algorithm with about a dozen cases. The strategy used in the algorithm is to use reductions and branchings to gradually impose more structure onto the problem instance, until finally the problem is structured enough to be converted into an instance of $(3, 2)$-CSP with a third as many variables. This instance can then be solved by a fast algorithm for this problem, such as Eppstein's algorithm [8]. In particular, we found it possible to enforce a large amount of structure when every variable occurs at most three times in F. Another key component is our measure of complexity $f(F) = L(F) - 2N(F)$, which we use to guide the decision of when to apply certain reductions. It is this measure that allows us to construct an algorithm that is simultaneously very strong when every variable occurs at most three times, and able to handle instances where some variables have a higher number of occurrences well.

The structure of this paper is as follows. Section 2 contains some notes on standard concepts used in the text, Section 3 presents the algorithm, Section 4 contains the proof of its running time, and finally Section 5 contains conclusions and some discussion of future work.

2 Preliminaries

A SAT instance is a boolean formula F in CNF form, with no restrictions on the clause lengths. A k-SAT instance is a SAT instance where no clause contains more than k literals; a clause with exactly k literals is a k-clause. The problem instances that are considered in this paper belong to the general SAT class, without restrictions on clause lengths. $Vars(F)$ is the set of all variables that occur in F.

Regarding notation, if v is a variable then $\neg v$ is its negative literal, and if l is the literal $\neg v$ then $\neg l$ is the literal v. In general, l (or l', l_i, etc) represents a literal that may be positive or negative, while other lowercase letters represent variables or positive literals. $|C|$ for a clause C denotes the number of literals in C (also called the length of C), and a clause $(l \vee C)$ for some literal l and clause C is the clause that contains all literals of C plus the literal l. Similarly, $(l \vee C \vee D)$ for literal l and clauses C and D would be the clause containing l plus every literal occurring in C or D. Unless explicitly stated otherwise, the clauses C, D must be non-empty.

If l is a literal of a variable occurring in F, $F[l]$ is the formula one gets if every clause C in F that contains l, and every occurrence of $\neg l$ in other clauses, is removed. For a set of literals A, $F[A]$ is the result of performing the same set of operations for every literal $l' \in A$. Note that $F[l]$ and $F[A]$ may contain empty clauses.

For a variable $v \in Vars(F)$, define the *degree* $d(v, F)$ of v in F to be the number of occurrences of either v or $\neg v$ in the clauses of F. Usually, the formula is understood from the context, and $d(v)$ is used. $d(F)$ is the maximum degree of any $v \in Vars(F)$, and F is *k-regular* if $d(v) = k$ for every $v \in Vars(F)$. A variable v where the literal v occurs in a clauses and $\neg v$ occurs in b clauses is an (a, b)-variable, in which case v is an a-literal and $\neg v$ a b-literal. If $b = 0$, then v is a *pure literal* (similarly, if $a = 0$ then $\neg v$ is a pure literal). We will usually assume, by symmetry, that $a \geq b$, so that *e.g.* any 1-literal is always a negative literal $\neg v$. If $d(v) = 1$, then v is a *singleton.*

The *neighbours* of a literal l are all literals l' such that some clause C in F contains both l and l'. If a clause C contains a literal of both variables a and b, then a and b *co-occur* in C.

We write $L(F)$ for the length of F and $N(F)$ for the number of variables in F (*i.e.* $L(F) = \sum_{v \in Vars(F)} d(v, F)$ and $N(F) = |Vars(F)|$).

We use the classic concept of *resolution* [7]. For clauses $C = (a \vee l_1 \vee \ldots \vee l_d)$ and $D = (\neg a \vee m_1 \vee \ldots \vee m_e)$, the *resolvent* of C and D by a is the clause $(l_1 \vee \ldots \vee l_d \vee m_1 \vee \ldots \vee m_e)$, shortened to remove duplicate literals. If this new clause contains both v and $\neg v$ for some variable v, it is said to be a *trivial resolvent.*

For a formula F and a variable v occurring in F, $DP_v(F)$ is the formula where all non-trivial resolvents by v have been added to F and all clauses containing the variable v have been removed from F. Resolution is the process of creating $DP_v(F)$ from F.

(d,l)-CSP is the constraint satisfaction problem where each variable has d possible values and every constraint involves l variables. $(3,2)$-CSP is used in a subcase of the algorithm in this paper, and for this problem there is an algorithm due to Eppstein [8] which runs in $O(1.3645^n)$ time for n variables. The problem formulation we use is the one used by Eppstein: An instance I of (d,l)-CSP is a collection of variables and a collection of constraints. For each variable v, there is a list of up to d values (called colours) that v can take, and each constraint is a tuple of up to l (variable,colour)-pairs. The constraints are seen as illegal combinations: A constraint $((v_1,X_1),\ldots,(v_k,X_k))$ is satisfied if, for at least one i, $1 \le i \le k$, variable v_i is assigned a colour different than X_i. The instance I is satisfied if there is an assignment of one colour to each variable that satisfies every constraint.

3 The Algorithm

The algorithm *LowdegSAT*(F) for determining the satisfiability of a CNF formula F is shown in Figure 1. It is shown as a list of cases, where the earliest case that applies is used, *e.g.* case 8 is only used if none of cases 0–7 apply. Case 0 is a base case. Cases 1–5 are referred to as *simple reductions*, since the effect of these cases is only to remove literals or variables from F, without adding any new literals or variables. Cases 6–7 are the non-simple reductions, and cases 8–12 are branchings.

We use $f(F) = L(F) - 2N(F) = \sum_{v \in Vars(F)}(d(v,F) - 2)$ as a measure of complexity, motivated by the fact that this is the maximum number of occurrences of v that need to be removed from F before v can be removed by a polynomial-time reduction. As an indication that this is a relevant measure, when using this measure the worst cases for $d(v) = 3$ and $d(v) = 4$ both get the same branching number, as we see in the next section. $f(F)$ is also used in the algorithm for deciding whether to apply certain reductions and branchings or not.

We say that a formula F' is the *step k-reduced version* of F if F' is the result of applying the reductions in cases 0–k, in the order in which they are listed, until no such reduction applies anymore. F' is called step k-reduced (without reference to F) if no reduction in case k or earlier applies (*i.e.* F is step k-reduced if *LowdegSAT*(F) reaches case $k+1$ without applying any reduction). A synonym to step 7-reduced is fully reduced.

Definition 1. Standardising *a CNF formula F refers to applying the following reductions as far as possible:*

1. *Subsumption: If there are two clauses C, D in F, and if every literal in C also occurs in D, then D is* subsumed *by C. Remove D from F.*

Algorithm LowdegSAT(F)

Case 0: If $F = \emptyset$, return 1. If $\emptyset \in F$, return 0.
Case 1: If F is not on standard form, standardise it (see Def. 1).
Case 2: If there is some 1-clause $(l) \in F$, return $LowdegSAT(F[l])$.
Case 3: If there is a pure literal l in F, return $LowdegSAT(F[l])$.
Case 4: If there is a 2-clause $(l_1 \vee l_2)$ and a clause $D = (l_1 \vee \neg l_2 \vee C)$ in F for some possibly empty C, construct F' from F by deleting $\neg l_2$ from D and return $LowdegSAT(F')$.
Case 5: If there is a variable x in F with at most one non-trivial resolvent (such as a $(1,1)$-variable), return $LowdegSAT(DP_x(F))$.
Case 6: If there is a variable x in F with $d(x) = 3$ such that resolution on x is admissible (see Def. 2), return $LowdegSAT(DP_x(F))$.
Case 7: If there are two clauses $(C \vee D), (C \vee E)$, with $|C| > 1$, construct F' from F by replacing these two clauses by $(C \vee \neg x), (x \vee D), (x \vee E)$ for a newly introduced variable x, and return $LowdegSAT(F')$.
Case 8: If $d(F) > 3$, pick a variable x of maximum degree. If some literal of x, assume $\neg x$, occurs in a single clause $(\neg x \vee l_1 \vee \ldots \vee l_k)$, return $LowdegSAT(F[x]) \vee LowdegSAT(F[\{\neg x, \neg l_1, \ldots, \neg l_k\}])$. If both x and $\neg x$ occur in at least two clauses, return $LowdegSAT(F[x]) \vee LowdegSAT(F[\neg x])$.
Case 9: If there is a 2-literal l such that $f(F)$ reduces by at least six in a branch $F[l]$, assume that $\neg l$ occurs in a clause C along with literals $l_1, \ldots, l_k$ and return $LowdegSAT(F[l]) \vee LowdegSAT(F[\{\neg l, \neg l_1, \ldots, \neg l_k\}])$.
Case 10: If there is a clause $C = (\neg v_1 \vee \ldots \vee \neg v_k)$ that contains only 1-literals, and $|C| \geq 4$, return $LowdegSAT(F - C + (\neg v_1 \vee \ldots \vee \neg v_{\lfloor k/2 \rfloor})) \vee LowdegSAT(F - C + (\neg v_{\lfloor k/2 \rfloor + 1} \vee \ldots \vee \neg v_k))$.
Case 11: Let a be a 2-literal (assumed to be positive) with a maximum number of neighbours. Let the clause that contains $\neg a$ be $(\neg a \vee \neg b \vee \neg c)$. If the literal a has at least three neighbours, return $LowdegSAT(F[a]) \vee LowdegSAT(F[\{\neg a, b, c\}])$.
Case 12: If no previous case applied, the formula can be converted into a $(3,2)$-CSP instance with $N(F)/3$ variables, as described in Lemma 1. Perform this conversion, and apply Eppstein's algorithm from [8].

Fig. 1. Algorithm for SAT when most variables have few occurrences

2. *Trivial clauses: If there is a clause C in F such that both literals v and $\neg v$ occur in C for some variable v, then C is a trivial clause. Remove it from F.*
3. *Multi-occurring literals: If there is a clause C in F where some literal l occurs more than once, remove all but one of the occurrences of l from C.*

A formula F where none of these reductions apply is said to be on standard *form.*

Definition 2. *Let F be a step 5-reduced SAT formula, and let F' be the step 5-reduced version of $DP_x(F)$, for some variable x that occurs in F. Then, resolution on x in F is* admissible *if $f(F') \leq f(F)$,* i.e. *$L(F) - L(F') \geq 2(N(F) - N(F'))$.*

Definition 3. Backwards resolution *is the operation of replacing two clauses $(C \vee D)$, $(C \vee E)$ in a formula F with $(\neg a \vee C), (a \vee D), (a \vee E)$ for a new* *variable a. To avoid loops of reductions, we only use this when $|C| > 1$, so that the net difference in $f(F)$ is strictly positive.*

Lemma 1. *Given a 3-regular SAT formula F where all 2-literals occur only in 2-clauses and all 1-literals occur only in 3-clauses, there is a corresponding $(3, 2)$-CSP instance I, constructible in polynomial time and with $N(F)/3$ variables, that is satisfiable if and only if F is satisfiable.*

Proof. By Lemma 14.6 of [13], a formula F with c 3-clauses and otherwise only 2-clauses can be converted into an instance I of $(3, 2)$-CSP with c variables (by first creating one variable in I for each clause in F, and then performing a reduction used by Eppstein in [8] to remove every variable with only two values). Since every variable of F occurs in only one 3-clause, the resulting instance I has $N(F)/3$ variables. □

Next we show the correctness of *LowdegSAT*. First we show the correctness of the branching that is used in some of the cases (introduced in [12], under the name complement search), and then the overall correctness of the algorithm. The proof contains some references to lemmas in the next section, as we feel that the correctness belongs to this section while these other lemmas are more easily shown in the context of algorithm analysis. No circular references occur, since no lemma in the analysis section refers to this lemma.

Lemma 2. *Let $\neg x$ be a 1-literal in a formula F, and let the clause where $\neg x$ occurs be $C = (\neg x \vee l_1 \vee \ldots \vee l_d)$. Then either $F[x]$ is satisfiable, or $F[\neg x]$ and $F[\{\neg x, \neg l_1, \ldots, \neg l_d\}]$ are equi-satisfiable (*i.e.* either both are satisfiable, or neither).*

Proof. Assume that $F[x]$ is unsatisfiable. Then, if there is a satisfying assignment A to F, it must set $\neg x$ to true and changing the value of x in A must create an unsatisfied clause. The only possible such clause is C, which means that all other literals of C must be false in A. □

Lemma 3. *The algorithm LowdegSAT applied to a CNF formula F correctly calculates the satisfiability of F.*

Proof. Case 0 is correct by the definition of the problem, and cases 1–4 are easily checked. Cases 5–7 use resolution, and the correctness of this operation is proven in *e.g.* [7]. Furthermore, the reduction process will terminate. If F' is the step 3-reduced version of F, by Lemma 4 the simple reductions keep $f(F')$ non-increasing and decrease $L(F')$. Resolution keeps $f(F)$ non-increasing while decreasing the number of variables, implying that $L(F)$ decreases, and creates no singletons. Backwards resolution decreases $f(F)$ strictly and also creates no singletons. This shows that no infinite chain of reductions is possible from F' onwards, and the process of applying reductions 1–3 is certainly finite. Cases 8,9 and 11 either use a branching with two assignments x and $\neg x$, which is obviously correct, or branchings that are correct by Lemma 2. Case 10 is correct, as any assignment that satisfies C must satisfy at least one of the new clauses. In case 11, the length of the clause containing the 1-literal must be 3, as a 2-clause with a 1-literal $\neg x$ implies that resolution on x is admissible (see Lemma 6). The correctness and completeness of case 12 given that cases 0–11 do not apply is proven in Lemma 15 in the next section, as this proof uses a number of other lemmas, that are best shown in the context of the algorithm analysis. □

4 Analysing the Running Time

This section contains the proof of the upper bound $O(1.1279^{f(F)})$ on the worst-case running time of the algorithm, presented by lemmas roughly following the cases of the algorithm. The section is split into subsections in the following way: Section 4.1 presents the method of analysis and contains some results regarding this and the measure $f(F)$. Section 4.2 deals with cases 0–7 of the algorithm, and gives the basic structural properties that are enforced there. Section 4.3 deals with case 8, where all variables with degrees higher than 3 are handled, and Section 4.4 with case 9, where most of the structure of the problem is enforced. Finally, Section 4.5 deals with the rest of the cases and concludes the proof of the running time of the algorithm. With the help of the structure enforced by case 9, cases 10-11 are easier cases that prepare for the applicability of the CSP construction in case 12.

4.1 Technical Aspects of the Method of Analysis

We use Kullmann's method from [13] to get an upper limit on the running time of *LowdegSAT*. In summary, if F is a fully reduced formula and if *LowdegSAT* applied to F branches into formulas $F[A_1], \ldots, F[A_d]$, let F_i be the fully reduced version of $F[A_i]$ for $i = 1, \ldots, d$. The branching number of this particular branching is $\tau(f(F) - f(F_1), \ldots, f(F) - f(F_d))$, where $\tau(t_1, \ldots, t_d)$ is the unique positive root to the equation $\sum_i x^{-t_i} = 1$. If α is the biggest branching number for all branchings that can occur in *LowdegSAT*, then the running time of the algorithm for a formula F is $O(\alpha^{f(F)})$. For a more detailed explanation, see Kullmann's paper.

As mentioned, we use $f(F) = L(F) - 2N(F) = \sum_{v \in Vars(F)}(d(v,F) - 2)$ as a measure of complexity. While this measure might seem odd at first, there is an intuitive reading of it: $d(v,F) - 2$ is the number of occurrences of v that need to be removed before a simple reduction on v is possible. Thus, it can be viewed as assigning a weight to each variable depending on its degree, so that one variable of degree four counts for as much as two variables of degree three. If F is 3-regular (such as after case 8 of the algorithm), $f(F) = N(F)$, but even for a 3-regular F we use $f(F)$ to guide when we should apply certain reductions and branchings. Furthermore, $f(F)$ obeys all the required properties of a measure, provided that F is free of singletons, as proven in the next lemma.

Lemma 4. *Let F be a CNF formula with $d(x) \geq 2$ for every variable x occurring in F. Let F' be the fully reduced version of F. Then, $f(F) \geq 0$, $f(F) = 0$ if and only if F' is empty, and $f(F') \leq f(F)$.*

Proof. The first is obvious from $f(F) = \sum_{v \in Vars(F)}(d(v, F) - 2)$. For the second, note that the simple reductions never increase the degree of any variable, and that no variable of degree 2 can remain after the simple reductions have been applied. $f(F) = 0$ if and only if all variables in F have degree 2, which implies that all variables will be removed by the simple reductions.

For the final part, $f(F)$ is non-increasing over all simple reductions, by the previous observations, and over standard and backwards resolution whenever these are applied, by the restrictions in the algorithm. To complete the argument, we only need to show that no singletons are created in these steps. For standard resolution, a variable that occurs in only one resolvent must co-occur exactly once with x in F, as both resolvents are non-trivial. For backwards resolution, the only change in the degrees of variables is that the variables in C have their degrees decreased by one. Thus, $f(F)$ is non-increasing over the entire process of reductions. □

Given these properties, we need one more lemma to give a lower bound for $f(F) - f(F')$ when F' is the result of an assignment and reductions on F.

Lemma 5. *Let F be a fully reduced formula, A an assignment to variables of F and F' the reduced version of $F[A]$. Further, let F_0 be the result of a sequence of applications of the reductions in cases 1–3 in any order to $F[A]$. If F_0 is free of singletons, and if F' contains no empty clause, we have $f(F') \leq f(F_0)$.*

Proof. This can be shown by induction on the number of reductions applied. Remember that cases 1–3 only remove clauses and literals from F, and note that the only case of these that will ever remove the last occurrence of a literal from a clause without also removing the entire clause is case 2.

First, if some reduction is applicable on $F[A]$, then every clause and literal that would be removed by the application of this reduction will be removed by any sequence of applications of cases 1–3 ending in a step 3-reduced formula. This can be verified without any great difficulty (using the above observations and the fact that F' contains no empty clause).

Second, assume that the induction hypothesis is true for every sequence of k of these reductions acting on $F[A]$; that is, for any sequence of k applications of cases 1–3 acting on $F[A]$, removing a set of clauses C^* and a set of literals L, every possible sequence of such reductions ending in a step 3-reduced formula will remove at least these clauses and literals. It can be verified without any great difficulty that any extra clauses and literals that would be removed by the application of one further reduction will also be missing in any resulting step 3-reduced formula (again using that F' contains no empty clause).

Thus, if F_1 is the true step 3-reduced version of $F[A]$, then for every variable v that occurs in both F_0 and F_1, $d(v, F_0) \geq d(v, F_1)$. If F_0 is free of singletons, this gives us $f(F_0) \geq f(F_1)$, and the previous lemma gives us $f(F_1) \geq f(F')$. □

Thus, even though we do not know the exact sequence of reductions that will be made after a particular assignment (this is not even defined in the algorithm), this result guarantees that we can safely underestimate the amount of reduction due to cases 1–3. We will permit ourselves to say that a particular chain of reductions occurs, rather than using cumbersome more exact phrases. Calculating $f(F) - f(F_0)$ is easily done using $f(F) = \sum_v (d(v, F) - 2)$.

4.2 Basic Structural Properties

This section contains some results regarding the basic structural properties that exist in a fully reduced formula. First we give a lemma that shows a sufficient condition for when resolution on a variable x is admissible.

Lemma 6. *Let F be a step 5-reduced CNF formula, and x, $d(x) = 3$, be a variable occurring in F such that applying resolution to x increases the degree of at most c variables, while the resolution together with the reductions in cases 1–5 removes or decreases the degree of at least c variables, including x. Then resolution on x is admissible.*

Proof. Use $f(F) - f(F') = \sum_{v \in Vars(F)} (d(v, F) - 2) - \sum_{v \in Vars(F')} (d(v, F') - 2)$, where F' is the step 5-reduced version of $DP_x(F)$. Note that since $d(x, F) = 3$, a variable can increase its degree by at most one in the resolution process. □

Next, Lemmas 7–8 show the mentioned structural properties.

Lemma 7. *If F is a 3-regular, fully reduced formula, and if C, D are two clauses in F, then $|Vars(C) \cap Vars(D)| \leq 2$. If $|C| = 2$, then $|Vars(C) \cap Vars(D)| \leq 1$, so $Vars(C) \not\subseteq Vars(D)$.*

Proof. For the first part, note that some reduction applies both if $l_1, l_2 \in C$ and $l_1, l_2 \in D$, and if $l_1, l_2 \in C$,$\neg l_1, \neg l_2 \in D$. There is no way for C and D to share three variables without one of these cases occurring. For the second part, if $C = (l_1 \vee l_2)$ and $l_1, \neg l_2 \in D$, then case 4 applies and D is shortened. □

Lemma 8. *Let F be a step 5-reduced formula, and let a, b be $(2, 1)$-variables in F. The following structures all guarantee an admissible resolution.*

1. *2-clause C with $\neg a \in C$*
2. *3-clause C with $\neg a, b \in C$ and clause D with $a, b \in D$*
3. *3-clause C with $\neg a, l \in C$, clause D with $a, b \in D$ and 2-clause $(\neg l \vee b)$ for some literal l.*

Proof. In the first two cases, we see immediately by Lemma 6 that resolution on a is admissible. In the third case, we see that one resolvent is either a copy of an existing clause or will be shortened or removed in case 4 at the latest. In either case, $f(F)$ has increased by at most 1 in the resolution process, and at least one simple reduction which strictly decreases $f(F)$ applies, guaranteeing that resolution on a is admissible. □

With these tools, we can prove that all cases 8–12 get a branching number of $\tau(4, 8)$ or better.

4.3 Case 8: Variables of Higher Degree

Here, we prove that the branching number is sufficiently good when branching on any variable x with $d(x) > 3$.

Lemma 9. *If F is a reduced formula with $d(F) > 3$, then applying case 8 of the algorithm results in a branching number of at most $\tau(4, 8)$.*

Proof. For any variable y occurring in any 2-clause with x, we are limited to the following options: $d(y) = 3$ so that the 2-literal y occurs in a 2-clause and x and y have no co-occurrences in any other clauses; $d(y) > 3$ and 2-clauses $(x \vee y), (\neg x \vee \neg y)$ are the only co-occurrences of x and y; or finally $d(y) > 3$ and x and y co-occur in only one 2-clause, say $(x \vee y)$. In the latter case, one longer clause $(\neg x \vee \neg y \vee C)$ for some C can occur. Similarly, for any variable y occurring with x, but not in any 2-clause, we have the following options: $d(y) = 3$ and x and y co-occur only once; $d(y) = 3$ and x and y co-occur only in clauses $(x \vee y \vee C), (x \vee \neg y \vee D)$ (or similarly with $\neg x$), where C and D do not share variables and $|D| > 1$ if $\neg y$ is a 1-literal; or finally $d(y) > 3$, where x and y can co-occur several times as long as the same pair of literals never occurs in more than one clause (in other words, the variable y occurs at most twice with the literal x). From all of this, we can infer the following: The reduction in $f(F)$ in a branch $F[x]$ is at least $d(x) - 2$ plus the number of 2-clauses that contain the variable x plus the contribution from the longer clauses involving the literal x. With only one such clause, this contribution is at least 2. With two such clauses, the contribution is at least 4. With three, the contribution is at least 5, occurring in a situation such as clauses $(x \vee y \vee a), (x \vee z \vee b), (x \vee \neg y \vee \neg z \vee c)$, if $d(v) = 3$ for every involved variable v. Let a be the reduction of $f(F)$ in the x branch, and b the reduction in the $\neg x$ branch. If each literal x and $\neg x$ is involved with at most two clauses longer than a 2-clause, then $a + b \geq 12$ and we need only prove that $a, b \geq 4$. Assume that $\neg x$ has at most as many occurrences as x.

If $\neg x$ is at least a 2-literal, or a 1-literal present in a 3-clause or longer clause, then the result is immediate. If $\neg x$ is a 1-literal present in a 2-clause, say $(\neg x \vee y)$, then the extra assignment $\neg y$ will ensure the result.

The remaining case is that there are three longer clauses containing the literal x, which ensures a reduction of $f(F)$ in the x branch of at least 7 plus the contribution from 2-clauses. If $\neg x$ is at least a 2-literal, then either there are two 2-clauses and a reduction of at least 4 in the $\neg x$ branch, or a reduction of at least 5 in the $\neg x$ branch. If $\neg x$ is a 1-literal, finally, then we shall see that $f(F)$ reduces by at least 5 in the branch $\neg x$. If $\neg x$ occurs in a 3-clause or longer, or in a 2-clause $(\neg x \vee y)$ where $d(y) > 3$, then the immediate assignments reduce $f(F)$ by at least 4 and at least one more variable is affected by the assignments. If $\neg x$ occurs in a 2-clause $(\neg x \vee y)$ with $d(y) = 3$, then y must be a 2-literal, so that $\neg y$ is a 1-literal occurring in a clause of length at least 3. This concludes the proof. □

In every case after this one, F is 3-regular.

4.4 Case 9: Imposing More Structure

We give some conditions under which case 9 of the algorithm applies, and show that the branching number will be at most $\tau(6,6)$.

Lemma 10. *If F is a 3-regular, fully reduced formula, the following statements are true. For the sake of convenience, assume w.l.o.g. that for any variable v, the literal $\neg v$ occurs only once in F.*

1. *Any branch $F[\neg a]$ for a variable a reduces $f(F)$ by at least 6.*
2. *Any branch $F[a]$ for a variable a where the literal a occurs in some clause C with $|C| \geq 5$ reduces $f(F)$ by at least 6.*
3. *If literals a, b occur together in one clause, and $a, \neg b$ occur together in another, then a branch $F[a]$ reduces $f(F)$ by at least 6.*

Proof. 1: Let S be the set of literals that occur in a clause together with $\neg a$ in F. For every literal $l \in S$, $\neg l$ is assigned in the branch. We know that if $l_1, l_2 \in S$, then any clause containing $\neg l_1$ does not contain $\neg l_2$ or a, and a clause C with $\neg l_1, l_2 \in C$ has $|C| > 2$ and $|S| > 2$ if l_1 is a negated literal, $|C| > 3$ if l_1 is an unnegated literal. Either way, each assignment $\neg l_i$ affects at least two literals not from the variables in S.

If $|S| \geq 3$, then at least four variables are assigned in the branch, and at least six literals beyond these are removed from F. By a simple counting argument, this requires at least six variables to be affected.

If $|S| = 2$, let $S = \{l_1, l_2\}$ where l_1, l_2 are some literals for variables b and c, respectively.

If some clause C contains both literal $\neg l_1$ and variable c, then by necessity $l_1 = b$, $l_2 = c$ and $C = (\neg b \vee c \vee C')$ where $|C'| \geq 2$ and C' contains no literals of variables a, b, c. In this case, no clause containing $\neg c$ can be formed without using a sixth variable, by Lemma 7.

Otherwise, any clause containing $\neg l_i$ for $i = 1, 2$ has no other variable in common with the clause containing $\neg a$. We have three further cases, depending on the negations in S.

If $S = \{b, c\}$, then there must exist clauses $(\neg b \vee C), (\neg c \vee D)$ with $|C|, |D| \geq 2$. If less than six variables are affected, $Vars(C) = Vars(D)$ and $|C| = |D| = 2$, but then, either resolution or backwards resolution is admissible on a variable in C. Otherwise, at least six variables are removed in the branch.

If $S = \{b, \neg c\}$, then there exist clauses $(\neg b \vee C)$ with $|C| \geq 2$ and $(c \vee D)$, $(c \vee E)$ with $|D|, |E| \geq 1$. If less than six variables are affected, $|D| = |E| = 1$ and $Vars(C) = Vars(D) \cup Vars(E)$, and by Lemma 8, we must have clauses $(\neg b \vee \neg u \vee \neg v), (c \vee u), (c \vee v)$ for variables u, v. Now, the second appearances of literals u and v must occur in different clauses, where no other literal of the variables a, b, c, u or v can occur. Counting these clauses, at least six variables are removed in the branch.

If $S = \{\neg b, \neg c\}$, then we have clauses $(b \vee A), (b \vee B), (c \vee C), (c \vee D)$, where no case uses only six variables. By Lemma 8 and since case 7 does not apply, we have $A, B \neq C, D$, and by Lemma 7, $Vars(A) \neq Vars(B)$ and $Vars(C) \neq Vars(D)$, so either A–D are all of length one with distinct variables (for a reduction of at least 7 in the branch) or at least one, say C, has $|C| > 1$. In the latter case, D still introduces a variable not in C, for a total reduction of at least 6.

2: Assume w.l.o.g. that $C = (a \vee l_1 \vee l_2 \vee l_3 \vee l_4)$, where l_1–l_4 are literals of variables b–e, respectively. By assumption, there is one more clause D containing literal a, and by Lemma 7, D contains at least one variable other than a–e. At least six variables are affected by the assignment a.

3: By Lemma 8, the clauses can w.l.o.g. be assumed to be $(a \vee b \vee l_1)$, $(a \vee \neg b \vee l_2 \vee l_3)$ where l_1–l_3 are literals of variables c–e. If the reduction in $f(F)$ is less than 6, the second occurrence of literal b must occur in a clause using only these variables. No such clause can exist. □

Lemma 11. *Let F be a 3-regular, fully reduced SAT formula where no condition from Lemma 10 applies. Assume w.l.o.g. that for every variable v, literal $\neg v$ is a 1-literal. Then the following statements hold:*

1. *If there is a clause C with literals a, $\neg b$ and $\neg c$ for some variables a, b, c, then a branch $F[a]$ reduces $f(F)$ by at least 6.*
2. *If there is no such clause, but there is a clause C with literals a and $\neg b$ for some variables a, b, then a branch $F[a]$ reduces $f(F)$ by at least 6.*

Proof. 1: Assignments b and c will be made. By the various restrictions on F, it can be verified that whether $C = (a \vee \neg b \vee \neg c)$ or $C = (a \vee \neg b \vee \neg c \vee l_1)$, at least six variables are affected.

2: Assignment b will be made, and $|C| \geq 3$. By similar arguments as before, considering both clauses that contain the literal a as well as the clauses containing literal b, at least six variables are affected. □

We see that for any F where none of cases 0–9 apply, we have a specific structure where every clause C contains either only 2-literals, in which case $2 \leq |C| \leq 4$, or only 1-literals, in which case $|C| \geq 3$. Additionally, every pair of variables co-occurs in at most one clause.

4.5 The Final Cases

Given the structure imposed by case 9, showing the rest of the results is relatively easy. Case 10 imposes a stricter limit on the length of a clause with 1-literals, case 11 gives us stronger guarantees on the neighbourhood of a 2-literal, and finally, if all other cases fail to apply, case 12 can be applied to convert the formula to an instance of $(3, 2)$-CSP.

Lemma 12. *Let F be a SAT formula where case 10 is the earliest case of the algorithm LowdegSAT that applies. The branching number for this case is at least $\tau(6, 6)$.*

Proof. Let C be the clause that is being split. For any literal $l_i \in C$ that is not included in the new clause, $\neg l_i$ becomes a pure literal, so that an assignment $\neg l_i$

is made. For each such assignment, two literals for other variables are affected. If there are at least three such assignments $\neg l_i$, we have at least six additional literals, and by a counting argument at least six variables are affected in total.

If only two literals become pure, say a, b, let S_i for $i = 1, 2$ be the set of literals v such that v occurs in i clauses together with literal a or b. Assume w.l.o.g. that $S_1 = \{u_1, \ldots, u_d\}$ and $S_2 = \{v_1, \ldots, v_e\}$. $|S_1| + 2|S_2| \geq 4$, and for every literal $l \in S_2$ an additional assignment $\neg l$ is made. We trace these assignments.

First, if $S_2 = \emptyset$, the reduction in $f(F)$ is at least $2 + |S_1| \geq 6$.

Second, if $|S_2| = 1$, $|S_1| \geq 2$. Let D be the clause where $\neg v_1$ occurs. If less than six variables are to be removed, $D = (\neg u_1 \vee \neg u_2 \vee \neg v_1)$, but then u_1 and u_2 are assigned and must lie in different clauses, which requires extra variables.

Third, if $|S_2| = 2$, then some literal $\neg w$ shares a clause with some $\neg v_i$, and assignment w is made. At least one occurrence of w is in a clause with some new variable, for a reduction of at least 6.

Finally, $|S_2| \geq 3$. If $|S_2|+|S_1| > 3$, then the reduction is at least 6. Otherwise, some extra variable is required to form a clause with $\neg v_i$. □

Lemma 13. *Let F be a CNF formula such that no case before case 11 of LowdegSAT applies. Let a be a variable. W.l.o.g., assume that literal $\neg a$ occurs once in F. Then, if a is a member of k 2-clauses, an assignment $F[\neg a]$ reduces $f(F)$ by at least $7 + k$.*

Proof. Let the clause that contains $\neg a$ be $(\neg a \vee \neg b \vee \neg c)$, so that assignments b and c are made. Each 2-clause containing a, b or c contributes one variable, and if there are l literals otherwise removed from F, these literals belong to at least $l/2$ variables. Since no clause contains both b and c, the result follows from this. □

Lemma 14. *If F is a CNF formula such that case 11 is the earliest case of LowdegSAT that applies, then the branching number is at most $\tau(4, 8)$. If case 11 does not apply either, then every 2-literal l is involved in two 2-clauses.*

Proof. If a is part of no 2-clauses, then the number of variables affected by assignment a is at least 5, which by Lemma 13 leads to a branching with a branching number of at most $\tau(5, 7) < \tau(4, 8)$. If literal a is neighbour to only three other variables, a must be involved in one 2-clause, and by the same lemma, we have a branching with a branching number of at most $\tau(4, 8)$. The remaining case, with only two other variables, can only be achieved by two 2-clauses. □

Lemma 15. *If F is a CNF formula such that no case among cases 0–11 of LowdegSAT applies to F, then the construction in Lemma 1 is applicable, and the total time for LowdegSAT(F) is $O(1.3645^{N(F)/3}) \subset O(1.1092^{f(F)})$.*

Proof. In addition to the structural properties noted previously, we have by case 10 that $|C| = 3$ for every clause C with 1-literals and by case 11, as noted in Lemma 14, $|C| = 2$ for every clause C with 2-literals, which proves the applicability of the construction. Eppstein's algorithm [8] runs in time $O(1.3645^n)$, and the resulting CSP instance has $N(F)/3$ variables. With $f(F) = N(F)$ at this point in the algorithm, we get the described running time. □

This concludes our sequence of lemmas. We will proceed with the main theorem.

Theorem 1. *If F is a CNF formula without singletons, then LowdegSAT(F) decides the satisfiability of F in time $O(1.1279^{L(F)-2N(F)})$.*

Proof. This follows from the various relevant lemmas. $\square$

Corollary 1. *If F is any CNF formula where the degree of a variable x is limited to at most d, the running time is in $O(1.1279^{(d-2)\cdot N(F)})$.*

Proof. All singletons will be removed in simple reductions before any branching is done. If F' is the step 3-reduced version of F, then $d(v, F') \leq d$ for every $v \in Vars(F')$ and $L(F') - 2N(F') \leq (d-2)\cdot N(F') \leq (d-2)\cdot N(F)$. $\square$

5 Conclusions

We have presented an algorithm which decides the satisfiability of a CNF formula F, with $N(F)$ variables and of length $L(F)$, in time $O(1.1279^{L(F)-2N(F)})$ if F is free of singletons. This implies a running time of $O(1.1279^{(d-2)N(F)})$ when F is either a formula with at most d occurrences for any variable or a singleton-free formula with d occurrences per variable on average. For $d \leq 4$, this is better than previous results.

No previous algorithms have been published for SAT problem instances where the number of occurrences per variable is limited. Looking at other NP-hard problems, we find only a few papers where similar attacks have been made. In [2], Chen et al. apply advanced methods of algorithm analysis to get an algorithm that solves the Vertex Cover problem for a graph of maximum degree 3 in parameterised time $O(1.194^k k + n)$ for a maximum cover size of k, and in time $O(1.1255^n)$ for the non-parameterised version. Closer to the work in this paper, one of the helper algorithms for the problem of counting max-weight models for 2SAT formulae in [3] is an algorithm that runs in time $O(1.1892^{N(F)})$ when $d = 3$. This helper algorithm is then used to extend this into running times of $O(1.2400^{N(F)})$ when $d = 4$ and $O(1.2561^{N(F)})$ for the general problem.

For future research, it could be fruitful to apply techniques similar to or inspired by those in [3] to extend the results in this paper to an algorithm that is more effective when $d > 4$.

References

[1] Tobias Brueggemann and Walter Kern. An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science*, 329(1–3):303–313, 2004.

[2] Jianer Chen, Iyad A. Kanj, and Ge Xia. Labeled search trees and amortized analysis: Improved upper bounds for NP-hard problems. In *Proceedings of the 14th Annual International Symposium on Algorithms and Computation (ISAAC 2003)*, pages 148–157, 2003.

[3] Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström. Counting models for 2SAT and 3SAT formulae. *Theoretical Computer Science*, 332(1-3):265–291, 2005.

[4] Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.

[5] Evgeny Dantsin, Edward A. Hirsch, and Alexander Wolpert. Algorithms for SAT based on search in Hamming balls. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, volume 2996 of *Lecture Notes in Computer Science*, pages 141–151. Springer, 2004.

[6] Evgeny Dantsin and Alexander Wolpert. Derandomization of Schuler's algorithm for SAT. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, pages 69–75, 2004.

[7] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[8] David Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms (SODA 2001)*, pages 329–337, 2001.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[10] Edward A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.

[11] Kazuo Iwama and Suguru Tamaki. Improved upper bounds for 3-SAT. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, page 328, 2004.

[12] Paul Walton Purdom Jr. Solving satisfiability with less searching. In *IEEE Transactions on Pattern Analysis and Machine Intelligence. PAMI-6*, pages 510–513, jul 1984.

[13] Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223:1–72, 1999.

[14] Uwe Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.

[15] Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of Algorithms*, 54(1):40–44, 2004.

[16] Stefan Szeider. Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. In *Proceedings of the 9th Annual International Conference on Computing and Combinatorics (COCOON 2003)*, pages 548–558, 2003.

[17] Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8:85–89, 1984.