A Tighter Bound for Counting Max-Weight Solutions to 2SAT Instances

Magnus Wahlström

wahl@mpi-inf.mpg.de
Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. We give an algorithm for counting the number of max-weight solutions to a 2SAT formula, and improve the bound on its running time to $\mathcal{O}(1.2377^n)$. The main source of the improvement is a refinement of the method of analysis, where we extend the concept of compound (piecewise linear) measures to multivariate measures, also allowing the optimal parameters for the measure to be found automatically. This method extension should be of independent interest.

1 Introduction

From a computational complexity point of view, the problem class #P of problems where you want to know the number of solutions to some problem in NP is a very difficult one. The class was proposed by Valiant in the 1970's [14], and it was later proved that the so-called polynomial hierarchy is contained in $P^{\#P}$ [12] (i.e. that a polynomial-time algorithm for any #P-complete problem would allow us to solve any problem in the polynomial hierarchy in polynomial time; in fact, a single query to the algorithm would suffice). #P-complete problems include the counting counterparts of both NP-complete problems such as 3SAT (counting counterpart #3SAT) and problems that are in P. The problem considered in this paper, #2SAT, is an example of the latter: the "decision variant" 2SAT is a well-known polynomial problem, while the counting version #2SAT is #P-complete [9, 13]. However, despite the apparent difficulty of the class, individual #P-complete problems can be solved in reasonable exponential time; for instance, the bound $\mathcal{O}^*(1.2377^n)$ [1] for #2SAT is significantly faster than any bound for solving 3SAT (for which the best bounds are a probabilistic algorithm with a bound of $\mathcal{O}^*(1.3238^n)$ by Iwama and Tamaki [8], and a deterministic algorithm with a bound of $\mathcal{O}^*(1.473^n)$ by Brueggemann and Kern [1]).

One of the first algorithms for a counting problem came in the early 1960's with Ryser's [11] $O(n^2 2^n)$ time algorithm for counting the number of perfect matchings in a bipartite graph (also known as computing the *permanent* of a 0/1 matrix). Previous work on the #2SAT problem with better bounds than $\mathcal{O}^*(2^n)$ includes results by Dubois [4], Zhang [17], Littman *et al.* [10], Dahllöf, Jonsson and Wahlström [2, 3], and Fürer and Kasiviswanathan [7]. In terms

[1] $\mathcal{O}^*(\cdot)$, $\Theta^*(\cdot)$, etc, signify that polynomial factors are ignored

of the actual bounds, the bound $\mathcal{O}^*(1.3247^n)$ appeared in [2]; later, a complete rewrite of the algorithm for the journal version produced the bound $\mathcal{O}^*(1.2561^n)$ [3], where the method of compound measures was introduced (under the name of piecewise linear measures); and the previously best bound $\mathcal{O}^*(1.2461^n)$ [7] was produced by a more detailed version of the analysis in [3].

The bound $\mathcal{O}^*(1.2377^n)$ produced in this paper is the result of a further improvement of the analysis through compound measures, this time introducing multi-variate compound measures, which are a combination of compound measures with the multi-variate recurrences of Eppstein [5]. Analysis through compound measures allows us to model that the behaviour of the algorithm varies depending on certain parameters on the instance, in this case the average degree, i.e. that the behaviour of the algorithm is non-uniform. Apart from earlier #2SAT$_w$ publications, this type of analysis has been applied to the problem of finding a solution to SAT instances F with a bounded $\ell(F)/n(F)$ value [15]. Combining the method with Eppstein's method for solving multi-variate recurrence improves the quality of the bound, and allows us to automate the bound calculations.

The paper is structured into Sect. 2 on preliminaries, Sect. 3 describing the improved compound measures, Sect. 4 defining the problem precisely and giving the algorithm, Sect. 5 providing the analysis of upper bounds for maximum degree four, and finally Sect. 6 containing the general upper bound.

Major portions of this paper appeared as Chapt. 7 in the author's thesis [16], but the material has not been published in any referreed publication. Some proofs have been omitted due to lack of space; these can usually be found in [16].

2 Preliminaries

A variable, in this paper, can take the values true or false, referred to as 1 resp. 0; a *literal* of a variable is either the unnegated literal v, having the same truth value as the variable, or the negated literal $\bar{v}$, with the opposite truth value; a *clause* is a disjunction of $(a \vee b)$ of literals, referred to as a *k-clause* if it is a disjunction on k literals; and a 2SAT formula $F = (a \vee b) \wedge (\bar{a} \vee c) \wedge \ldots$ is a conjunction of clauses where every clause contains at most two literals. A *model* for a 2SAT formula F is an assignment to all its variables that satisfies the formula (i.e. at least one literal in every clause is true). When we write $\tilde{v}$, this refers to either literal v or $\bar{v}$.

We talk of the *graph* of a 2SAT formula F. This is mainly an analogous term: we consider a graph where we have one vertex for every variable in F, and one edge (a, b) for every 2-clause $(\tilde{a}, \tilde{b})$ in F, where $\tilde{v}$ is v or $\bar{v}$, and let terms such as *connected component* and *subgraph* hold the meaning they would have in this graph. Therefore, the *degree* $d(x)$ of a variable x is the number of clauses where it occurs, and $d(F)$ is the maximum degree of any variable in F. $Var(F)$ is the set of variables occurring in F, $n(F) = |Var(F)|$ is the number of variables, and $n_i(F)$ is the number of variables of degree i. A variable of degree i is called an

i-variable; a 1-variable is rather called a singleton. A *heavy* variable has degree at least three. We also use $\ell(F)$ for the total length of F, i.e. $\ell(F) = \sum_i i n_i(F)$.

Other graph terms used in the paper include the concept of a *neighbourhood*: for a variable v, the *(open) neighbourhood* $N(v)$ of v is the set of all variables w such that there exists a 2-clause $(\tilde{v} \vee \tilde{w})$ in E (i.e. the definition is identical to the neighbourhood of the vertex v in the graph of the formula). The *closed neighbourhood* $N[v]$ is defined as $N(v) \cup \{v\}$.

3 On Analysis by Compound Measure

A *complexity measure* $\mu(F)$ is a function assigning a non-negative complexity value to any possible instance F of some problem. They are used for estimating upper bounds on the running times of branching algorithms; see e.g. the survey by Fomin et al. [6].

For a branching algorithm and a complexity measure $\mu(F)$, the *branching number* of a particular branching from an instance F to subinstances $F_1, \ldots, F_k$, with $\mu(F) - \mu(F_i) = \delta_i$, is $\tau(\delta_1, \ldots, \delta_k)$, defined as the unique positive root of $\sum_i x^{-\delta_i} = 1$.

We say that $\mu(F)$ is a *well-behaved* measure for the algorithm if the following hold:

1. $\mu(F) \geq 0$ for all possible F;
2. $\mu(F) = 0$ only for cases the algorithm solves in a given polynomial time;
3. $\mu(F') \leq \mu(F)$ if the algorithm, when applied to F, may apply a reduction replacing F by F'; and
4. $\mu(F') < \mu(F)$ if the algorithm, when applied to F, may perform a branching where F' is one of the branches.

If this is the case, then the running time of the algorithm will be in $\mathcal{O}^*\left(c^{\mu(F)}\right)$, where c is the largest possible branching number that can occur in the algorithm. A *hard case* is a case with a branching number identical to the maximum.

Under a model of an algorithm as a set of possible branchings, of which any branching can be applied at any time, and of an instance as a point in $\mathbb{Z}^d$ (i.e. a set of integer attributes, e.g. $n_i(F)$ for $i \leq d$), Eppstein [5] gives a method for finding a *weight-based measure* $\mu(I) = \mathbf{w} \cdot I$ (e.g. $\mu(F) = \sum_i w_i n_i$) such that the resulting bound for the worst-case behaviour is tight to within a polynomial factor (under a further restriction that the instance agrees with a certain *target vector* t; see the paper for details).

However, for the algorithm in this paper (and for other algorithms as well), the basic assumption of the model is false: not all branchings can be used in all cases. In this paper, we use a model where along with every branching B that could possibly be a hard case in the above sense, we associate a *highest average degree* p_B, such that if $\ell(F)/n(F) \geq p_B$, then the branching B will not be used (since better cases are found). We analyse this using *compound measures*. Compound measures were used in previous publications for $\#2\mathrm{SAT}_w$

[3, 7], but here, we introduce multi-variate compound measures, and show how to use Eppstein's method to find the optimal weights for them.

Our compound measures in this paper follow the pattern

$$\mu(F) = \begin{cases} \mu_0(F) & \text{If } \ell(F)/n(F) \leq p_0 \\ \mu_1(F) & \text{If } p_0 < \ell(F)/n(F) \leq p_1 \\ \cdots & \\ \mu_t(F) & \text{If } \ell(F)/n(F) > p_{t-1} \end{cases}$$

where p_i is an increasing sequence selected from the p_B, referred to as the *pivot points*, and each $\mu_i(F)$ is a weight-based measure, specifically $\mu_i(F) = \sum_j w_{i,j} n_j(F)$; our measure $\mu(F)$ is *piecewise linear*. The region from p_i to p_{i+1} is referred to as *section* p_i to p_{i+1}. Two additional things must hold for μ:

$$\mu_i(F) \geq \mu(F) \text{ for every possible } F, i \tag{1}$$
$$\mu_i(F) = \mu_{i+1}(F) \text{ when } \ell(F)/n(F) = p_i \tag{2}$$

(i.e. the measure is concave and continuous). Note that by (1), $\mu(F) - \mu(F') \geq \mu(F) - \mu_i(F')$ for any i, so we do not need to know the value of $\ell(F')/n(F')$ to find $\Delta\mu(F)$. By this, we can derive

$$\mu_{i+1}(F) = \mu_i(F) - \alpha_i(\ell(F) - p_i n(F))$$

for $\alpha_i \geq 0$; α_i is the *pivot amount* at pivot point i.

Note that for the worst-case behaviour, $\mu_i(F)$ for $i < t$ is of only secondary interest, since $\ell(F)/n(F) \leq p_i(F)$ must hold, limiting the possible number of highest-degree variables. What we are interested in is $\mu_t(F)$, and the values of α_i that show that our bound is correct. The values of $w_{i,j}$ can be derived from this:

$$w_{t-k,j} = w_{t,j} + \sum_{i=t-k}^{t-1} (j - p_i)\alpha_i.$$

Now, for every branching B, we have one case for every measure μ_i as long as $p_B \geq p_i$, and this case can be expressed in terms of $w_{t,j}$ and α_i using the previous formula. Finally, select the target vector to be the situation $n_d(F) = n(F)$ for the maximum considered degree d.

Lemma 1. *The adaption to Eppstein's method that is described above will produce the best possible compound measure for a given set of degree-bounded branchings* (B, p_B) *and preselected pivot points* p_i*.*

Proof. There is a one-to-one correspondence between values of $w_{t,j}$ and α_i on the one hand, and all values $w_{i,j}$ on the other, so the optimality guarantee of Eppstein is enough. □

Finally, we point at two things that are missing. First, we have not described how to select pivot points. In general, we have no answer, but for the branchings

in this paper, where $(\Delta\ell(F))/(\Delta n(F)) \geq p_B$ for every branch of every branching B, it can be shown that only the values p_B need to be used. Second, there is no lower bound guaranteeing optimality of the resulting bound relative to the model we have described.

4 Problem Definition and Algorithm

In its basic form, the problem #2SAT consists of a 2SAT instance F, and the question is how many solutions F has. Here, we extend the problem with weights, and ask for the number of solutions with maximum solution weight (i.e. the number of solutions with weights identical to the maximum possible weight of any solution to F)[2]: each instance consists of a 2SAT formula F along with a *weight vector* $\mathbf{w}$ and a *cardinality vector* $\mathbf{c}$, assigning a weight $w(\tilde{v})$ resp. a cardinality $c(\tilde{v})$ to each literal $\tilde{v}$ of each variable v occurring in F. The weight of a model M of F is

$$\mathcal{W}(M) = \sum_{l \text{ is true in } M} w(l);$$

that is, the sum of $w(v)$ for every variable that is true in M and $w(\bar{v})$ for every variable that is false in M. In the same manner, let the cardinality of a model M be

$$\mathcal{C}(M) = \prod_{l \text{ is true in } M} c(l).$$

Again, l ranges over literals, not over variables. The solution to the instance is the maximum weight of any model M for F, and the sum of the cardinalities of all true models M of F.

We use cardinality vectors in order to enable the use of multiplier reduction (described below); the most natural instances to the problem may be where the cardinality of every literal is 1, so that we ask simply for the number of max-weight models.

Because of the weight and cardinality vectors, there is some bookkeeping involved in performing an assignment to a formula and propagating its direct implications (e.g. from a 2-clause $(a \vee b)$ and an assignment $a = 0$, a further assignment $b = 1$ is derived). We will understand the term *recursively branch on* x to include handling all details of weight, cardinality, and propagation; thus, we will also always assume that the formula we are dealing with contains only 2-clauses. A precise description, if required, can be found in earlier publications [3, 16].

Our main algorithm $\mathrm{C}(F, \mathbf{c}, \mathbf{w})$, taking a formula F, a cardinality vector $\mathbf{c}$, and a weight vector $\mathbf{w}$, is defined as Algorithm 1 later in this section. All

[2] The most immediate uses of the algorithm are probably to either find the number of solutions in total, or the maximum possible weight of a solution (both of which being problems for which this algorithm is the fastest known), but note that the max-weight requirement can be used to implement non-2SAT constraints, e.g. the gadget $(\bar{x} \vee \bar{u}), (y \vee \bar{u}), (\bar{x} \vee \bar{v}), (\bar{y} \vee \bar{v}), (z \vee \bar{v})$ with $w(l) = 1$ iff $l \in \{x, u, v\}$, where u and v only occur in these clauses, implements $(x \vee y \vee z)$.

references to $\mathrm{C}(\cdots)$ in the following definitions are references to this algorithm. First, we give a definition that is used when selecting a variable to branch on.

Definition 1. *In a formula F with average degree $\ell(F)/n(F) = k$, the* associated average degree *of a variable x in F is $\alpha(x)/\beta(x)$, where:*

$$\alpha(x) = d(x) + |\{y \in N(x) \mid d(y) < k\}| \tag{3}$$

$$\beta(x) = 1 + \sum_{\{y \in N(x) \mid d(y) < k\}} 1/d(y) \tag{4}$$

The following lemma shows the use of this definition in the algorithm.

Lemma 2 (Lemma 6 of [3]). *Let F be a non-empty formula such that $\ell(F)/n(F) = k$. There exists some variable $x \in Var(F)$ with $d(x) \geq k$ with associated average degree at least k.*

We also use a reduction called *multiplier reduction*. When a formula F consists of two parts F_1 and F_2, with $|Var(F_1) \cap Var(F_2)| = 1$ (say, the variable v), then multiplier reduction applies. We then need only two pieces of information from F_2: the number and weight of max-weight solutions when $v = 1$ resp. $v = 0$. We can find these numbers through calls to $C(F_2[v = 1])$ and $C(F_2[v = 0])$ and incorporate them into $w(\tilde{v}), c(\tilde{v})$, to then proceed to calculate $C(F_1)$ with the modified values. This will be referred to as *removing F_2 by multiplier reduction*.

Finally, we provide the algorithm. Note that though the analysis is split into several parts, using different measures, these parts are only different ways of analysing this same algorithm.

Algorithm 1 $\mathrm{C}(F, \mathbf{c}, \mathbf{w})$:

1. If $F = \emptyset$, then return $(1, 0)$. If $\emptyset \in F$, then return $(0, 0)$.
2. If F is not connected, then return (c, w) where $c = \prod_{i=0}^{j} c_i$, $w = \sum_{i=0}^{j} w_i$, and $(c_i, w_i) = \mathrm{C}(F_i, \mathbf{c}, \mathbf{w})$ for the connected components $F_0, \ldots, F_j$.
3. If multiplier reduction applies, then apply it, removing the lightest part (as measured by the complexity measure).
4. If $d(F) \in \{3, 4\}$, then let x be a variable of maximum degree, secondarily maximising the associated average degree $\alpha(x)/\beta(x)$.
   (a) If there exists a set of two heavy variables $\{y, z\}$, $y, z \notin N[x]$, whose removal leaves F disconnected and leaves $N(x)$ in a non-heaviest component, then recursively branch on y.
   (b) Otherwise, recursively branch on x.
5. Let x be a variable of maximum degree, which if possible does not have only neighbours of degree $d(x)$, and recursively branch on x.

5 Analysis: Maximum Degree 4

In this section, we will give upper bounds for the running time of the algorithm in cases where $d(F) \leq 4$. The bounds of this section are given using the method described in Sect. 3. We begin with an observation for the case $d(F) = 2$.

Lemma 3. *The algorithm C applied to a formula F with $d(F) \leq 2$ runs in polynomial time.*

Degrees of neighbours	Highest average degree	Branching (case 4b)
$(2, 2, 2)$	$6/2.5 = 2.4$	$\tau(12w_l + 4w_n, 12w_l + 4w_n)$
$(2, 2, 3)$	$5/2 = 2.5$	$\tau(10w_l + 3_w n, 18w_l + 6w_n)$
$(2, 3, 3)$	$4/1.5 \approx 2.67$	$\tau(8w_l + 2w_n, 16w_l + 5w_n)$
$(3, 3, 3)$	$3/1 = 3$	$\tau(6w_l + w_n, 16w_l + 4w_n)$

Table 1. Possible cases when branching on a 3-variable.

We will now give the bound for the case $d(F) = 3$. For reference, and to illustrate the process, the possible neighbourhoods of a heavy variable, with their respective average degree guarantees, are given in Table 1. The measure is based on the attributes $\ell(F)$ and $n(F)$, rather than $n_2(F)$ and $n_3(F)$, since they are equivalent when there are only two attributes, and the former is somewhat easier to work with.

Section	w_l	w_n	Time
2 to 2.4	0.25	-0.5	$\mathcal{O}^*\left(2^{0.1n}\right) \subset \mathcal{O}^*\left(1.0718^n\right)$
2.4 to $2 + 2/3$	0.185373	-0.344895	$\mathcal{O}^*\left(2^{0.1495n}\right) \subset \mathcal{O}^*\left(1.1092^n\right)$
$2 + 2/3$ to 3	0.155985	-0.266527	$\mathcal{O}^*\left(2^{0.2015n}\right) \subset \mathcal{O}^*\left(1.1499^n\right)$

Table 2. Component measures $w_l\ell(F) + w_n n(F)$ for maximum degree 3

Lemma 4. *For a formula F with $d(F) \leq 3$, algorithm C runs in $\mathcal{O}^*\left(1.1499^n\right)$ time.*

Proof. The components of the compound measure for this case are on the form $\mu_a(l, n) = w_l l + w_n n$, with the parameters of the measures given in Table 2. It may seem strange that $w_n < 0$ for these components, but this can be translated into the form $\sum_i w_i n_i$ with $w_i = iw_l + w_n$, in which case $w_i \geq 0$ for every $i \geq 2$, which also shows that $\mu_a(l, n)$ is non-increasing over every reduction (and since $\mu_a(F) \geq \mu(F)$, so is $\mu(F)$). For cases 2 and 3 of the algorithm, note that since

μ_a is linear, $\mu(F) = \mu_a(F) = \sum_i \mu_a(F_i) \geq \sum_i \mu(F_i)$; the time used is dominated by the time spent on the heaviest component.

Also, when estimating Δf, this means that our underestimations are safe unless the formula we compare against contains a singleton (since $w_1 < 0$ for some sections of f). Specifically, Δf can be described as w_2 for every removed 2-variable, w_3 for every removed 3-variable, and w_l for every variable that has had its degree reduced from 3 to 2.

Next, consider case 4a. We can see that in both branches we will have removed all of $N[x]$ plus the variable y, and at least two heavy variables will have been reduced to light variables (since multiplier reduction does not apply). Both branches get a reduction of at least $(S(x) + 5)w_l + 5w_n$, where $S(x) = \sum_{y \in N[x]} d(y)$, which will compare favourably to the results of using case 4b, and will never result in a worse branching.

Section	w_2	w_3	w_4	Time
2–3	0.045443	0.201428	0.324788	$\mathcal{O}^*(1.1499^n)$
3–3.2	0.084777	0.201428	0.285454	$\mathcal{O}^*(1.1634^n)$
3.2–3.5	0.092882	0.202779	0.280051	$\mathcal{O}^*(1.1822^n)$
3.5–3.75	0.097593	0.204349	0.278481	$\mathcal{O}^*(1.1975^n)$
3.75–4	0.107950	0.208788	0.277001	$\mathcal{O}^*(1.2117^n)$

Table 3. Component measures $\sum_i w_i n_i(F)$ for maximum degree 4

The worst cases of the algorithm remain. These branchings are given in Table 1, in terms of generic weights w_l and w_n, since they do not depend upon the particular measure associated with the current section, and the branchings can be verified without great difficulty. Using the measures from Table 2, it can be verified that every branching has a branching number of at most 2 in a section where it is applicable (e.g. the first measure gives branching numbers of at most 2 for every case, while in the final section up to 3, only the 3-regular case is applicable).

We see that the time is indeed in $\mathcal{O}^*\left(2^{\mu_3(F)}\right)$ for the $\mu_3(F)$ given in Table 2, and the total worst time is $\mathcal{O}^*(1.1499^n)$, as given in the table. □

Now, we present the analysis of the case when $d(F) = 4$. For this case, the multiple attributes-version of the analysis is used, with component measures $\sum_i w_i n_i(F)$, as explained in Sect. 3. We use $\Delta w_i = w_i - w_{i-1}$ to simplify expressions of branchings. The weights of the measure are given in Table 3. These weights were calculated automatically according to the approach described in Sect. 3, with resulting non-zero pivot points at average degrees 3, 3.2, 3.5, and 3.75 (the amount of pivot at the other potential pivot points was found to be zero in an optimal solution). The component measure for section 2–3 coincides with the top-most component measure for $d(F) = 3$: $0.155985\ell(F) - 0.266527n(F)$ results in $w_2 = 2w_l - w_n = 0.045443$ and $w_3 = 3w_l - w_n = 0.201428$. The auto-

matic weight calculation also guarantees that the choice of weights and pivoting strategy is optimal. The bound achieved for $d(F) = 4$ is $\mathcal{O}^*(1.2117^n)$.

Lemma 5. *The weights of Table 3 form a correct compound measure.*

Lemma 6. *For a formula F with $d(F) = 4$, $C(F)$ runs in time $\mathcal{O}^*(1.2117^n)$.*

Proof. We refer again to Table 3 for a definition of the weights in the compound measure. The measure is clearly well-behaved. The branching depends on the neighbourhood of the variable that is chosen; the explicit table has been cut, but the cases are similar to those for $d(F) = 3$ (and easier to find). We will prove that these are the worst-case branchings shortly, but first we consider case 4a: if case 4a is used, then $N[x]$ and y are removed in both branches, and at least two variables decrease their degree, which can be adjusted for the parity of $\ell(F)$. In the heavy branch of the maximally unbalanced branching, $N[x]$ is removed and at least three variables get their degrees decreased, likewise adjusted for the parity of $\ell(F)$. We see that in case 4a, both branches will be at least as heavy as the heaviest possible branch of case 4b. Thus, we only consider case 4b in the following.

When removing only the variable x and repeatedly applying cases 1 and 2 of the algorithm, if any variable gets its degree reduced to 0[3] or ends up in a non-biggest connected component (even if this happens after subsequent applications of case 2), then x is a cut-vertex and multiplier reduction applies to F. Also, obviously, if any variable gets its degree reduced to 1, then multiplier reduction applies and one more reduction of the degree of some variable occurs. Thus, when x has k light neighbours and only x is removed in one branch, the total reduction is at least $w_4 + kw_2 + k\Delta w_4$ plus the reductions in degree of the other neighbours of x, for the light branch of the maximally unbalanced branching. When some other variable is assigned, this does not apply, though we do know that in total, there are at least three variables outside of $N[x]$ that have links to variables in $N(x)$.

The case when the neighbours of x are not all removed in the same branch does not provide a hard case; this is in agreement with the general principle that $\tau(a - d, b + d) > \tau(a, b)$ when $a \leq b$ and $d \geq 0$, and can be proven by going through the cases in a simple though lengthy manner.

All that remains are the cases of a neighbourhood of degrees (d_1, d_2, d_3, d_4) where in one branch only the branching variable disappears (producing $\Delta\mu = w_4 + \sum_i \Delta w_{d_i}$) and in the other, $N[x]$ disappears (producing $\Delta\mu = \sum_{y \in N[x]} w_{d(y)} + k\Delta w_4$, for $k = 3$ or $k = 4$ according to the parity of the sum of degrees of $N[x]$). Again, it can be verified that each such case has a branching number of at most 2 in every section where it is applicable. The total worst-case time for the $d(F) = 4$ case, as stated, is $\mathcal{O}^*(2^{w_4 n})$ for the final value of $w_4 = 0.277001$, or $\mathcal{O}^*(1.2117^n)$. □

[3] By "reduced to 0" we mean that all neighbours of the variable are removed, and we do not include when a variable is removed by multiplier reduction.

w_2	w_3	w_4	w_5	w_6
0.115507	0.208788	0.277001	0.301245	0.307612

Table 4. Weights for $d(F) > 4$ analysis

6 Analysis: The General Case

With $d(F) > 4$, the effects of a changing average degree seem to be less important than the number of variables removed. The analysis is performed in terms of a standard weight-based measure $\mu(F) = \sum_i w_i n_i(F)$, whose weights are given in Table 4. Note that while the values of w_3 and w_4 are the same as in the topmost measure for the $d(F) = 4$ analysis, the value of w_2 is increased to get a better worst-case branching number. This inequality is no problem, since the degree of a variable never increases by the application of a reduction: once $d(F) < 5$, the case $d(F) > 4$ does not appear in any subinstance.

The hard cases will be one case with a smallest-possible neighbourhood ($d(x) = 5$ and $N(x)$ is 2-regular), and the two cases with biggest-possible neighbourhoods (for $d(x) = 5$ and $d(x) = 6$). Since the latter two cases have average degree limits of 5 resp. 6, a compound measure would not be the right tool for this analysis.

Lemma 7. *Using $\mu(F) = \sum_i w_i n_i(F)$ with the weights given in Table 4, the running time of C for a formula F with $d(F) \leq 6$ is in $\mathcal{O}^*\left(2^{\mu(F)}\right)$.*

Proof. If $d(F) < 5$, then see Sect. 5 (note that the weights $w_2, \ldots, w_4$ give a bound that is consistent with that for section 3.75–4, which is in turn a valid bound for all cases with $d(F) \leq 4$). As before, the application of case 4a guarantees a reduction in both branches that is at least as high as the reduction in the heavy branch of the maximally unbalanced branching, and all cases with a branching number of 2 appear in case 4b with the maximally unbalanced branching. Providing a list of all branching numbers would require several pages; such lists are omitted for space, but the claim can be verified by a simple computer program. The cases of k-regular neighbourhoods with $d(x) = k$ are avoided as far as possible in case 5 of the algorithm, and as a result these cases happen at most once each in every path through the branching tree: they only apply if the k-variables form a regular connected component, and since no reduction creates a new occurrence of any variable in the formula, any k-regular connected component that appears in some subsequent subcase of some k-regular formula F must occur as a subformula in F, which is impossible. Since these cases occur at most once in every path of the tree, they contribute only to the polynomial part of the running time.

No case with a more balanced branching has a higher branching number than 2; the proof for this is also omitted. We see that all cases have a branching number of at most 2. □

Theorem 1. *The algorithm C counts the number of max-weight models for a formula F in time* $\mathcal{O}^*(1.2377^n)$.

Proof. If $d(F) \leq 6$, then this follows from Lemma 7. Otherwise, we can perform a quick analysis in terms of $n(F)$: the measure $n(F)$ is a well-behaved measure for the algorithm and since $d(F) \geq 7$, the branching number for case 5 is at worst $\tau(1,8) < 1.2321$. □

Finally, we make a brief note on lower bounds for the algorithm. While it would be good as a reference point to have strong lower bounds to match the upper bounds on the algorithm's running time, e.g. through presenting a class of instances for which we can prove that the algorithm, through making poor choices compatible with the algorithm description, can require $\Omega^*(c^n)$ time for some c, actually producing instances for which we can confidently predict the execution process proves very difficult. The best we are able to present in this paper is a type of instance of maximum degree 3 taking $\Omega^*(1.1048^n)$ time, to contrast with our upper bound $\mathcal{O}^*(1.1499^n)$ for the same situation.

For this purpose, build an instance I_k as a form of warped k-rung "ladder": Use variables x_i and y_i for $1 \leq i \leq k$, and clauses $(y_i \vee y_{i+1})$ for all $1 \leq i < k$, $(x_i \vee x_{i+2})$ for all $1 \leq i \leq k-2$, and $(x_i \vee y_i)$ for all $1 \leq i \leq k$. We show that $\Omega^*(1.1048^n)$ is consistent with the algorithm description for this type of instance.

Lemma 8. *The algorithm C, applied to an instance* I_k *with* $n = 2k$ *variables, can take* $\Omega^*(\tau(6,8)) > \Omega^*(1.1048^n)$ *time.*

Proof. It is consistent with the algorithm description that it branches on the leftmost y_i whose every neighbour is of degree 3 (in the original instance I_k, this would be y_3). When $y_i = 0$, all variables x_j and y_j for $j \geq i+2$ remain, making the next branching candidate y_{i+4}; when $y_i = 1$, at least all variables x_j and y_j with $j > i$ remain, making the next branching candidate y_{i+3}. Following this process until $i \geq k-3$ forms a recursion tree with, asymptotically, $\tau(3,4)^k$ lowest-level instances. □

7 Conclusions

We have shown how to integrate analysis by non-uniform, piecewise linear measures, as used in previous #2SAT$_w$ bounds [3, 7], with the multi-variate recurrence approach by Eppstein [5], thereby combining the ability of the former to model algorithms whose behaviour varies depending on parameters of the input, with the good bounds and the automatability of the bound calculation of the latter. On the one hand, we have used this to give a tighter upper bound of $\mathcal{O}^*(1.2377^n)$ on the time required for solving #2SAT$_w$. On the other, we would like to point out that the question of the tightness of the resulting bound is unresolved. In other words, under what conditions will the bound $\mathcal{O}^*(c^\mu)$ produced by the method correctly describe the worst-case behaviour of a model of degree-bounded branchings as described in Sect. 3, or of other related variants?

References

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

[2] V. Dahllöf, P. Jonsson, and M. Wahlström. Counting satisfying assignments in 2-SAT and 3-SAT. In *Proceedings of the 8th Annual International Conference on Computing and Combinatorics, (COCOON-2002)*, pages 535–543, 2002.

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

[4] O. Dubois. Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science*, 81:49–64, 1991.

[5] D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proceedings of the 15th annual ACM-SIAM symposium on Discrete algorithms (SODA-2004)*, pages 788–797, 2004.

[6] F. V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87:47–77, 2005.

[7] M. Fürer and S. P. Kasiviswanathan. Algorithms for counting 2-SAT solutions and colorings with applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 5(033), 2005.

[8] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2004)*, page 328, 2004.

[9] D. Kozen. *The design and analysis of algorithms*. Springer-Verlag, New York, 1992.

[10] M. Littman, T. Pitassi, and R. Impagliazzo. On the complexity of counting satisfying assignments. In the working notes of the LICS 2001 workshop on Satisfiability.

[11] H. J. Ryser. *Combinatorial Mathematics*. The Mathematical Association of America, Washington, 1963.

[12] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[13] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8:410–421, 1979.

[14] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[15] M. Wahlström. An algorithm for the SAT problem for formulae of linear length. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA-2005)*, pages 107–118, 2005.

[16] M. Wahlström. *Algorithms, measures, and upper bounds for satisfiability and related problems*. Linköping Studies in Science and Technology, PhD Dissertation no 1079, 2007. Available online through `http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-8714`.

[17] W. Zhang. Number of models and satisfiability of sets of clauses. *Theoretical Computer Science*, 155:277–288, 1996.