Algorithms for Counting 2-Sat Solutions and Colorings with Applications[*]

Martin Fürer[†] furer@cse.psu.edu
Shiva Prasad Kasiviswanathan kasivisw@cse.psu.edu
Computer Science and Engineering,
Pennsylvania State University

Abstract

An algorithm is presented for exactly solving (in fact, counting) the number of maximum weight satisfying assignments of a 2-Cnf formula. The worst case running time of $O(1.246^n)$ for formulas with n variables improves on the previous bound of $O(1.256^n)$ by Dahllöf, Jonsson, and Wahlström. The algorithm uses only polynomial space. As a consequence we get an $O(1.246^n)$ time algorithm for counting maximum weighted independent sets.

The above result when combined with a better partitioning technique for domains, leads to improved running times for counting the number of solutions of binary constraint satisfaction problems for all domain sizes. For large domain size d we approach $O((0.601d)^n)$ improving the previous best bound of $O((0.622d)^n)$. We further improve this bound for counting 3-colorings in a graph. The upper bound of $O(1.770^n)$ for graphs with n vertices improves on the previous bound of $O(1.788^n)$ by Angelsmark and Jonsson.

1. Introduction

There has a been a growing interest in the analysis of algorithms for NP-hard problems, such as satisfiability [11] or graph coloring [6]. Improvements in the exponential bounds are critical, for even a slight improvement from $O(c^n)$ to $O((c-\epsilon)^n)$ can significantly change the range of the problem being tractable. Also exhaustive algorithms are motivated by the fact that, some of these problems like Maximum Independent Set are hard to approximate even within a factor of $O(n^{1-\epsilon})$ in polynomial time [18, 21]. Approximation algorithms for NP-hard decision problems like satisfiability or graph k-colorability are nonsensical as pointed out by [17].

Most of the super-polynomial algorithms known are only for decision problems. As a natural extension we have counting problems, where we wish to not only decide the existence of solution, but to count the number of solutions. Counting problems are not only mathematically interesting, but they also arise in many applications [28, 29].

The decision problem of weighted 2-Sat is NP-hard and the corresponding counting problem (#2-Sat) is #P-complete even for the unweighted case [24, 31]. The class #P (proposed by Valiant [30]) is defined as $\{f : \exists$ a non deterministic polynomial time Turing Machine M such that on input x, M has exactly $f(x)$ accepting leaves$\}$. We consider

∗ A preliminary version appeared as ECCC report TR05-033 [19]

† This material is based upon work supported by the National Science Foundation under Grant CCR-0209099

the problem of counting the number of maximum weight satisfying assignments of 2-Cnf formula. Earlier works in this area include papers by Dubois [14], Zhang [34], Littman *et al.* [27]. The algorithm by Zhang [34] runs in $O(1.618^n)$ time for #2-Sat, whereas the algorithm by Littman *et al.* runs in $O(1.381^n)$ time. In a recent paper Dahllöf *et al.* [10] proved the upper bound to $O(1.2561^n)$ for solving (also counting) weighted 2-Sat solutions.

We extend the series of work done by Dahllöf *et al.* [8, 9, 10] on this problem. Our algorithm for counting models and max-weight models uses polynomial space and runs in time $\mathrm{O}(1.2461^n)$. The weighted 2-Sat problem is closely related to the well-studied problem of finding (or counting) maximum weighted independent sets. Using a standard reduction we also obtain an $O(1.2461^n)$ time algorithm for the problem of counting maximum weighted independent sets.

Since the preliminary version of the paper appeared as the technical report [19] our results have found applications in a variety of problems. Anglesmark [1] noted that the faster algorithm for the decision 2-Sat leads to a faster algorithm for the problem of MAX Hamming Distance $(2, 2)$-Csp. Very recently, Björklund and Husfeldt [7] used our results to obtain a faster algorithm for determining the chromatic number of graph. They also present a $2^n n^{O(1)}$ time and space algorithm for counting k-colorings of a graph. Analogous results were also obtained by Koivisto [23].

Our *main improvement* in the running time for 2-Sat, is by improved handling of the subproblem with a restriction to at most three occurrences of every variable, i.e., the corresponding constraint graph (formally defined later) is of degree at most three. Here, the decisive parameter determining the running time is the number of degree 3 nodes. However, more progress in eliminating degree 3 nodes is possible when there are many of them, i.e., when the average degree is higher. For example, when the average degree is more than 12/5, we can find a degree 3 vertex with a neighbor of degree 3, and both are eliminated in at least one of the assignments of the first degree-3 vertex. We take advantage of this by choosing a different complexity measure above 12/5. Our improved time bounds for degree 3 propagate to formulas of higher degrees, because the average degree has a tendency to shrink during the iterative algorithm. This extension to higher degrees is done in an automated manner with the framework of Dahllöf *et al.* [10].

The decision problem for (d, l)-Csp (Constraint Satisfaction Problem) is NP-complete and the corresponding counting problem is #P-complete even when restricted to binary Csp [28]. From [28] we also know that for every fixed $\epsilon > 0$, approximating the number of solutions to a binary Csp within $2^{n^{1-\epsilon}}$ is NP-hard. Solving a Csp instance is equivalent to finding a homomorphism between graphs [22], and finding the number of graph homomorphisms has important applications in statistical physics [15, 16, 32], e.g., computation in the Potts and hardcore lattice gas model and the problem of counting q-particle Widom-Rowlinson configurations in graphs, where $q > 2$ (See [15] for detailed descriptions of these models). In artificial intelligence, problems like approximate reasoning [28] can be viewed as #Csp-instances. Also a large number of problems in computer science can be seen as constraint satisfaction problems. Some examples are floor planning, graph problems, scheduling a collection of tasks, and interpreting a visual image [26].

In this paper we also extend the series of algorithms presented by Angelsmark *et al.* [2, 3] for counting the number of solutions of binary Csp's. The algorithm presented in [2] partitions the domains of the variables into sub domains of sizes between 2 and 5 elements

Problem	Old Results		This Paper	
	Decision	*Counting*	*Decision*	*Counting*
Weighted 2-Sat,	$O(1.2561^n)$	$O(1.2561^n)$	$O(1.2461^n)$	$O(1.2461^n)$
$(d,2)$-Csp (large d)	$\tilde{O}((d!)^{n/d})$	$O((0.6224d)^n)$		$O((0.6009d)^n)$
3-Col	$O(1.3289^n)$	$O(1.7879^n)$		$O(1.7702^n)$

Figure 1. Improvements in Running Times. The decision version of $(d,2)$-Csp is from [17]. The decision version of the 3-coloring is from [6]. Only polynomial space results are considered.

and uses #2-Sat for solving these smaller instances. We use the same algorithm as in Angelsmark and Jonsson [2], however the speedup is due to improved #2-Sat algorithm and by using an improved partitioning for the domains. Another related work by Williams [33] deals with exponential space algorithms for the case for $\#(d,2)$-Csp where any subset of constraints need to be satisfied. We count solutions satisfying all the constraints, and our base in the exponential running time is significantly smaller for this case.

For large domain size d we approach $O((0.6009d)^n)$ time improving the previous best bound of $O((0.6224d)^n)$ achieved in [2]. Also we provide an improved algorithm for counting the number of 3-colorings (problem #3-Col) in a graph, which runs in $O(1.7702^n)$ time, an improvement over the $O(1.7879^n)$ time algorithm from [2]. Finding a k-coloring of a graph G is equivalent to finding a homomorphism from G to a complete graph with k vertices. The algorithm builds on the ideas presented in [2] to obtain the improvement. The improvement in the running time for #3-Col is particularly important because it automatically improves the running time of the best known algorithm for counting k-colorings. Throughout this paper, $\tilde{O}(f(n))$ will denote $n^{O(1)}f(n)$.

Figure 1 summarizes our key results (excluding applications). The remaining paper is organized as follows. In Section 2 we define some preliminaries and technical tools. In Section 3 we present the algorithm for #2-Sat and analyze its performance. In Section 3.3 we define some interesting problems whose running times have improved as a consequence. In Section 4 we present an improved analysis for counting solutions to binary constraint satisfaction problems and in Section 5, we present an improved algorithm for counting 3-colorings.

2. Preliminaries and Problem Defintions

We will employ notation similar to that used in [10, 20]. A propositional variable, or variable takes values *true* or *false*. A literal is a variable (x) or its negation $(\neg x)$. A clause is a finite non empty collection of literals. A propositional formula in conjunctive normal form is a conjunction of disjunction of literals. A k-Cnf formula is in conjunctive normal form if each clause contains at most k literals.

#k-Sat is the problem of computing the number of maximum weight models (i.e., satisfying assignments) for a k-Cnf formula. With each literal l, a weight $w(l) \in \mathbb{N}$ and a count $c(l) \geq 1$ are associated; the vectors W and C are the corresponding vectors. Let L be the set of literals, we define the weight and cardinality of of a model M respectively as

$$\mathcal{W}(M) = \textstyle\sum_{\{l \in L : l \text{ is true in } M\}} w(l),$$
$$\mathcal{C}(M) = \textstyle\prod_{\{l \in L :| l \text{ is true in } M\}} c(l).$$

Given a formula F, let $F[x=0]$ be the result of assigning $x = 0$ and not propagating. $F[x=1]$ is defined accordingly. $Var(F)$ denotes the set of variables of function F and $n(F) = |Var(F)|$. A variable which occurs only as x or only as $\neg x$ is called *monotone*.

We define the *constraint graph* $G(F) = (Var(F), E)$, as an undirected graph where the vertex set is the set of variables and the edge set E is defined as

$$\{(x, y) : x, y \text{ appear in the same clause of } F\}.$$

The *degree* $d(x)$ of a variable x is the degree of x in G. We use $d(F)$ to denote the maximum degree of any variable in F and $n_d(F)$ is the number of variables of degree d in F. *Singleton* variables are variables of degree 1. For convenience, we use a slightly altered notion of neighborhood. The *neighborhood* of a vertex x in the graph G, denoted by $N_G(x)$, is the set $\{y : (x, y) \in E\} \cup \{x\}$. The size of the neighborhood of x is $S(x) = \sum_{y \in N_G(x)} d(y)$.

We define $m(F)$ as $\sum_{x \in Var(F)} d(x)$. Both $n(F)$ and $m(F)$ are used as measures of formula complexity. For $\mathcal{M}$ being the set of all maximum weight models for F and M' being any arbitrary maximum weight model in $\mathcal{M}$ define

$$\#2\text{-}\textsc{Sat}(F, C, W) = \left(\sum_{M \in \mathcal{M}} \mathcal{C}(M), \mathcal{W}(M') \right).$$

A $(d, 2)$-Csp instance is a triple (V, D, C), where V is a finite set of variables, D a finite domain of values with $|D| = d$, and C a finite set of constraints $\{c_1, \dots, c_q\}$. Each constraint c_i is a triple xRy, where $x, y \in V$ and $R \subseteq D^2$. A solution is a function $f : V \to D$, s.t. $(f(x), f(y)) \in R$ for each constraint xRy. We say that a variable occurring in the Csp is k-valued if it takes only values from $\{1, \dots, k\}$.

Let G be a graph with V as set of vertices and E as set of edges. A k-coloring of a graph G is a function $C : V \to \{1, \dots, k\}$ such that for all $v, w \in V$, if $C(v) = C(w)$ then $(v, w) \notin E$. The $\#k$-Col problem is to determine the number of k-colorings for G.

2.1 Estimation of Tree Size

Loosely speaking, the idea behind our algorithms is recursive decomposition based on a popular approach that has originated in papers by Davis, Putnam, Logemann and Loveland [13, 12]. The recurrent idea behind these algorithms is to choose a variable x and to recursively count the number of satisfying assignments where x is true as well as those where x is false, i.e., we *branch on* x. The algorithms operate on all connected components of the constraint graph.

For #2-Sat we follow the analysis of Kullmann [25]. In the implicit branching tree constructed, let x be a node with 2 branches labeled with positive real numbers t_1, t_2. The labels are the measures of the reduction in complexity in the respective branch. The *branching number* is the largest real-valued solution of $\sum_{i=1}^{2} z^{-t_i} = 1$. For a branching tuple

```
Function PROPAGATE($F, C, W$)
(Initialize $w \leftarrow 0$, $c \leftarrow 1$)
1) If there exists a clause $(1 \vee \ldots)$, then it is removed. Any variable $a$ in the clause which
   gets removed is handled according to following cases
      a) If $w(a) = w(\neg a)$, then $c \leftarrow c \cdot (c(a) + c(\neg a))$; $w \leftarrow w + w(a)$.
      b) If $w(a) < w(\neg a)$, then $c \leftarrow c \cdot (c(\neg a))$; $w \leftarrow w + w(\neg a)$.
      c) If $w(a) > w(\neg a)$, then $c \leftarrow c \cdot c(a)$; $w \leftarrow w + w(a)$.
2) If there is a clause of the form $(0 \vee \ldots)$, then remove 0 from it.
3) If there is a clause of the form $(a)$, then remove the clause and $c \leftarrow c \cdot c(a)$; $w \leftarrow w + w(a)$,
   and, if $a$ still appears in $F$ then set $F \leftarrow F[a = 1]$.
4) Return($F, c, w$).
```

Figure 2. Function PROPAGATE.

```
Function REDUCE($F, v, f$)
(Assume $F = F_0 \wedge F_1$ with $Var(F_0) \cap Var(F_1) = \{x\}$)

1) Let $f(F_i) \leq f(F_{1-i})$, $i \in \{0, 1\}$.
2) Let $(c_t, w_t) \leftarrow$ #2-SAT($F_i[x = 1], C, W$).
3) Let $(c_f, w_f) bec$ #2-SAT($F_i[x = 0], C, W$).
4) Modify the vectors $C$ and $W$ so that $c(x) \leftarrow c_t \cdot c(x)$, $c(\neg x) \leftarrow c_f \cdot c(\neg x)$, $w(x) \leftarrow w_t + w(x)$,
   $w(\neg x) \leftarrow w_f + w(\neg x)$.
5) Return #2-SAT($F_{1-i}, C, W$).
```

Figure 3. Function REDUCE.

(t_1, t_2) the branching number is denoted by $\tau(t_1, t_2)$. The branch from F to F_i is labeled by $t_i = \triangle f(F) = f(F) - f(F_i)$, where $f(F)$ is some algorithm specific measure of complexity. Defining $f_{max}(n) = max_{n(F)=n} f(F)$, ensures a running time of $\tilde{O}(\gamma^{f_{max}(n)})$, where γ is the largest branching number occurring in any tuple of the branching tree. We will define the function f such that the worst case branching number is $\tau(1, 1) = 2$. Let l, l' and l'' denote literals over the variable set $Var(F)$. In a step satisfying literal l, we eliminate all the clauses of the form $(l \vee l')$ and replace all clauses of the form $(\neg l \vee l'')$ by l''. We call a branching as *maximally unbalanced* if clauses of only one form occur.

2.2 Helper Functions

This subsection deals with important functions used for reducing the input fomula. We use similar functions and structures as in [10], some of which have been reproduced for completeness. The first function called PROPAGATE (Figure 2) simplifies the formula by removing dead variables. The input to the algorithm is the formula, count vector and weight vector. The four steps of the algorithm are performed until not applicable. The function returns the updated formula, the weight of the variables removed, and count for the eliminated variables.

```
Algorithm C-2-Sat(F, C, W)

1) If $F$ contains no clauses, then return (1,0).
2) If $F$ contains an empty clause, then return (0,0).
3) If the graph $G(F)$ is disconnected, then return $(c, w)$ where $c \leftarrow \prod_{i=0}^{j} c_i$, $w \leftarrow \prod_{i=0}^{j} w_i$,
   and $(c_i, w_i) \leftarrow$ C-2-Sat$(F_i, C, W)$ for connected components $G(F_0), \ldots, G(F_j)$.
4) If there exists a non-monotone variable $x$ with $d(x) \geq 6$, then branch on $x$.
5) If $d(F) \leq 6$, then return C-2-Sat$_6(F, C, W)$.
6) Pick a variable $x$ of maximum degree and branch on it.
```

Figure 4. Algorithm C-2-Sat.

```
Algorithm C-2-Sat_6(F, C, W)
(Assume $d(F) \leq 6$)

1) If $F$ contains no clauses, then return (1,0).
2) If $F$ contains an empty clause, then return (0,0).
3) If the graph $G(F)$ is disconnected, then return $(c, w)$ where $c \leftarrow \prod_{i=0}^{j} c_i$, $w \leftarrow \prod_{i=0}^{j} w_i$,
   and $(c_i, w_i) \leftarrow$ C-2-Sat$_6(F_i, C, W)$ for connected components $G(F_0), \ldots, G(F_j)$.
4) If Reduce is applicable, then apply it.
5) Pick a variable $x$ of maximum degree, with the maximum $S(x)$. There are two sub-cases:
    a) If $N_{G(F)}(x)$ is connected to the rest of the graph^1. using only two external^2. vertices
       $y$ and $z$, such that $d(y) \geq d(z)$, then branch on $y$.
    b) Else branch on $x$.
```

Figure 5. Algorithm C-2-Sat$_6$.

Another function called Reduce (Figure 3) reduces the input formula. It takes advantage of the fact that if a formula F can be partitioned into sub-formulas F_0 and F_1 such that each clause of F belongs to either of them, and $|Var(F_0) \cap Var(F_1)| = 1$, then we can remove F_0 or F_1 while appropriately updating count and weight associated with the common variable. Let $Var(F_0) \cap Var(F_1) = \{x\}$. The input to the function is the formula, x, and some algorithm specific measure of complexity f. We say that in such a situation that Reduce is *applicable*. Among F_0, F_1 we always remove the one having a smaller value under f. Note that Reduce needn't be a polynomial time operation. We apply the function Reduce as long as it is applicable. A formula F is called a *maximally reduced* formula if this routine doesn't apply.

The following lemma (stated without proof) from [10] shows that the value of #2-Sat(F, C, W) is preserved under both Propagate and Reduce routines.

Lemma 1 *(Dahllöf et al. [10]) Applying the functions* Reduce *and* Propagate *does not change the return value of* #2-Sat(F, C, W).

```
1) Let $(F_t, c_t, w_t) \leftarrow \textsc{Propagate}(F[x=1], C, W)$.
2) Let $(F_f, c_f, w_f) \leftarrow \textsc{Propagate}(F[x=0], C, W)$.
3) Let $(c'_t, w'_t) \leftarrow$ C-2-SAT$(F_t, C, W)$ and $(c'_f, w'_f) =$ C-2-SAT$(F_f, C, W)$.
4) Let $W_{true} \leftarrow w(x) + w_t + w'_t$, $W_{false} \leftarrow w(\neg x) + w_f + w'_f$, $C_{true} \leftarrow c(x) \cdot c_t \cdot c'_t$, and
   $C_{false} \leftarrow c(\neg x) \cdot c_f \cdot c'_f$. There are 3 cases:
    a) If $W_{true} = W_{false}$, then return$(C_{true} + C_{false}, W_{true})$.
    b) Else if $W_{true} > W_{false}$, then return$(C_{true}, W_{true})$.
    c) Else if $W_{true} < W_{false}$, then return$(C_{false}, W_{false})$.
```

Figure 6. Shorthand for the phrase branch on x.

3. Algorithm for #2-Sat

We have a main Algorithm C-2-Sat (Figure 4) which makes use of another function C-2-Sat$_6$ when $d(F) \leq 6$. The algorithms operate on all connected components of the constraint graph. The phrase *branch on* used in the algorithms is a shorthand for technicalities described in Figure 6. The proof of correctness of the Algorithm C-2-Sat is straightforward and can be shown as in [10].

We now concentrate on the analysis of the running time for C-2-Sat$_6$. For the analysis we use a continuous and piecewise linear function $f(n, m)$ similar to the one introduced by [10] as a measure of complexity where $n = n(F)$ and $m = m(F)$. A branching variable is chosen to optimize the progress in the next step. There is a worst case branching associated with each value of m/n. Using a classical model of complexity, such as $n(F)$, means that the worst case branching numbers are smaller near the top of the branching tree and increase as we go down. Informally, this is because the maximum degree $d(F)$ is smaller at the bottom of the branching tree, hence smaller pieces of the formula are removed. The estimation of the running time as $\tilde{O}(\gamma^{f_{max}(n)})$ (with $f_{max}(n) = \max\{f(n,m) : m \in \mathbb{N}\}$) is best when the branching numbers are uniform throughout.

The complexity measure introduced by Dahllöf *et al.* [10] incorporates the effects of the decreasing m/n quotient in the upper time bound, leading to a better worst case running time estimates. We will find a sequence of worst cases as the m/n quotient increases. Each worst case i is associated with a piecewise linear function $f_i(n, m) = a_i n + b_i m$, a lower limit k_i for m/n below which it is possible that worse cases appear, and an upper limit k_{i+1} for the m/n above which that worst case can't occur. We define the coefficient χ_i by $\chi_i = a_i + k_i b_i$ implying that $f(n, m) = f_i(n, m) = f_{i-1}(n, m) = \chi_i n$ for $m/n = k_i$. Now $f_i(n, m)$ can also be expressed as $f_i(n, m) = \chi_i n + (m - k_i n) b_i$. We define a *Interval* i as the range k_i to k_{i+1}. The formal definitions of the functions are (similar to [10]):

$$\begin{aligned}
f(n,m) &= f_i(n,m) \text{ where } k_i < m/n \leq k_{i+1},\ 0 \leq i \leq 18,\\
f_i(n,m) &= a_i n + b_i m = \chi_i n + (m - k_i n) b_i,\ 0 \leq i \leq 18,\\
\chi_i &= \chi_{i-1} + (k_i - k_{i-1}) b_{i-1},\ 1 \leq i \leq 19,\\
\chi_0 &= 0.
\end{aligned}$$

1. Graph induced by the vertex set $Var(F) \setminus N_{G(F)}(x)$.
2. $\{y, z\} \cap N_{G(F)}(x) = \emptyset$.

The values[3.] of k_i, χ_i, a_i, b_i are in Figure 7. Also provided are the worst case recurrences and the corresponding running times. $\widetilde{O}(2^{\chi_i n})$ is the upper limit on the running time for a formula F with $m(F)/n(F) \leq k_i$. Following are some interesting properties of $f(n,m)$ used in the analysis. The first property can be observed from the Figure 7, and the second is derived in [10].

Properties: 1) $f(n,m) > f(n-1,m)$ if $m > 3.75n$.

2) $f(n,m) \geq f(n_1,m_1) + f(n-n_1, m-m_1)$ if $0 \leq n_1 \leq n, 0 \leq m_1 \leq m$.

Interval i	k_i, k_{i+1}	*Worst case*	χ_i	b_i	a_i	*Time*
0	0, 2		0	0	0	poly(n)
1	2, 2.4	$\tau(4a_1+12b_1, 4a_1+12b_1)$	0	1/4	-1/2	$2^{n/9.99}$
2	2.4, 2+2/3	$\tau(2a_2+8b_2, 4a_2+14b_2)$	0.1	0.188	-0.352	$2^{n/6.65}$
3	2+2/3, 3	$\tau(a_3+6b_3, 4a_3+16b_3)$	0.150	0.155	-0.265	$2^{n/4.94}$
4	3, 3.2	$\tau(2a_5+10b_5, 5a_5+18b_5)$	0.202	0.090	-0.068	$2^{n/4.54}$
5	3.2, 3.5	$\tau(a_5+8b_5, 5a_5+20b_5)$	0.220	0.089	-0.067	$2^{n/4.04}$
6	3.5, 3.75	$\tau(a_6+8b_6, 5a_6+22b_6)$	0.247	0.076	-0.018	$2^{n/3.75}$
7	3.75, 4	$\tau(a_7+8b_7, 5a_7+24b_7)$	0.266	0.065	0.021	$2^{n/3.54}$
8	4, 4+4/29	$\tau(a_8+10b_8, 6a_8+26b_8)$	0.282	0.036	0.136	$2^{n/3.47}$
9	4+4/29, 4+4/9	$\tau(a_9+10b_9, 6a_9+28b_9)$	0.287	0.032	0.153	$2^{n/3.36}$
10	4+4/9, 4+4/7	$\tau(a_{10}+10b_{10}, 6a_{10}+30b_{10})$	0.297	0.028	0.169	$2^{n/3.33}$
11	4+4/7, 4.8	$\tau(a_{11}+10b_{11}, 6a_{11}+32b_{11})$	0.301	0.026	0.182	$2^{n/3.25}$
12	4.8, 5	$\tau(a_{12}+10b_{12}, 6a_{12}+34b_{12})$	0.307	0.023	0.195	$2^{n/3.21}$
13	5, 5+5/47	$\tau(a_{13}+12b_{13}, 7a_{13}+36b_{13})$	0.311	0.006	0.278	$2^{n/3.20}$
14	5+5/47, 5+1/3	$\tau(a_{14}+12b_{14}, 7a_{14}+38b_{14})$	0.312	0.006	0.281	$2^{n/3.18}$
15	5+1/3, 5.5	$\tau(a_{15}+12b_{15}, 7a_{15}+40b_{15})$	0.313	0.005	0.283	$2^{n/3.17}$
16	5.5, 5+5/8	$\tau(a_{16}+12b_{16}, 7a_{16}+42b_{16})$	0.314	0.005	0.286	$2^{n/3.16}$
17	5+5/8, 5+5/6	$\tau(a_{17}+12b_{17}, 7a_{17}+44b_{17})$	0.315	0.004	0.289	$2^{n/3.15}$
18	5+5/6, 6	$\tau(a_{18}+12b_{18}, 7a_{18}+46b_{18})$	0.316	0.004	0.291	$2^{n/3.15}$

Figure 7. Parameter table for $a_i n + b_i m$ $(k_i < m/n \leq k_{i+1})$.

3.1 Worst Case Branching

In this subsection we discuss the worst case branching situation for C-2-SAT$_6$. The following lemma shows that if we change Interval during branching it is even better for our measure f. Therefore, the worst case occurs only when both m/n and m_1/n_1 are in the same *interval.*

Lemma 2 *Let $f(n,m)$ and a_i, b_i, m_1, n_1 be defined as above. Then $\triangle f(n,m) = f(n,m) - f(n_1,m_1) \geq \triangle f_i(n,m) = f_i(n,m) - f_i(n_1,m_1)$ if $k_i < m/n \leq k_{i+1}$.*

Proof. The equality holds if $k_i < m_1/n_1 \leq k_{i+1}$. The proof for $m_1/n_1 \leq k_i$ is omitted and can be found in [10]. Now assume that $m_1/n_1 > k_{i+1}$. By inductive arguments it is enough

3. Generated using a simple computer program.

to focus on the transition of a single barrier, i.e., $m/n \geq k_i$ belongs to Interval i, whereas $m_1/n_1 \geq k_{i+1}$ belongs to Interval $i+1$. We want to check that $f_i(n_1, m_1) \geq f_{i+1}(n_1, m_1)$.

$$\begin{aligned} f_i(n_1,m_1) - f_{i+1}(n_1,m_1) &= (a_i - a_{i+1})n_1 + (b_i - b_{i+1})m_1 \\ &= (\chi_i - \chi_{i+1} - k_i b_i + k_{i+1} b_{i+1})n_1 + (b_i - b_{i+1})m_1 \\ &= k_{i+1}(b_{i+1} - b_i)n_1 + (b_i - b_{i+1})m_1 \geq 0 \end{aligned}$$

The last step follows because $m_1/n_1 \geq k_{i+1}$ implying that $|(b_i - b_{i+1})m_1| \geq |k_{i+1}(b_{i+1} - b_i)n_1|$. Also, b_i is non-increasing with i. ❑

A worst case branching occurs if the branching is maximally unbalanced. If $d(F) = 2$ in Step 5a, then F is a cycle and we are done after one branching. Thus, assume that we are in Step 5a with $d(F) \geq 3$, and after branching on y we obtain the maximally reduced formulas F_1 and F_2. In both branches, we eliminate at least y by assignment and $N_{G(F)}(x)$ by Reduce. It can be seen that in the worst case, in one branch we have $\triangle n_1 = d(x) + 2$, $\triangle m_1 \geq S(x) + 6$. Also in the worst case the other branch will have $\triangle n_2 = d(x) + 4$, $\triangle m_2 \geq S(x) + 10$.

The Step 5b of the Algorithm C-2-Sat$_6$ is the more interesting case and requires special attention. Let $x \in F$ be the variable we branch on to get maximally reduced formulas F_1 and F_2. In the worst case, in one branch we have $\triangle n_1 = 1+$ #degree 2 nodes in $N_{G(F)}(x) \setminus \{x\}$, $\triangle m_1 = 2 \cdot (d(x) +$ #degree 2 nodes in $N_{G(F)}(x) \setminus \{x\})$. Also in the worst case the other branch will have $\triangle n_2 = d(x) + 1$, $\triangle m_2 \geq 2\lceil \frac{S(x)+3}{2} \rceil$ (Lemma 3). We use these bounds as our worst cases throughout the paper. If $m > 3.75n$ (when both a_i and b_i are positive) it is obvious from the Property 1 of f that Step 5b is harder than Step 5a. For any Interval i with $m \leq 3.75n$ one can easily verify with the given a_i and b_i that Step 5a always has a worst case branching number less than 2. Long chains of degree 2 nodes don't hurt when $m \leq 2.4n$ because $2b_i + a_i = 0$, $i \in \{0, 1\}$ and long chains are beneficial if $m > 2.4n$ because $2b_i + a_i > 0$, $i \in \{2, 3, \ldots, 18\}$. So, from now on we will only be focusing on Step 5b.

Lemma 3 *Let $x \in F$ be the variable we branch on in Step 5b of the Algorithm C-2-Sat$_6$. In a worst case branching, i.e., the branching is maximally unbalanced, we decrease $m(F)$ by at least $2\lceil \frac{S(x)+3}{2} \rceil$ in at least one of the branchings.*

Proof. The proof follows by observing that in order to achieve a maximally unbalanced branching we delete all the edges incident on $N_{G(F)}(x)$ in one branching step. Since $N_{G(F)}(x)$ is connected to the outside by at least 3 vertices. We get that the change in $m(F)$ is at least $S(x) + 3$ when $S(x)$ is odd, and $S(x) + 4$ when $S(x)$ is even. The later is true because any edge between two elements on $N_{G(F)}(x)$ contributes two to $S(x)$. Therefore the decrease in $m(F)$ is $2\lceil \frac{S(x)+3}{2} \rceil$. ❑

We also use the following lemma (stated without proof) from [10] that makes a connection between the values of $m(F)/n(F)$ and worst case branchings.

Lemma 4 *(Dahllöf et al. [10]) Let F be a non-empty formula with $m(F)/n(F) = k \in \mathbb{Q}$, and define $\alpha(x)$ and $\beta(x)$ such that*

$$\alpha(x) = d(x) + |\{y \in N_{G(F)}(x) \,:\, d(y) < k\},$$
$$\beta(x) = 1 + \textstyle\sum_{\{y \in N_{G(F)}(x) \,:\, d(y) < k\}} 1/d(y).$$

There exists some variable $x \in Var(F)$ such that $d(x) \geq k$ and $\alpha(x)/\beta(x) \geq k$.

3.2 Performance of C-2-Sat$_6$

The proof will be divided according to the values of $m(F)/n(F)$. We branch on some variable $x \in F$, eventually resulting in two maximally reduced formulas F_1 and F_2. It is possible to end up with more than two maximally reduced formulas, then the above applies to all connected components $G(F_i)$ and by Property 2 the total work is smaller. It is shown that in each Interval i the worst case branching number is 2. The case where $m \leq 2n$ is a straight forward consequence of applying Reduce.

Lemma 5 *(**Dahllöf et al. [10]**)Let F be a maximally reduced formula with $m \leq 2n$, then C-2-*Sat$_6$*(F, C, W) runs in $O(poly(n))$ time.*

We now handle denser cases. First the case where $m \leq 3n$. As mentioned earlier, this case is one of the our main sources of improvement from Dahllöf *et al.* [10]. The benefits of this propagate through to the later cases.

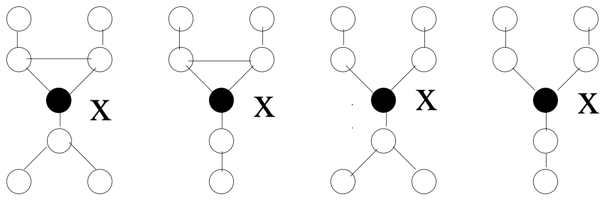

Figure 8. All possible neighborhood configurations with $d(F) = d(x) = 3$.

Lemma 6 *Let F be a maximally reduced formula with $m \leq 3n$ and $d(F) = 3$, then C-2-*Sat$_6$*(F, C, W) runs in $\tilde{O}(2^{\chi_4 n})$ time.*

Proof. We divide this case into worst cases depending on the number of degree 3 nodes (0,1,2 or 3) adjacent to x (Figure 8). k_i to k_{i+1} captures the range of m/n where each worst case can appear. For example, if no degree 3 nodes are adjacent to one another, then the worst case is a bipartite graph with 2n/5 degree 3 nodes on one side and 3n/5 degree 2 nodes on the other side. This leads to a value of $(3 \cdot 2/5 + 2 \cdot 3/5) = 12/5$ for k_2. This lemma uses $f_1(n, m)$ to $f_3(n, m)$ as measures.
Interval 1: $m/n \in (2, 12/5]$, $d(F) = 3$.
In this case we focus on the number of degree 3 variables $n_3(F)$. We will decrease this number by 4 in one step (in the worst case). Therefore, we actually use $n_3(F)/4$ as a measure, i.e., $b_1 = 1/4$ and $a_1 = -1/2$. If F is maximally reduced then $n_3(F) = m(F) - 2n(F)$. We can show that $\triangle n_3(F) \geq 4$ along any branch by proving that $\triangle m \geq 2\triangle n + 4$. Let V and $V_1 \subseteq V$ be the set of variables in F and F_1 respectively. The reduction in m is

$$\triangle m = \sum_{v \in V - V_1} d(v) + 2|\{\text{clauses } C' \text{ in } F : C' \text{ involves variables from both } V - V_1 \text{ and } V_1\}|.$$

Since there are no singleton variables in F, $d(v) = 3$, the first term is at least $2\triangle n + 1$, and since Reduce does not apply, the second term is at least 2. Taking them together and also

noting the fact that $m(F)$ is even, we have $\triangle m \geq 2\triangle n + 4$, so $\triangle n_3(F) \geq 4$. The same argument also works for F_2. Furthermore, $\chi_2 = \chi_1 + (12/5 - 2)b_1 = 0.1$.
Interval 2: $m/n \in (12/5, 2 + 2/3]$, $d(F) = 3$.
In this case x has at least one degree 3 node as its neighbor. There are two worst case recursions to be considered in this case. Here we use the estimates for $\triangle n_1$, $\triangle m_1$, $\triangle n_2$, $\triangle m_2$ from Section 3.1.
Case 1: $S(x) = 10$.
In this case x has exactly one degree 3 node as its neighbor. The worst case branching is when the branching is maximally unbalanced, with a branching number of $\tau(3a_2+10b_2, 4a_2+14b_2) < 2$.
Case 2: $S(x) = 11$.
In this case x has exactly two degree 3 nodes as its neighbors. The worst case branching is when the branching is maximally unbalanced, with a branching number of $\tau(2a_2+8b_2, 4a_2+14b_2) = 2$. As given in Figure 7 we have $b_2 \approx 0.1884$ and $a_2 \approx -0.3520$ and $\chi_3 = \chi_2 + (8/3 - 12/5)b_2 \approx 0.1502$.
Interval 3: $m/n \in (2 + 2/3, 3]$, $d(F) = 3$.
In this case x has three degree 3 nodes as its neighbors. The worst case branching number is $\tau(a_3 + 6b_3, 4a_3 + 16b_3) = 2$. As given in Figure 7 we have $b_3 \approx 0.1557$ and $a_3 \approx -0.2650$ and $\chi_4 = \chi_3 + (3 - 8/3)b_3 \approx 0.2021$.
As we see the worst case branching number is 2 and the worst case running time for C-2-SAT$_6$ with maximum degree 3 and $m \leq 3n$ is $\tilde{O}(2^{\chi_4 n})$. ❑

Lemma 7 *Let F be a maximally reduced formula with $m \leq 3n$ and $4 \leq d(F) \leq 6$, then C-2-*SAT$_6(F, C, W)$ *runs in* $\tilde{O}(2^{\chi_4 n})$ *time.*

Proof. There are 5 types of worst cases (Figure 9) depending on the number of degree 2 nodes (0 to 4) adjacent to the selected vertex x, i.e., $S(x)$ varies from 12 to 16. Bigger values of $S(x)$ are better as $\triangle n$ stays the same, but $\triangle m$ increases. The following numbers are generated using bounds as given above.

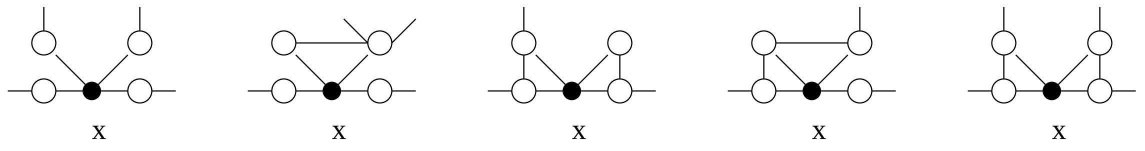

Figure 9. Sample worst cases with $d(F) = d(x) = 4$ with $m \leq 3n$.

$\triangle n_1$	$\triangle m_1$	$\triangle n_2$	*Lower bound for* $\triangle m_2$
5	6	5	16
4	14	5	16
3	12	5	18
2	10	5	18
1	8	5	20

In all these cases, the branching number is less than 2 for any one of the f_1 to f_3. For $d(F) = 5$ there are 6 types of worst case branchings again depending on the number of degree 2 nodes (0 to 5) adjacent to the selected vertex x.

$\triangle n_1$	$\triangle m_1$	$\triangle n_2$	*Lower bound for* $\triangle m_2$
6	20	6	18
5	18	6	20
4	16	6	20
3	14	6	22
2	12	6	22
1	10	6	24

Again, for all these cases[4.] the branching number is less than 2 for any one of the f_1 to f_3. Using the same idea to generate the worst cases when $d(F) = 6$ we get:

$\triangle n_1$	$\triangle m_1$	$\triangle n_2$	*Lower bound for* $\triangle m_2$
7	24	7	22
6	22	7	22
5	20	7	24
4	18	7	24
3	16	7	26
2	14	7	26
1	12	7	28

One can repeat the same exercise as above to verify that the claim holds true even if $d(F) = 6$. The worst case running time of C-2-SAT$_6$ with $m \leq 3n$ is therefore $\tilde{O}(2^{\chi_4 n}) \approx O(1.1503^n)$. ❑

Lemma 8 *For a maximally reduced formula F with $3n < m \leq 4n$, C-2-*SAT$_6(F, C, W)$ *runs in $\tilde{O}(2^{\chi_8 n})$ time.*

Proof. In this case we have $d(F) = 4$, 5 or 6. We use the measures of $f_4(n, m)$ to $f_7(n, m)$. If $d(F) = 5$ or 6 we have no guarantees on $S(x)$. First we consider the case $d(F) = 5$. The worst cases are the same as in the previous lemma. It can be again checked that in all these cases the branching number is less than 2 for $f_4(n, m)$ to $f_7(n, m)$. The same argument works for $d(F) = 6$ too. If we have the case $d(F) = 4$, Lemma 3 guarantees that there will be a variable y with $\alpha(y)/\beta(y) > 3$. The minimum value of $S(y)$ satisfying this property is 15. Even though we know that for the chosen variable x, $S(x) \geq 15$, we don't have the guarantee that $\alpha(x)/\beta(x) > 3$.
Interval 4: $m/n \in (3, 3.2]$. $d(F) = 4$.
In this case we have neighbors with degrees 2, 3, 3, and 3. The worst case branching number is $\tau(2a_2 + 10b_2, 5a_2 + 18b_2)$, solving which we get $b_4 \approx 0.090$ and $a_4 \approx -0.068$ and $\chi_5 = \chi_4 + (3.2 - 3)b_4 \approx 0.220$. Since cases with $S(x) = 15$ can appear only till $m(F)/n(F) \leq 3.2$ (from Lemma 3), the next *interval* starts with $m(F)/n(F) > 3.2$.
Intervals 5-7 can be shown similarly, and also have been handled by [10]. The worst case running time of C-2-SAT$_6$ with $m \leq 4n$ is $\tilde{O}(2^{\chi_8 n}) \approx O(1.216^n)$. ❑

4. One can eliminate a few cases by noting that $|2b_i| \geq |a_i|$ in this range

Lemma 9 *For a maximally reduced formula F with $4n < m \leq 5n$, C-2-*Sat$_6(F, C, W)$ *runs in* $\tilde{O}(2^{\chi_{13}n})$ *time.*

Proof. In this case we have $d(F) = 5$ or 6. We use the measures of $f_8(m, n)$ to $f_{12}(m, n)$. If $d(F) = 6$ one can show as in previous lemmas that the branching number is less than 2. If $d(F) = 5$, Lemma 3 guarantees that there will be a variable x with $\alpha(x)/\beta(x) > 4$. The minimum value of $S(x)$ satisfying this property is 23. We can again handle Intervals 8-12 as in previous lemmas (also handled by [10]). The worst case running time of C-2-Sat$_6$ with $m \leq 5n$ is $\tilde{O}(2^{\chi_{13}n}) \approx O(1.2411^n)$. ❑

Lemma 10 *For a maximally reduced formula F with $5n < m \leq 6n$, C-2-*Sat$_6(F, C, W)$ *runs in* $\tilde{O}(2^{\chi_{19}n})$ *time.*

Proof. We know that $d(F) = 6$. In this lemma measures $f_{13}(m, n)$ to $f_{18}(m, n)$ are used. The minimum value of $S(x)$ for variables with $\alpha(x)/\beta(x) > 5$ is 33. We can again handle Intervals 13-18 as in previous lemmas. This results in $\chi_{19} \approx 0.317$. The worst case running time of C-2-Sat$_6$ with $m \leq 6n$ is $\tilde{O}(2^{\chi_{19}n}) \approx O(1.2461^n)$. ❑

Putting together Lemmas 6, 7, 8, 9, and 10 we get that C-2-Sat$_6$ has a worst case running time of $O(2^{n/3.15})$. The main Algorithm C-2-Sat (Figure 4) checks for three different cases.

Theorem 1 *Algorithm C-2-*Sat(F, C, W) *runs in* $O(1.2461^n)$ *time.*

Proof. The runtime bound $T(n)$ of C-2-Sat(F, C, W) is of the form:

$$T(n) \leq$$
$$\max\{T(n-2) + T(n-6), 1.2461^n, T(n-1) + T(n-8)\},$$

where the first term is from Step 4, the second is from Step 5, and the last is from Step 6. Note in Step 6 we branch on a variable with degree greater than 6. The recurrence $T(n) = T(n-2) + T(n-6)$ solves to $T(n) = \tilde{O}(\tau(2, 6)^n) \approx O(1.2106^n)$, and the recurrence $T(n) = T(n-1) + T(n-8)$ solves to $T(n) = \tilde{O}(\tau(1, 8)^n) \approx O(1.2320^n)$. Therefore the Algorithm C-2-Sat runs in $O(1.2461^n)$ time. ❑

3.3 Applications of Improved Algorithm for #2-Sat

We now summarize some other interesting problems whose running time have improved as a direct consequence of our result. We present only the definitions and results but not the reductions. The first result is consequence of a standard reductions (also presented in [10]), the second result is from [4, 1], and the last one is from [7].

1. **#Maximum Weighted Independent Set** and **#Independent Set** can be solved in $O(1.2461^n)$ time.
   Instance: Graph G=(V,E) with a weight $w(x)$ for each vertex $x \in V$.
   Solution: Subset of vertices $V' \subseteq V$ s.t. no vertices have an edge between them. Count the number of maximum weighted independent sets or independent sets.

2. **Max Hamming Distance** $(2, 2)$-Csp can be solved in $O(1.7200^n)$ time.
   Instance: $(2, 2)$-Csp.

Solution: Given a set of variables X, the hamming distance between a pair of assignments f_1 and f_2 of values to variables in X, denoted by $d_H(f_1, f_2)$, is the number of variables on which the two assignments disagree. The max hamming distance $(2,2)$-Csp problem is to find two satisfying assignments f, g of the Csp instance which maximizes $d_H(f, g)$.

3. **Chromatic Number** of a graph can be determined in $O(2.2461^n)$ time.
Instance: Graph G=(V,E).
Solution: Smallest integer k such that there is a mapping $V \to \{1, \ldots, k\}$ that gives different values to neighboring vertices.

These problems do not however constitute an exhaustive list or problems that can be reduced to 2-Sat. The reader can easily come up with other interesting problems (see for e.g. [10, 19]).

4. Counting Solutions in Constraint Satisfaction Problems

In this section we use the results for 2-Sat to obtain better worst case bounds for counting the number of solutions of a weighted binary constraint satisfaction problem. First note that the result obtained for 2-Sat, is applicable for any weighted binary constraint satisfaction problem, whose clauses are over pairs of binary variables ($(2,2)$-Csp). So we now concentrate on the case where $d \geq 3$. We use the same approach as in [2, 3]. The idea is to partition the domains of the variable into sub-domains of sizes between 2 and 5 elements. The algorithm for #2-Sat is used for solving these smaller instances and the results are recombined. The decrease in running times from [2] is because of the faster running times for #2-Sat and better partitioning of the domains. In the following analysis we use α^n to indicate the running time for #2-Sat, i.e., $\alpha = 1.2461$.

Transformation to #2-Sat: A $\#(d,2)$-Csp instance can be solved by transforming it into a weighted #2-Sat instance. The transformation from [2] is reproduced for completeness:

1. If x is single valued we can remove it by assigning 1 to x and by propagating (as in Propagate).

2. If x is two-valued, we introduce a propositional variable x_1 with the interpretation that $x = 1$ if x_1 is true and $x = 2$ otherwise.

3. If x is k-valued, we create k propositional variables $x_1, \ldots, x_k$ (also called propositions) with the interpretation that $x = i$ if x_i is true. Also, we introduce clauses to ensure that at most one of them is true.
$$\bigwedge_{i \leq j,\, i,j \in \{1,\ldots,k\}} (\neg x_i \vee \neg x_j).$$

4. For every constraint $xRy \in C$, with x having domain D_x and y having domain D_y, we add the clauses
$$\bigwedge_{a \in D_x, b \in D_y, (a,b) \notin R} (\neg x_a \vee \neg y_b).$$

If one of the variables is two-valued, we need to take into account that its negation also corresponds to an assignment. For, e.g., if x is two-valued and $y \in D_y$, we introduce clauses like:

$$\bigwedge_{b \in D_y, (0,b) \notin R} (\neg x_0 \vee \neg y_b) \wedge \bigwedge_{b \in D_y, (1,b) \notin R} (x_0 \vee \neg y_b).$$

Cases where both the variables are two-valued can be handled similarly.

In this context it is possible that the new variables $x_1, \ldots, x_k$ corresponding to a k-valued variable x ($k > 2$) may all be false and thus x would not get an assignment. To remedy this, weights are introduced: weight 0 is assigned to each proposition corresponding to a two-valued variable, and 1 for all propositions corresponding to higher valued variables. The running time for #2-Sat is $\tilde{O}(1^{k_1} \cdot \alpha^{k_2} \cdot \alpha^{3k_3} \ldots)$, where k_i indicates number of i-valued variables. The search is for satisfying assignments having a weight of $\sum_{i \geq 3} k_i$, i.e., maximum possible weight. This immediately provides an $\tilde{O}(\alpha^{dn})$ time algorithm for $\#(d,2)$-Csp.

Partitioning the Domain: Angelsmark [2] however noted that the result can be speeded up using partitioning. The following theorem (stated without proof) from [2] demonstrates how partitions can be used for solving $\#(d,2)$-Csp.

Theorem 2 *(Angelsmark et al. [2]) Let A be an algorithm for $\#(p,2)$-Csp running in $\tilde{O}\left(\prod_{i=1}^{p} \alpha_i^{n_i}\right)$ time ($\alpha_i \geq 1$) when applied to an instance containing n_i, i-valued variables for $1 \leq i \leq p$. Choose d and partitioning $P = \{P_1, \ldots, P_k\}$ of $\{1, \ldots, d\}$ such that $|P_i| \leq p$ for every i. Then there exists an algorithm for $\#(d,2)$-Csp running in $\tilde{O}\left(\left(\sum_{i=1}^{p} \sigma(P,i)\alpha_i\right)^n\right)$ time, where $\sigma(P,i)$ denotes the number of parts of size i in P.*

Corollary 1 *Given a partitioning P of a domain D containing d elements with $\sigma(P,i)$ denoting the number of parts of size i in P, the $\#(d,2)$-Csp can be solved in $\tilde{O}(T(P)^n)$, where*

$$T(P) = \sigma(P,1) + \sigma(P,2)\alpha + \sum_{i=3}^{d} \sigma(P,i)\alpha^i.$$

Optimal Partitioning for α: We let the multi set $[|P_1|, \ldots, |P_k|]$ represent the partition P. The following theorem defines the optimal partitioning (i.e., minimizing $T(P)$) with the current best bounds of α.

Theorem 3 *Let D be a domain of size $d \geq 2$. If $d < 6$, then the partitioning $P = [d]$ is optimal. Otherwise, the following partitions with $k \geq 1$ are optimal:*
a) *if $d = 5k$, $P = [5, 5, \ldots, 5]$ (k parts of size 5),*
b) *if $d = 5k+1$, $d \geq 16$, $P = [4, 4, 4, 4, 5, 5, \ldots, 5]$ ($k-3$ parts of size 5) ,*
c) *if $d = 5k+2$, $d \geq 12$, $P = [4, 4, 4, 5, 5, \ldots, 5]$ ($k-2$ parts of size 5),*
d) *if $d = 5k+3$, $P = [4, 4, 5, 5, \ldots, 5]$ ($k-1$ parts of size 5),*
e) *if $d = 5k+4$, $P = [4, 5, 5, \ldots, 5]$ (k parts of size 5),*
f) *if $d = 6$ then $P = [4, 2]$, if $d = 7$ then $P = [5, 2]$, if $d = 11$ then $P = [5, 4, 2]$.*

Proof. If P is not optimal then there exists P^* which is a strictly better partition. We show that such a P^* does not exist and, consequently, P is optimal. One can easily show

Time complexity	*Domain size*
$O(((d/5) \cdot \alpha^5)^n)$	$d = 5k, k \geq 1.$
$O((4\alpha^4 + \lfloor d/5 - 3 \rfloor \cdot \alpha^5)^n)$	$d = 5k+1, k \geq 3.$
$O((3\alpha^4 + \lfloor d/5 - 2 \rfloor \cdot \alpha^5)^n)$	$d = 5k+2, k \geq 2.$
$O((2\alpha^4 + \lfloor d/5 - 1 \rfloor \cdot \alpha^5)^n)$	$d = 5k+3, k \geq 1.$
$O((\alpha^4 + \lfloor d/5 \rfloor \cdot \alpha^5)^n)$	$d = 5k+4, k \geq 1.$

Figure 10. Time Complexity for #$(d, 2)$-CSP

as in [2] that P^* contains no parts of size 1. The idea is to use induction over parts of size greater than 1. This immediately implies that the domain sizes $1 \leq d \leq 3$ should not be partitioned further. Also domain sizes of 4 and 5 should not be partitioned further because $T([4]) < T([2,2])$ and $T([5]) < T([3,2])$. The following four steps complete the proof:
Step 1: P^* contains only parts of size a, $a \in \{2,4,5\}$ for $d > 3$.
The proof idea is to show inductively that $T([a, p_1, \ldots, p_k]) > T([a-2, 2, p_1, \ldots, p_k])$ for $a > 5$. So it is better to partition any domain size greater than 5, implying that optimal partitions only contain parts of size 2, 3, 4 and 5. Assume $P^* = [3, a, \ldots]$ where $a \in \{2,3,4,5\}$. It can be checked that $T([3,2]) > T([5])$, $T([3,3]) > T([4,2])$, $T([3,4]) > T([5,2])$, $T([3,5]) > T([4,4])$, so P^* contains no parts of size 3 for $d > 3$.
Step 2: P is optimal for $d = 6, 7$ and 11.
For $d \in \{6, 7, 11\}$ there is only one way of partitioning with parts of size a, where $a \in \{2,4,5\}$. So P is optimal for the above choices.
Step 3: P^* contains only parts of size a, $a \in \{4,5\}$, when $d > 5$ and $d \notin \{6,7,11\}$.
P^* can't have more than one parts of size 2 because $T([2,2]) > T([4])$. Also, P^* can't have any partition of the form $[5,5,2]$ (as $T([5,5,2]) > T([4,4,4])$) and of the form $\{4,4,2\}$ (as $T([4,4,2]) > T([5,5]))$. Implying that P^* has no parts of size 2 for $d > 5$ and $d \notin \{6,7,11\}$.
Step 4: P is optimal when $d > 5$ and $d \notin \{6,7,11\}$.
P^* can't have more than four parts of size 4 because $T([4,4,4,4,4]) > T([5,5,5,5])$. P^* can't have fewer parts of size 4, since otherwise it would need parts of size different from 4 and 5.
All the above imply that, $P = P^*$, so partitioning P is optimal. ❑

Figure 10 summarizes the improved time complexities. For large domains, the term $\frac{d}{5} \cdot \alpha^5$ dominates. Consequently, the bounds approach $\tilde{O}((\frac{d}{5} \cdot \alpha^5)^n) \approx O((0.6009d)^n)$.

In the Figure 11 we compare the improvement in running times for sample domain sizes.

5. Counting Colorings in Graphs

We now present an algorithm for counting the number of colorings of a graph. We first present a faster algorithm for #3-COL and show how it results in faster algorithm for #k-COL. Note that results from the previous section implies an upper bound of $O(1.935^n)$ for the problem of #3-COL.

Let $G = (V(G), E(G))$ be the input graph. We choose the colors from the set $\{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$. Denote by $\mathcal{I}(G)$ a maximum independent set in G. We define a $\mathcal{R}\{\mathcal{G}/\mathcal{B}\}$ assignment as a total function $\mathcal{S} : V(G) \to \{\mathcal{R}, \mathcal{GB}\}$ with $\mathcal{S}^{-1}(\mathcal{R}) = \mathcal{I}(G)$. Once we have a $\mathcal{R}\{\mathcal{G}/\mathcal{B}\}$

Domain Size	*Previous Time*	*Improved Time*
2	1.2561^n	1.2461^n
3	1.9819^n	1.9348^n
4	2.4895^n	2.4109^n
5	3.1270^n	3.0041^n
10	6.2350^n	6.0081^n
25	15.5740^n	15.0204^n

Figure 11. Time Complexity Comparison for #$(d, 2)$-CSP.

```
Algorithm C-3-COL($G$)
(Assume $\mathcal{I}(G)$ is the maximum independent set in $G$)

1) If $|\mathcal{I}(G)| \leq c \cdot |V(G)|$ then
    a) If all $v \in V(G)$ are $\mathcal{R}\{\mathcal{G}/\mathcal{B}\}$-colored, then return $Count(G, S)$.
    b) Else if there exists an uncolored vertex $x$ with $U$ as the set of its uncolored neighbors
       and $U \neq \emptyset$, then return (C-3-COL($G[x = \mathcal{R}, U = \mathcal{GB}]$) + C-3-COL($G[x = \mathcal{GB}]$)).
    c) Else if there is an uncolored vertex $x$, then return
       (C-3-COL($G[x = \mathcal{R}]$) + C-3-COL($G[x = \mathcal{GB}]$)).
2) Else
    a) Find the induced graph $G'$ of $V(G) \setminus \mathcal{I}(G)$.
    b) Find a maximal independent set $I(G')$ in $G'$, then cycle through all the possible
       3-colorings of $I(G')$ and permitted $\{\mathcal{G}, \mathcal{B}\}$-colorings of $V(G') \setminus I(G')$.
    c) For every 3-coloring $\mathcal{S}$ of $G'$ do
       $out \leftarrow out + \prod_{x \in \mathcal{I}(G)} (3 - |\{\mathcal{S}(w) : w \in N_G(x) \setminus \{x\}\}|)$.
    d) Return $out$.
```

Figure 12. Algorithm C-3-COL.

assignment, it can be tested in polynomial time whether it is possible to reassign colors $\mathcal{G}$ or $\mathcal{B}$, to all vertices v with the initial color $\mathcal{S}(v) = \mathcal{GB}$. The number of such reassignments is denoted by $Count(G, S)$. It equals 0 if no such reassignment is possible, otherwise equals 2^d with d as the number of connected components in the subgraph of G induced by set of vertices $\{u : S(u) = \mathcal{GB}\}$.

The algorithm (Figure 12) starts with finding a maximum independent set $\mathcal{I}(G)$. The constant $c = 0.4242$ (from Theorem 4). The "if part" (Step 1) of the Algorithm C-3-COL is similar to Algorithm #3C-1 of [2]. Our improvement comes from the better handling of the "else part" (Step 2). Let G' be the subgraph of G induced by the set of vertices $V(G) \setminus \mathcal{I}(G)$. In Step 2, we wish to enumerate all the possible 3-colorings of the graph G'. Instead of doing it in a straightforward manner as in [2], we make a detour. We find a maximal independent set $(I(G'))$ in G'. Let $V(G')$ be the set of vertices of G'. We search through all possible 3-colorings of $I(G')$ and all permitted $\{\mathcal{G}, \mathcal{B}\}$-colorings of $V(G') \setminus I(G')$. The proof of correctness is straightforward and can be shown as in [2].

Theorem 4 *Algorithm C-3-*Col(G) *runs in* $O(1.7702^n)$ *time.*

Proof. Using Beigel's algorithm [5], we can find the maximum independent set $\mathcal{I}(G)$ in $O(1.222^n)$ time. The analysis of C-3-Col has two parts.
Case 1: If $|\mathcal{I}(G)| \leq c \cdot |V(G)|$.
The worst case time bound of Step 1b) for $U \neq \varnothing$ has the Fibonacci form

$$T(n) \leq T(n-1) + T(n-2),$$

and therefore solves to $\tilde{O}(\phi^n)$, where ϕ is the golden ratio[5]. If Step 1c) of the algorithm is reached, the uncolored vertices form an independent set S. Therefore, the running time of this part is $\tilde{O}(2^{|S|} \cdot \phi^{|V(G)|-|S|})$. The worst case occurs when $|S| = c \cdot |V(G)|$ and the resultant worst case running time is $\tilde{O}(2^{cn} \cdot \phi^{(n-cn)})$.
Case 2: If $|\mathcal{I}(G)| > c \cdot |V(G)|$.
Since $\mathcal{I}(G)$ is a maximum independent set and $I(G')$ is a maximal independent set, $|I(G')| \leq |\mathcal{I}(G)|$. We can enumerate all possible 3-colorings of G' in $\tilde{O}(3^{|I(G')|} \cdot 2^{|V(G)|-|\mathcal{I}(G)|-|I(G')|})$ time. If $|\mathcal{I}(G)| \geq \frac{|V(G)|}{2}$, then the worst case occurs when $|I(G')| = |V(G)| - |\mathcal{I}(G)| = \frac{|V(G)|}{2}$ and the resultant worst case running time is $\tilde{O}(3^{n/2})$.

The more interesting case is when $|\mathcal{I}(G)| < \frac{|V(G)|}{2}$. Let $c' \cdot |V(G)| = |I(G')| \leq |\mathcal{I}(G)|$. The worst case for this part occurs when $|I(G')| = |\mathcal{I}(G)|$ and the resultant worst case running time is $\tilde{O}(3^{cn} \cdot 2^{(n-2cn)})$.

Finally, by solving $2^c \cdot \phi^{1-c} = 3^c \cdot 2^{1-2c}$, we obtain the values of $c = c' = 0.4242$. So the Algorithm C-3-Col runs in $O(1.7702^n)$ time. ❑

The improvement in the running time for #3-Col automatically improves the running time of the best known algorithm of Angelsmark *et al.* [2] for #k-Col. The following theorem and its subsequent corollary summarizes this improvement.

Theorem 5 *(Angelsmark et al. [2]) There exists an algorithm for solving the* #k-Col *problem in* $\tilde{O}((c_k)^n)$ *time, where, for some* $i \in \mathbb{N}$

$$c_k = \begin{cases} \lfloor \log_2 k \rfloor & \text{if } k = 2^i, \\ \lfloor \log_2 k \rfloor + (\beta - 1) & \text{if } 2^i < k \leq 2^i + 2^{i-1}, \\ \lfloor \log_2 k \rfloor + 1 & \text{if } 2^i + 2^{i-1} < k < 2^{i+1}. \end{cases}$$

with β^n *as the running time of #3-*Col.

Corollary 2 *There exists an algorithm for solving the* #k-Col *problem in* $\tilde{O}((c_k)^n)$ *time, where, for some* $i \in \mathbb{N}$

$$c_k = \begin{cases} \lfloor \log_2 k \rfloor & \text{if } k = 2^i, \\ \lfloor \log_2 k \rfloor + .7702 & \text{if } 2^i < k \leq 2^i + 2^{i-1}, \\ \lfloor \log_2 k \rfloor + 1 & \text{if } 2^i + 2^{i-1} < k < 2^{i+1}. \end{cases}$$

5. $\phi = \frac{1+\sqrt{5}}{2}$

6. Concluding Remarks

We presented an algorithm with improved bound for counting satisfying assignments of a weighted 2-Cnf formula. The improved bounds for this problem leads to better bounds for many interesting problems, including that of counting solutions of $(d, 2)$-Csp. We also presented a faster algorithm for counting 3-colorings of a graph.

An obvious open problem is to obtain algorithms with improved running time. To improve the running time for #2-Sat one could think of utilizing dynamic programming. By using exponential space algorithms, one could expect to bring the time complexity down considerably. For practical purposes, exponential space algorithms are of course highly undesirable. Other way could be to figure out a way to characterize the Interval boundaries better and to perform a more careful analysis of the neighborhood of a branching vertex. For #3-Col, it might be possible to further decrease the running time for some special classes of graphs, e.g., for bipartite graphs it is not hard to show that the algorithm runs in $\tilde{O}(3^{n/2})$ time.

References

[1] Angelsmark, O. Constructing algorithms for constraint satisfaction and related problems : Methods and applications. *PhD. thesis, Linköping University* (2005).

[2] Angelsmark, O., and Jonsson, P. Improved algorithms for counting solutions in constraint satisfaction problems. In *International Conference on Constraint Programming* (2003), vol. 2833, Springer, pp. 81–95.

[3] Angelsmark, O., Jonsson, P., Linusson, S., and Thapper, J. Determining the number of solutions to binary CSP instances. In *International Conference on Constraint Programming* (2002), vol. 2470, Springer, pp. 327–340.

[4] Angelsmark, O., and Thapper, J. Algorithms for the maximum hamming distance problem. In *International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP)* (2004), vol. 3978, Springer, pp. 128–141.

[5] Beigel, R. Finding maximum independent sets in sparse and general graphs. In *Symposium on Discrete Algorithms* (1999), SIAM, pp. 856–857.

[6] Beigel, R., and Eppstein, D. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms 54*, 2 (2005), 168–204.

[7] Björklund, A., and Husfeldt, T. Inclusion–exclusion algorithms for counting set partitions. *Foundations of Computer Science* (2006), 575–582.

[8] Dahllöf, V., and Jonsson, P. An algorithm for counting maximum weighted independent sets and its applications. *Symposium on Discrete Algorithms* (2002), 292–298.

[9] Dahllöf, V., Jonsson, P., and Wahlström, M. Counting satisfying assignments in 2-SAT and 3-SAT. In *Annual International Computing and Combinatorics Conference (COCOON)* (2002), vol. 2387, Springer, pp. 535–543.

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

[11] Dantsin, E., Gavrilovich, M., Hirsch, E., and Konev, B. MAX SAT approximation beyond the limits of polynomial-time approximation. *Annals of Pure and Applied Logic 113* (2002), 81–94.

[12] Davis, M., Logemann, G., and Loveland, D. A machine program for theorem-proving. *Communications of the ACM 5*, 7 (1962), 394–397.

[13] Davis, M., and Putnam, H. A computing procedure for quantification theory. *Journal of Association Computer Machinery 7* (1960), 201–215.

[14] Dubois, O. Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science 81*, 1 (1991), 49–64.

[15] Dyer, M., and Greenhill, C. The complexity of counting graph homomorphisms. *Random Structures and Algorithms 17* (2000), 260–289.

[16] Dyer, M., and Greenhill, C. Corrigendum: The complexity of counting graph homomorphisms. *Random Structures and Algorithms 25* (2004), 346–352.

[17] Feder, T., and Motwani, R. Worst-case time bounds for coloring and satisfiability problems. *Journal of Algorithms 45*, 2 (2002), 192–201.

[18] Feige, U., Goldwasser, S., Lovász, L., Safra, S., and Szegedy, M. Approximating clique is almost NP-complete. In *Foundations of Computer Science* (1991), IEEE Computer Society, pp. 2–12.

[19] Fürer, M., and Kasiviswanathan, S. P. Algorithms for counting 2-SAT solutions and colorings with applications. TR05-033, Electronic Colloquium on Computational Complexity, 2005.

[20] Fürer, M., and Kasiviswanathan, S. P. Exact Max 2-SAT: Easier and faster. In *Current Trends in Theory and Practice of Computer Science (SOFSEM)* (2007), vol. 4362, Springer, pp. 272–283.

[21] Håstad, J. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica 182*, 1 (1999), 105–142.

[22] Jeavons, P. G., Cohen, D. A., and Pearson, J. K. Constraints and universal algebra. *Annals of Mathematics and Artificial Intelligence 24* (1998), 51–67.

[23] Koivisto, M. An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion. *Foundations of Computer Science* (2006), 583–590.

[24] Kozen, D. C. *The Design and Analysis of Algorithms.* Springer, Berlin, 1992.

[25] Kullmann, O. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science 223* (1999), 1–72.

[26] Kumar, V. Algorithms for constraint-satisfaction problems: A survey. *Artificial Intelligence Magazine 13*, 1 (Spring 1992), 32–44.

[27] Littman, M. L., Pitassi, T., and Impagliazzo, R. On the complexity of counting satisfying assignments. In *The Working notes of LICS 2001 Workshop on Satisfiability* (2001).

[28] Roth, D. On the hardness of approximate reasoning. *Artificial Intelligence 82*, 1–2 (1996), 273–302.

[29] Vadhan, S. P. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing 31*, 2 (2002), 398–427.

[30] Valiant, L. G. The complexity of computing the permanent. *Theoretical Computer Science 8*, 2 (1979), 189–201.

[31] Valiant, L. G. The complexity of enumeration and reliability problems. *SIAM Journal of Computing 8*, 3 (1979), 410–421.

[32] Vigoda, E. Improved bounds for sampling colorings. *Journal of Mathematical Physics 41*, 3 (2000), 1555–1569.

[33] Williams, R. A new algorithm for optimal constraint satisfaction and its implications. In *International Colloquium on Automata, Languages and Programming* (2004), vol. 3142, Springer, pp. 1227–1237.

[34] Zhang, W. Number of models and satisfiability of sets of clauses. *Theoretical Computer Science 155*, 1 (1996), 277–288.