Electronic Colloquium on Computational Complexity, Report No. 33 (2005)

Algorithms for Counting 2-SAT Solutions and Colorings with Applications

Martin Fürer* and Shiva Prasad Kasiviswanathan

Computer Science and Engineering, Pennsylvania State University
University Park, PA, 16802
{furer, kasivisw}@cse.psu.edu

Abstract. An algorithm is presented for counting the number of maximum weight satisfying assignments of a 2SAT formula. The worst case running time of $O(\text{poly}(n) \cdot 1.2461^n)$ for formulas with n variables improves on the previous bound of $O(\text{poly}(n) \cdot 1.2561^n)$ by Dahllöf, Jonsson, and Wahlström . The weighted 2SAT counting algorithm can be applied to obtain faster algorithms for many combinatorial counting problems, including those of counting maximum weighted independent sets, exact covers, weighted set packings. 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. Also presented is an improved algorithm for counting 3-colorings in a graph. The upper bound of $\text{O}(\text{poly}(n) \cdot 1.7702^n)$ for graphs with n vertices improves on the previous bound of $\text{O}(\text{poly}(n) \cdot 1.7879^n)$ 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 [6] or graph coloring [12]. Many important problems can be modeled by these instances, and with Moore's law playing its part, bigger instances of these problems can be solved efficiently. Improvements in the exponential bounds are critical, for even a slight improvement from $\text{O}(c^n)$ to $\text{O}((c-\epsilon)^n)$ 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 with in a factor of $\text{O}(n^{1-\epsilon})$ in polynomial time [14, 15]. Approximation algorithms for NP-hard decision problems like satisfiability or graph k-colorability are nonsensical as pointed out by [13].

Most of the super-polynomial algorithms known are only for decision problems. As an 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 [21, 22]. In his seminal work Valiant [23] proposed the class #P and showed that computing the permanent is #P-complete.

The decision problem of weighted 2SAT is NP-hard. #2SAT is known to be #P-complete [17, 24]. Earlier works on counting models for problems like SAT include papers by Dubois [9], Zhang [26], Littman *et al.* [19]. The algorithm by Zhang[26] runs in time $\text{O}(1.6180^n)$ for #2SAT, where as the algorithm by Littman *et al.* runs in time $\text{O}(1.381^n)$.

* Research supported in part by NSF Grant CCR-0209099

ISSN 1433-8092

In a recent paper Dahllöf *et al.* [5] improved the upper bound to $O(1.2561^n)$ for weighted #2SAT (a.k.a. #2SAT$_w$). In this paper we extend the series of work done by Dahllöf *et al.* [3–5] on this problem. Our algorithm for counting models and max-weight models uses polynomial space and runs in time $O(1.246069^n)$. The improvements are achieved by using a simpler and improved analysis of the framework introduced in [5]. Improving the worst case bound automatically improves the previous best running times for interesting problems like *#Maximum Independent Set, #Exact Cover, #Exact Hitting Set, #Weighted Set Packing.*

To understand our improvement in the running time, it is good to focus on the subproblem of #2SAT with a restriction to at most three occurrences of every variable, i.e., the corresponding graph 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 $\frac{12}{5}$, we can find a degree 3 vertex with a neighbor of degree 3, and we can eliminate both at once. We take advantage of this by choosing a different complexity measure above $\frac{12}{5}$. 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 with the framework of Dahllöf *et al.* [5].

The decision problem for (d, l)-CSP is NP-complete and the corresponding counting problem belongs to #P-complete even when restricted to binary CSP [21]. From [21], 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 [16] and finding the number of graph homomorphisms has important applications in statistical physics [10, 11, 25], 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 [10] for detailed description of these models). In artificial intelligence, problems like approximate reasoning [21] can be viewed as #CSP-instances. In this paper we extend the series of algorithms presented by Angelsmark *et al.* [1, 2] for counting the number of solutions of binary CSPs. The algorithm presented by [1] partitions the domains of the variables into sub domains of sizes between 2 and 5 elements and uses #2SAT$_w$ for solving these smaller instances. We use the same algorithm as in [1], however the speedup (see Table 2 for numerical values) is due to improved #2SAT$_w$ algorithm and by using an improved partitioning for the domains.

We also provide an improved algorithm for counting the number of 3-colorings in a graph, which runs in time $O(1.7702^n)$, an improvement over the $O(1.7879^n)$ algorithm from [1]. Finding a k-coloring of a graph G is equivalent to finding a homomorphism from G to a complete graph with k vertices. The speedup is achieved by exploiting problem specific knowledge. The improvement is more important because it automatically improves the running time of #k-coloring given by [1].

The paper is organized as follows: Sections 2 to 3 are devoted to the #2SAT$_w$ problem. In Section 2, we define some preliminaries and technical tools. In Section 3 we present the algorithm for #2SAT$_w$ and analyze its performance. Section 4, defines some interesting problems whose running times have improved as a consequence. In Section 5, we present

an improved analysis for counting solutions to binary constraint satisfaction problems and in Section 6, we present an improved algorithm for counting 3-colorings.

2 Preliminaries

#2SAT is the problem of computing the number of satisfying assignments or models for a 2SAT formula. With each literal l, a weight $w(l) \in \mathbb{N}$ and a count $c(l) \geq 1$ is associated; the vectors W and C are the corresponding vectors. Let L is the set of literals, we define the weight of a model M as

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

and cardinality of a model M as

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

The problem is here to count the number of maximum weight models (a.k.a. MWM). $Var(F)$ denotes the variable set of F and $n(F) = |Var(F)|$. A variable which occurs only as x or $\neg x$ is called a monotone. Given a formula F, we define the constraint graph $G = (Var(F), E)$, as an undirected graph where the vertex set is the set of variables and the edge set is $\{(x, y) \,|\, x, y \text{ appear in the same clause } F|\}$. The degree $d(x)$ of a variable x is the number of clauses in F containing x. 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. The neighborhood of a vertex x in graph G, denoted by $N_G(x)$, is the set $\{y \,|\, (x, y) \in E\} \cup \{x\}$. The size of the neighborhood of x, $S(x) = \sum_{y \in N_G(x)} d(y)$. We define $m(F)$ as

$$m(F) = \sum_{x \in Var(F)} d(x)$$

Both $n(F)$ and $m(F)$ are used as measures of formula complexity. For $\mathcal{M}$ being the set of all MWM for F, define

$$\#2SAT_w(F, C, W) = \left(\sum_{M \in \mathcal{M}} \mathcal{C}(M), \mathcal{W}(M') \right)$$

where M' is any MWM in $\mathcal{M}$.

2.1 Estimation of Tree Size

The idea behind the algorithm is recursive decomposition based on the DPLL algorithm [7], which itself was based on the Davis-Putnam algorithm [18]. The recurrent idea is to choose a variable $x \in \mathrm{Var}(F)$ and to recursively count the number of satisfying assignments where x is true or x is false, i.e., we branch on x. We follow the analysis of Kullmann [8]. In the implicit branching tree constructed, let v be a node with d branches labeled with positive real numbers $t_1, \ldots, t_d$. The labels are the measures of the reduction in complexity in the respective branch. The branching number is the largest, real-valued solution of $\sum_{i=1}^{d} x^{-t_i} = 1$. For a branching tuple $(t_1, \ldots, t_d)$ the branching number is denoted by

Function Propagate(F, C, W)

1) if F contains an empty clause then $F = \{\emptyset\}$, $c = 0$ and $w = 0$
2) if there is a clause $(1 \vee \ldots)$ then it is removed, Any variable a which gets removed is handled according to cases
   a) if $w(a) = w(\neg a)$ then $c = c \cdot (c(a) + c(\neg a))$; $w = w + w(a)$.
   b) if $w(a) < w(\neg a)$ then $c = c \cdot (c(\neg a))$; $w = w + w(\neg a)$.
   c) if $w(a) > w(\neg a)$ then $c = c \cdot c(a)$; $w = w + w(a)$.
3) if there is a clause of the form $(0 \vee \ldots)$ remove 0 from it.
4) if there is a clause of the form (a) then remove it and $c = c \cdot c(a)$; $w = w + w(a)$, and, if a still appears in F then $F = F[a = 1]$.
5)return(F, c, w)

Function Reduction(F, v)

Let $F = F_1 \wedge F_2$ with $Var(F_1) \cap Var(F_2) = v$
1) let $(c_t, w_t) = \#2\text{SAT}_w(F_1[v = 1], C, W)$ and $(c_f, w_f) = \#2\text{SAT}_w(F_1[v = 0], C, W)$.
2) modify C as $c(v) = c_t \cdot c(v)$, $c(\neg v) = c_f \cdot c(\neg v)$, W as $w(v) = w_t + w(v)$, $w(\neg v) = w_f + w(\neg v)$
3) return $\#2\text{SAT}_w(F_2, C, W)$.

$\tau(t_1, \ldots, t_d)$. In this paper we only branch 2-fold.

In this paper the branch from F to F_i is labeled by $t_1 = \triangle f(F) = f(F) - f(F_i)$, where $f(F)$ is some algorithm specific measure of complexity. Defining $f_{max}(n) = max_{n(F)=n} f(F)$, ensures us a running time of $\text{O}(\text{poly(n)} \cdot \alpha^{f_{max}(n)})$, where α is the largest branching number occurring in any tuple in the tree. We will define the function f such that the worst case branching number is $\tau(1, 1) = 2$. In the special case where we make equal progress to F_1 and F_2 in a 2-fold branching, this means that $\triangle f(F) = f(F) = f(F_i) = 1$ (for $i = 1, 2$). Let l, l' and l'' be some literals of F. In a step satisfying literal $l \in F$, we eliminate all the clauses of the form $(l \vee l')$ and if we don't, we eliminate all the clauses of the form $(\neg l \vee l'')$. We call a branching as *maximally unbalanced* if clauses of only one form occur.

2.2 Structures & Helper Function

We use similar functions and structures as in [5], some of which has been reproduced for completeness. Integer variables $c \geq 0$ and $w \geq 0$ are used for keeping track of the contribution to the number and weight of the models arising from the eliminated variables. We also maintain a count vector (C) and a weight vector (W). The first function called Propagate simplifies the formula by removing dead variables. It returns the updated formula F', the weight w of the variables removed and count c for the eliminated variables. Another function called Reduction reduces the input formula. It takes advantage of the fact that if there exists a F that can be partitioned into F_1 and F_2 such that each clause belongs to either of them, and $|Var(F_1) \cap Var(F_2)| = 1$, then we can remove F_1 while appropriately updating c and w of the common variable. We apply both these routines to F as long as applicable. A formula F is called a *maximally reduced* formula if neither of these routines apply. It can be easily shown as in [5] that the value of $\#2\text{SAT}_w(F, C, W)$ is preserved under both these routines.

We have a main algorithm $C2SAT$ which is split into two functions depending on $d(F)$. The main function $C2SAT$, is used whenever $d(F) > 6$. It has a helper routine: $C2SAT_6$ which is used when $d(F)$ =3,4,5 or 6. In all our algorithms (because of the bookkeeping involved) the process of branching on a variable is lengthy, therefore we will use the phrase *branch on* v as a shorthand for the following:

let (F_t, c_t, w_t) = Propagate $(F[v = 1], C, W)$ and (F_f, c_f, w_f)=Propagate $(F[v = 0], C, W)$.
let $(c'_t, w'_t) = C2SAT(F_t, C, W)$ and $(c'_f, w'_f) = C2SAT(F_f, C, W)$.
let $W_{true} = w(v) + w_t + w'_t$, $W_{false} = w(\neg v) + w_f + w'_f$, $C_{true} = c(v) \cdot c_t \cdot c'_t$, and $C_{false} = c(\neg v) \cdot c_f \cdot c'_f$. There are 3 cases:
if $W_{true} = W_{false}$, return$(C_{true} + C_{false}, W_{true})$.
else if $W_{true} > W_{false}$, return(C_{true}, W_{true}).
else $W_{true} < W_{false}$, return(C_{false}, W_{false}).

The proof of correctness of the algorithm $C2SAT$ is straight forward and can be found in [5].

3 Algorithm

In the analysis of C2SAT$_6$ we use a continuous and piecewise linear function $f(n, m)$ similar to the one introduced by [5] as a measure of complexity. 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 worst case branching numbers are smaller near the top of the tree and increases as we go down. The estimation of the running time as $O(\alpha^{f_{max}(n)})$ (with $f_{max} = \max\{f(n, m) \mid m \in \mathbb{N}\}$) is best when the branching numbers are uniform throughout. The complexity measure introduced by [5] incorporates the effects of decreasing the m/n quotient in the upper bound, leading to 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 a upper limit k_{i+1} for the m/n above which that worst case can't occur. There could be still be vertices x with such bad neighborhoods, but the algorithm would not select such an x to branch on. As in [5], we define the coefficient χ_i by $\chi_i = a_i + k_i b_i$ implying $f(n, m) = f_i(n, m) = f_{i-1}(n, m) = \chi_i n$ for $m/n = k_i$. Now $f_i(n, m)$ can also be expressed as $f_i(n, m)$ as $\chi_i n + (m - k_i n)b_i$. We define a *Section* i as the range k_i to k_{i+1}. The formal definitions of the functions are:

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

Following are some interesting properties of $f(n, m)$ used in the analysis. The first two properties can be observed from the Table 1 and Property 3 is from [5].
Properties:

1) $f(n, m) > f(n - 1, m)$ if $m > 3.75n$.
2) $f(n, m) > f(n, m - 1)$ if $m > 2n$.
3) $f(n, m) \geq f(n_1, m_1) + f(n - n_1, m - m_1)$ if $0 \leq n_1 \leq n$ and $0 \leq m_1 \leq m$.

Algorithm $C2SAT_6(F, C, W)$

Assume $d(F) \leq 6$.
1) if F contains no clauses, return(1,0). If F contains empty clause return (0,0)
2) if F is not connected, return (c, w) where $c = \prod_{i=0}^{j} c_i, w = \prod_{i=0}^{j} w_i$ and
$(c_i, w_i) = C2SAT(F_i, C, W)$ for connected components $F_0, \ldots, F_j$.
3) if multiplier reduction applies, apply it, removing the F_i with the smallest f value.
4) Pick a variable x of maximum degree, with the maximum $S(x)$. There are two sub cases
a) if $N(x)$ is connected to the rest of the graph using only two external vertices y and z, such that $d(y) \geq d(z)$, then branch on y.
b) else branch on x.

The values [1] of k_i, χ_i, a_i, b_i are in Table 1. Also provided are the worst case recurrences and the corresponding running times. $O(\text{poly}(n) \cdot 2^{\chi_i n})$ is the upper limit on the running time for a formula F with $m(F)/n(F) \leq k_i$.

Section	k_i, k_{i+1}	*Worst Case*	χ_i	b_i	a_i	*Running Time* [2]
0	0, 2		0	0	0	$O(\text{poly}(n))$
1	2, 2.4	$\tau(4a_1 + 12b_1, 4a_1 + 12b_1)$	0	1/4	-1/2	$O(1.071773^n)$
2	2.4, 2+2/3	$\tau(2a_2 + 8b_2, 4a_2 + 14b_2)$	0.1	0.188329	-0.351991	$O(1.109739^n)$
3	2+2/3, 3	$\tau(a_3 + 6b_3, 4a_3 + 16b_3)$	0.150221	0.155676	-0.264914	$O(1.150382^n)$
4	3, 3.2	$\tau(2a_5 + 10b_5, 5a_5 + 18b_5)$	0.202113	0.090158	-0.068363	$O(1.164850^n)$
5	3.2, 3.5	$\tau(a_5 + 8b_5, 5a_5 + 20b_5)$	0.220145	0.089883	-0.067481	$O(1.186825^n)$
6	3.5, 3.75	$\tau(a_6 + 8b_6, 5a_6 + 22b_6)$	0.247107	0.075935	-0.018665	$O(1.202545^n)$
7	3.75, 4	$\tau(a_7 + 8b_7, 5a_7 + 24b_7)$	0.266091	0.065244	0.021424	$O(1.216218^n)$
8	4, 4+4/29	$\tau(a_8 + 10b_8, 6a_8 + 26b_8)$	0.282402	0.036544	0.136223	$O(1.220475^n)$
9	4+4/29, 4+4/9	$\tau(a_9 + 10b_9, 6a_9 + 28b_9)$	0.287442	0.032416	0.153328	$O(1.228908^n)$
10	4+4/9, 4+4/7	$\tau(a_{10} + 10b_{10}, 6a_{10} + 30b_{10})$	0.297377	0.028781	0.169460	$O(1.232025^n)$
11	4+4/7, 4.8	$\tau(a_{11} + 10b_{11}, 6a_{11} + 32b_{11})$	0.301031	0.025915	0.182562	$O(1.237093^n)$
12	4.8, 5	$\tau(a_{12} + 10b_{12}, 6a_{12} + 34b_{12})$	0.306955	0.023227	0.195464	$O(1.241083^n)$
13	5, 5+5/47	$\tau(a_{13} + 12b_{13}, 7a_{13} + 36b_{13})$	0.311600	0.006557	0.278814	$O(1.241683^n)$
14	5+5/47, 5+1/3	$\tau(a_{14} + 12b_{14}, 7a_{14} + 38b_{14})$	0.312297	0.006069	0.281303	$O(1.242869^n)$
15	5+1/3, 5.5	$\tau(a_{15} + 12b_{15}, 7a_{15} + 40b_{15})$	0.313675	0.005561	0.283724	$O(1.243675^n)$
16	5.5, 5+5/8	$\tau(a_{16} + 12b_{16}, 7a_{16} + 42b_{16})$	0.314610	0.005177	0.286132	$O(1.244605^n)$
17	5+5/8, 5+5/6	$\tau(a_{17} + 12b_{17}, 7a_{17} + 44b_{17})$	0.315688	0.004669	0.289421	$O(1.245444^n)$
18	5+5/6, 6	$\tau(a_{18} + 12b_{18}, 7a_{18} + 46b_{18})$	0.316661	0.004336	0.291363	$O(1.246069^n)$

Table 1. Parameter Table

3.1 Worst Case Branching

In this subsection we discuss about the worst case branching situation for $C2SAT_6$. The following lemma says that if we start with formula F with average density m/n in *Section* i and after branching the *maximally reduced* F_1 has its average density m_1/n_1 lying in different *Section* j it is only better. Therefore, the worst case occurs only when both m/n and m_1/n_1 are in the same *Section*.

[1] Numbers were generated using Mathematica
[2] We hide a poly(n) factor throughout

Lemma 1. *Let $f(n,m)$ and a_i, b_i, m_1, n_1 be defined as above. Then,*

$$\triangle f(n,m) = f(n,m) - f(n_1,m_1) \geq \triangle f_i(n,m) = f_i(n,m) - f_i(n_1,m_1) \text{ if } k_i < m/n \leq k_{i+1}$$

Proof. The equality holds if $k_1 < m_1/n_1 \leq k_{i+1}$. The proof for $m_1/n_1 \leq k_i$ is omitted and can be found in [5]. Now assume that $m_1/n_1 > k_{i+1}$. By inductive arguments it is enough to focus on the transition of a single barrier, i.e., $m/n \geq k_i$ belongs in *Section* i, where as $m_1/n_1 \geq k_{i+1}$ belongs to *Section* $i+1$. 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 decreasing with i. □

Worst case branching is when branching is *maximally unbalanced*. If $d(F) = 2$ in Case 4a then F is a cycle and we are done after one branching. Thus assume that we are in Case 4a with $d(F) \geq 3$. In both branches, we eliminate at least y by assignment and $N(x)$ by Reduction. 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 Case 4b of the algorithm is the more interesting case and needs 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 $N_G(x)$, $\triangle m_1 = 2 \cdot (d(x) + \#$degree 2 nodes in $N_G(x))$. 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 2). 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 properties of f that Case 4b is harder than Case 4a. In $m \leq 3.75n$ one can easily verify with the given a_i and b_i that 4a always has a worst case branching less than 2. Also long chains of degree 2 nodes don't hurt because $|2b_i| \geq |a_i|$ when $m \leq 3.75n$ and long chains are beneficial if $m > 3.75n$. So from now on we will only be focusing on Case 4b.

Lemma 2. *Let $x \in F$ be the variable we branch on, in Case 4b of the Algorithm $C2SAT_6$. In a worst case branching, i.e., the branching is maximally unbalanced, we reduce $m(x)$ by at least $2\lceil \frac{S(x)+3}{2} \rceil$ in at least one of the branchings.*

Proof. Omitted in this extended abstract. The proof idea is by observing to achieve *maximally unbalanced* branching we delete all the edges incident on $N_G(x)$ in one branching step. The proof follows by showing that change in $m(x)$ is at least $S(x) + 3$ when $S(x)$ is odd, and $S(x) + 4$ when $S(x)$ is even. □

We also use the following lemma from [5] that makes a connection between the values of $m(F)/n(F)$ and worst-case branchings.

Lemma 3. ***(Dahllöf et al. [5])*** *Let F be a non-empty formula such that $m(F)/n(F) = k$, and define $\alpha(x)$ and $\beta(x)$ such that*

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

$$\beta(x) = 1 + \sum_{\{y \in N(x) \,|\, d(y) < k\}} \frac{1}{d(y)}$$

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

3.2 Proofs

We can proceed to prove upper bounds on the performance of $C2SAT_6$. The proof will be divided according to the values of $m(F)/n(F)$. In all lemmas, except for the case when $m \leq 3n$, we use Lemma 3 to generate the k_i values. We branch on some variable $x \in F$, eventually resulting in two *maximally reduced* formulas F_1 and F_2. It is shown that in each *Section i* the worst case branching number is 2. It is possible to end up with more than two *maximally reduced* formulas, then the above applies to all connected components F_i and by Property 3 the total work is smaller. The next lemma talks about the simple case where $m \leq 2n$. The proof is omitted but is a straight forward consequence of applying Reduction [5].

Lemma 4. ***(Dahllöf et al. [5])****For a* maximally reduced *formula F with $m \leq 2n$, $C2SAT_6(F, C, W)$ runs in time* $\mathrm{O}(\mathrm{poly}(n))$.

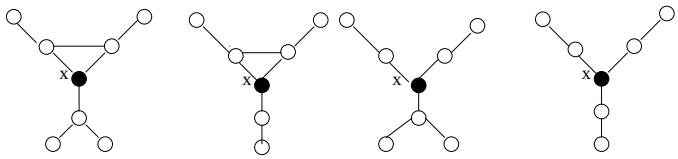

Fig. 1. Sample cases with $d(F) = 3$

Lemma 5. *Let F be a* maximally reduced *formula with $m \leq 3n$ and $d(F) = 3$, then $C2SAT_6(F, C, W)$ runs in time* $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_4 n})$.

Proof. If $n_3(F) = 0$, then we are in the 2-regular case and only polynomial time is required using Propagation and Reduction. So the interesting case is when $d(F) = 3$. We divide this case into worst cases depending on the number of degree 3 nodes (0,1,2 or 3) adjacent to x (Figure 1). 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 12/5 (3*2/5+2*3/5) for k_2. This lemma uses $f_1(n, m)$ to $f_4(n, m)$ as measures.
Section 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 $\triangle m$ in m is

$$\sum_{v \in V - V_1} d(v) + |\{ \text{ clauses } C' \text{ in } F \,|\, C' \text{ involves variables from both } V - V_1 \text{ and } V_1 \}|$$

Since $d(v) = 3$ and there are no singleton variables in F, the first term is at least $2\triangle n + 1$, and since Reduction 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. Also, $\chi_2 = \chi_1 + (12/5 - 2)b_1 = 0.1$.
Section 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.

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 branching number of $\tau(3a_2 + 10b_2, 4a_2 + 14b_2) < 2$.
2. $S(x) = 11$. In this case x has exactly two degree 3 nodes as its neighbor. The worst case branching is when the branching is *maximally unbalanced*, with branching number of $\tau(2a_2 + 8b_2, 4a_2 + 14b_2) = 2$. As given in Table 1 we have $b_2 \approx 0.188329$ and $a_2 \approx -0.351991$ and $\chi_3 = \chi_2 + (8/3 - 12/5)b_2 \approx 0.150221$.

Section 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 Table 1 we have $b_3 \approx 0.155676$ and $a_3 \approx -0.264914$ and $\chi_4 = \chi_3 + (3 - 8/3)b_3 \approx 0.202113$. As we see the worst case branching number is 2 and the worst case running time for $C2SAT_6$ with maximum degree 3 and $m \leq 3n$ is $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_4 n}) \approx \mathrm{O}(\mathrm{poly}(n) \cdot 1.150382^n)$. □

Lemma 6. *Let F be a* maximally reduced *formula with $m \leq 3n$ and $4 \leq d(F) \leq 6$, then $C2SAT_6(F, C, W)$ runs in time* $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_4 n})$.

Proof. First we consider the case $d(F) = 4$. There are 5 types of worst cases (Figure 2) 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 numbers are generated using bounds as given above.

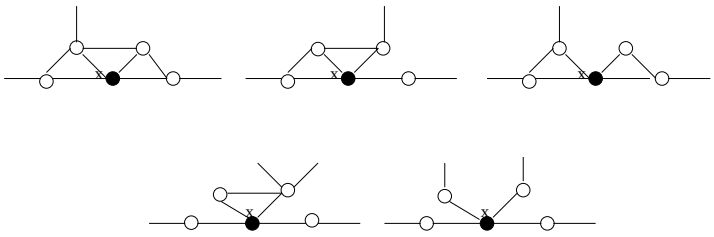

Fig. 2. Sample worst cases with $d(F) = 4$

1. $\triangle n_1 = 5, \triangle m_1 = 16, \triangle n_2 = 5, \triangle m_2 \geq 16$.
2. $\triangle n_1 = 4, \triangle m_1 = 14, \triangle n_2 = 5, \triangle m_2 \geq 16$.
3. $\triangle n_1 = 3, \triangle m_1 = 12, \triangle n_2 = 5, \triangle m_2 \geq 18$.
4. $\triangle n_1 = 2, \triangle m_1 = 10, \triangle n_2 = 5, \triangle m_2 \geq 18$.
5. $\triangle n_1 = 1, \triangle m_1 = 08, \triangle n_2 = 5, \triangle m_2 \geq 20$.

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 nodes(0 to 5) adjacent to the selected vertex x.

1. $\triangle n_1 = 6, \triangle m_1 = 20, \triangle n_2 = 6, \triangle m_2 \geq 18$.
2. $\triangle n_1 = 5, \triangle m_1 = 18, \triangle n_2 = 6, \triangle m_2 \geq 20$.
3. $\triangle n_1 = 4, \triangle m_1 = 16, \triangle n_2 = 6, \triangle m_2 \geq 20$.
4. $\triangle n_1 = 3, \triangle m_1 = 14, \triangle n_2 = 6, \triangle m_2 \geq 22$.
5. $\triangle n_1 = 2, \triangle m_1 = 12, \triangle n_2 = 6, \triangle m_2 \geq 22$.
6. $\triangle n_1 = 1, \triangle m_1 = 10, \triangle n_2 = 6, \triangle m_2 \geq 24$.

Again, for all these cases [3] 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:

[3] One can eliminate a few cases by noting that $|2b_i| \geq |a_i|$ in this range

1. $\triangle n_1 = 7, \triangle m_1 = 24, \triangle n_2 = 7, \triangle m_2 \geq 22$.
2. $\triangle n_1 = 6, \triangle m_1 = 22, \triangle n_2 = 7, \triangle m_2 \geq 22$.
3. $\triangle n_1 = 5, \triangle m_1 = 20, \triangle n_2 = 7, \triangle m_2 \geq 24$.
4. $\triangle n_1 = 4, \triangle m_1 = 18, \triangle n_2 = 7, \triangle m_2 \geq 24$.
5. $\triangle n_1 = 3, \triangle m_1 = 16, \triangle n_2 = 7, \triangle m_2 \geq 26$.
6. $\triangle n_1 = 2, \triangle m_1 = 14, \triangle n_2 = 7, \triangle m_2 \geq 26$.
7. $\triangle n_1 = 1, \triangle m_1 = 12, \triangle n_2 = 7, \triangle m_2 \geq 28$.

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 $C2SAT_6$ with $m \leq 3n$ is is $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_4 n}) \approx \mathrm{O}(\mathrm{poly}(n) \cdot 1.150382^n)$. □

Lemma 7. *For a* maximally reduced *formula F with $3n < m \leq 4n$, $C2SAT_6(F, C, W)$ runs in time* $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_8 n})$.

Proof. In this case we have $d(F) = 4$, 5 or 6. We use the measures of $f_4(m, n)$ to $f_7(m, n)$. 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 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 x with $\alpha(x)/\beta(x) > 3$. The minimum value of $S(x)$ 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 $\alpha(x)/\beta(x) > 3$.
Section 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 recursion is $\tau(2a_2 + 10b_2, 5a_2 + 18b_2)$, solving which we get $b_4 \approx 0.090158$ and $a_4 \approx -0.068363$ and $\chi_5 = \chi_4 + (3.2 - 3)b_4 \approx 0.220145$. Since cases with $S(x) = 15$ can appear only till $m(F)/n(F) \leq 3.2$ (from Lemma 3), so the next *Section* starts with $m(F)/n(F) > 3.2$.
Other *Sections* 5 to 7 can be shown similarly, and also have been handled by [5]. The worst case running time of $C2SAT_6$ with $m \leq 4n$ is $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_8 n}) \approx \mathrm{O}(\mathrm{poly}(n) \cdot 1.216218^n)$.
□

Lemma 8. *For a* maximally reduced *formula F with $4n < m \leq 5n$, $C2SAT_6(F, C, W)$ runs in time* $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_{13} n})$.

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 *Sections* 8 to 12 as in previous lemmas (also handled by [5]). The worst case running time of $C2SAT_6$ with $m \leq 5n$ is $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_{13} n}) \approx \mathrm{O}(\mathrm{poly}(n) \cdot 1.241083^n)$. □

Lemma 9. *For a* maximally reduced *formula F with $5n < m \leq 6n$, $C2SAT_6(F, C, W)$ runs in time* $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_{19} n})$.

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 *Sections* 13 to 18 as in previous lemmas. This results in $\chi_{19} \approx 0.317384$. The worst case running time of $C2SAT_6$ with $m \leq 6n$ is $\mathrm{O}(\mathrm{poly}(n) \cdot 2^{\chi_{19} n}) \approx \mathrm{O}(\mathrm{poly}(n) \cdot 1.246069^n)$. □

Putting together Lemmas 4-9 we get that $C2SAT_6$ has a worst case running time of $\mathrm{O}(1.246069^n)$.

Algorithm $C2SAT(F, C, W)$

1) if F contains no clauses, return(1,0). If F contains empty clause return (0,0)
2) if F is not connected, return (c, w) where $c = \prod_{i=0}^{j} c_i, w = \prod_{i=0}^{j} w_i$ and $(c_i, w_i) = C2SAT(F_i, C, W)$ for connected components $F_0, \ldots, F_j$.
3) if there exists a non-monotone variable x with $d(x) \geq 6$, then branch on x.
4) if $d(F) \leq 6$, then return $C2SAT_6(F, C, W)$.
5) pick a variable x of maximum degree and branch on it.

Theorem 1. *$C2SAT(F, C, W)$ runs in time* $\mathrm{O}(\mathrm{poly}(n) \cdot 1.246069^n)$.

Proof. Algorithm $C2SAT(F, C, W)$ handles various cases. If we are in Case 3, the worst case is $T(n) = T(n-2) + T(n-6)$, so we have the solution $\mathrm{O}(\tau(2,6)^n) \approx \mathrm{O}(1.21061^n)$. If we apply the algorithm $C2SAT_6$ we have a running time of $\mathrm{O}(\mathrm{poly}(n) \cdot 1.246069^n)$. In case we branch on the variable of a vertex of degree greater than 6, we have a trivial worst case of $T(n) = T(n-1) + T(n-8)$ with solutions in $\mathrm{O}(\tau(1,8)^n) \approx \mathrm{O}(1.23205^n)$. So we have a worst case upper bound of $\mathrm{O}(\mathrm{poly}(n) \cdot 1.246069^n)$. □

4 Applications

Here we summarize some other interesting problems whose running times from [5, 1] have improved as a direct consequence of our result. We present only the definitions and upper bounds but not the reductions.

1. *#Maximum Weighted Independent Set* can be solved in $\mathrm{O}(1.246069^n)$.
   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 Maximum Weighted Independent Sets.
2. *#Exact Cover* can be solved in $\mathrm{O}(1.246069^n)$.
   Instance: Collection C of subsets of a finite set S.
   Solution: A subset $C' \subseteq C$ s.t. every element in S belongs to at exactly one member of C'. Count the number of such covers.
3. *#Exact Hitting Set* can be solved in $\mathrm{O}(1.246069^n)$.
   Instance: Collection $C = \{c_1, \ldots, c_n\}$ of subsets of a finite set S, s.t. $\bigcup c_i = S$.
   Solution: A minimum subset $S' \subseteq S$ s.t. S' contains exactly one element from each subset c_i. Count the number of such sets.
4. *#Weighted Set Packing* can be solved in $\mathrm{O}(1.246069^n)$.
   Instance: Collection $C = \{c_1, \ldots, c_n\}$ of subsets of a finite set S with weights on each c_i.
   Solution: A collection of disjoint sets $C' \subseteq C$ of maximum weight. Count the number of such packings.

To the best of our knowledge no faster algorithm is known for finding a truth assignment for $2\mathrm{SAT}_w$ or for solving the above Problems 1-4. Also, #Perfect Matchings and #Matchings for general graphs can be solved in $\mathrm{O}(1.246069^{|E|})$ where $|E|$ is the number of edges. To improve the running time one could think of utilizing dynamic programming. Other ways could be to find a way to characterize the *Section* boundaries better and to perform a more careful analysis of the neighborhood.

5 Counting Solutions in Constraint Satisfaction Problems

Another problem which has a better running time because of our result is counting the number of Weighted Binary CSP (Constraint Satisfaction Problem) solutions. A $(d, 2)$-CSP instance is a triple (V, D, C), where V is a set of variables, D a finite domain of values with $|D| = d$, and C a set of constraints $\{c_1, \ldots, c_q\}$. Each constraint c_i is a triple xRy, where $x, y \in V$ and $R \in D^2$. A solution is a function $f : V \to D$, s.t., $(f(x), f(y)) \in R$ for each constraint xRy. The counting problem is to determine the number of solutions.

We use the same approach as in [1]. The idea is to create a weighted $\#(d, 2)$ CSP instance and to partition the domains of the variable into subdomains of sizes between 2 and 5 elements. The algorithm for the $\#2\mathrm{SAT}_w$ is used for solving these smaller instances and results are recombined. The decrease in running times from [1] is because of the faster running times for $\#2\mathrm{SAT}_w$ and better partitioning of the domains. The following theorem from [1] demonstrates how partitions can be used for solving $\#(d, 2)$-CSP. In the following analysis we use α^n to indicate the running time for $\#2\mathrm{SAT}_w$, i.e., $\alpha = 1.246069$.

Theorem 2. *(**Angelsmark et al. [1]**) Let A be an algorithm for $\#(p, 2)$-CSP running in $\mathrm{O}\left(\prod_{i=1}^{p} \alpha_i^{n_i}\right)$ 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 time $\mathrm{O}\left(\left(\sum_{i=1}^{p} \sigma(P, i)\alpha_i\right)^n\right)$, where $\sigma(P, i)$ denotes the number of parts of size i in P.*

A $\#(d, 2)$-CSP instance is solved by transforming it into a weighted $\#2\mathrm{SAT}_w$ instance. The transformation from [1] is reproduced for completeness:

1. if x is single valued we can remove it by assigning 1 to x and by propagating.
2. if x is two-valued, we introduce a propositional variable x_i 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 $\#2\mathrm{SAT}_w$ is $\mathrm{O}(1^{k_1} \cdot \alpha^{k_2} \cdot \alpha^{3k_3} \ldots)$, 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.

Given a partitioning P of a domain D containing d elements, it follows by using Theorem 2 and the above construction that $\#(d,2)$-CSP can be solved in $\mathrm{O}(T(P)^n)$ where

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

We let the multiset $[[P_1],\ldots,[P_k]]$ represent the partition P. The following theorem defines the optimal partitioning for $\alpha = 1.246069$. We use the following fact throughout the proof: if $T([a_1,\ldots,a_n]) < T([b_1,\ldots,b_m])$, then $T([a_1,\ldots,a_n,c_1,\ldots,c_k]) < T([b_1,\ldots,b_m,c_1,\ldots,c_k])$ for all choices of $c_1,\ldots,c_k$.

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:*
1) if $d = 5k$, $P = [5,5,\ldots,5]$
2) if $d = 5k+1$, $d \geq 16$, $P = [4,4,4,4,5,5,\ldots,5]$
3) if $d = 5k+2$, $d \geq 12$, $P = [4,4,4,5,5,\ldots,5]$
4) if $d = 5k+3$, $P = [4,4,5,5,\ldots,5]$
5) if $d = 5k+4$, $P = [4,5,5,\ldots,5]$
6) 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 as in [1] 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 $1 < d < 3$ 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:
1) P^* contains only parts of size a, $a \in \{2,4,5\}$.
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 $d > 3$.

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.

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 $d \notin \{6,7,11\}$.

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 P is optimal. □

Using the above theorem we can compute the running times [4]. For large domains, the term $\frac{d}{5} \cdot \alpha^5$ dominates the time complexity. Consequently, all the bounds approach to

[4] Not included are domain sizes 2,3,4,5,6,7,11

$O(\frac{d}{5} \cdot \alpha^5) \approx O(0.6009^n)$ improving the previous best bound of $O(0.6224^n)$. The following table summarizes our time complexity.

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

In the Table 2 we compare the improvement in running times for sample domain sizes.

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
15	9.3619^n	9.0122^n
25	15.5740^n	15.0204^n

Table 2. Time Complexity Comparison for $\#(d, 2)$-CSPs

6 Counting #3-Colorings

We now present an algorithm for counting the number of 3-colorings of a graph. Let G be a graph with $V(G)$ as set of vertices and $E(G)$ as set of edges. A k-coloring of a graph G is a function $C : V(G) \rightarrow \{1, \ldots, k\}$ such that for all $v, w \in V(G)$, if $C(v) = C(w)$ then $(v, w) \notin E(G)$. The #k-COL problem is to determine the number of such k-colorings for G. We first present a faster algorithm #3-COL and then show how it results in faster #k-COL algorithm when combined with results from [1].

The 3-COL problem is a special (3,2)-CSP problem, hence the algorithm from the previous section trivially implies that we have a running time of $O(1.9348^n)$, but as will be seen, this can be improved down to $O(1.7702^n)$. The previous best algorithm for this problem has an upper bound of $O(1.7879^n)$. We choose the colors from the set $\{R, G, B\}$. We denote by $I(G)$ a maximum independent set in G. As in [1] we define an $R\{G/B\}$ assignment as a total function $C : V(G) \rightarrow \{R, GB\}$. Once we have an $R\{G/B\}$ assignment it can be tested in polynomial time whether it is possible to reassign colors G or B, to all vertices v with the initial color $C(v) = GB$. The number of such reassignments is 2^c, where c is the number of connected components in the subgraph induced by such vertices. The algorithm (C3COL) for #3-COL begins by identifying a maximum independent set $I(G)$. If $|I(G)| \leq c \cdot |V(G)|$, we search for an uncolored vertex x with at least one of its neighbors uncolored. We perform a two way branching on x. In one branch we assign color R to x and assign GB to its uncolored neighbors. In the other branch we assign color GB to x. If $|I(G)| > c \cdot |V(G)|$ we enumerate all the 3-colorings of the induced subgraph G' on $V(G) - I(G)$. To do so we find a maximal independent set $(I(G'))$ for G'. We search through all possible 3-colorings of $I(G')$ and all permitted 2-colorings of $V(G') - I(G')$. The correctness of the algorithm C3COL can be shown similarly to [1].

Function C3COL(G)

1) if $|I(G)| \leq c \cdot |V(G)|$ then
   a) if all $v \in V(G)$ are $R\{G/B\}$-colored then return $Count(G)$.
   b) else if there exists an uncolored vertex x with U as the set of its uncolored neighbors, and, $U \neq \emptyset$, then return (C3COL($G[x = R, U = GB]$) + C3COL($G[x = GB]$)).
   c) else if there is an uncolored vertex x return (C3COL($G[x = R]$) + C3COL($G[x = GB]$)).
2) else
   a) find the induced graph G' of $V(G) - I(G)$.
   b) find $I(G')$, then cycle through all the possible 3-colorings of $I(G')$ and permitted 2-colorings of $V(G') - I(G')$.
   d) for every 3-coloring C of G' do $c = c + \prod_{v \in I(G)}(3 - |\{C(w)|w \in N_G(x) - \{x\}\}|)$.
   e) return c.

Theorem 4. *Algorithm C3COL runs in time* $\mathrm{O}(\mathrm{poly}(n) \cdot 1.7702^n)$.

Proof. The best known algorithm for finding a maximum independent set is by Robson [20] running in time $\mathrm{O}(1.2025^n)$. The analysis of C3COL has two parts.
1) if $|I(G)| \leq c \cdot |V(G)|$:
The worst case recursive equation at line 1b) has the Fibonacci form $T(n) \leq T(n-1) + T(n-2)$ and $T(n) \in \mathrm{O}(\phi^n)$, where ϕ is the golden ratio [5]. If line 1c) of the algorithm is reached, the uncolored vertices form an independent set I'. Therefore, the total running time of this part is $\mathrm{O}(2^{|I'|} \cdot \phi^{|V(G)|-|I'|})$. The worst case running time is when $|I'| = c \cdot |V(G)|$. In this case algorithm runs in time $\mathrm{O}(2^{cn} \cdot \phi^{(n-cn)})$.
2) if $|I(G)| > c \cdot |V(G)|$:
Since $I(G)$ is a maximum independent set, we know that $|I(G')| \leq |I(G)|$. We can enumerate all possible 3-colorings of G' in time $\mathrm{O}(3^{|I(G')|} \cdot 2^{|V(G)|-|I(G)|-|I(G')|})$. If $|I(G)| \geq \frac{1}{2}|V(G)|$ we have a worst case when $|I(G')| = |V(G)| - |I(G)| = \frac{1}{2}|V(G)|$ resulting in running time of $\mathrm{O}(3^{n/2})$. The more interesting case is when, $c' \cdot |V(G)| = |I(G')| \leq |I(G)| < \frac{1}{2} \cdot |V(G)|$. The worst case running time in this interval is when $|I(G')| = |I(G)| = c \cdot |V(G)|$. In this case algorithm runs in time $\mathrm{O}(3^{cn} \cdot 2^{(n-2cn)})$.
Finally, we obtain the values of $c = c' = 0.4242$ by solving:

$$2^c \cdot \phi^{1-c} = 3^c \cdot 2^{1-2c}$$

So the algorithm C3COL runs in time $\mathrm{O}(\mathrm{poly}(n) \cdot 1.7702^n)$. □

It might be possible to further reduce the running time for some special graphs, e.g., for bipartite graphs we can reduce the upper bound to $\mathrm{O}(3^{n/2})$. The improvement in the running time for #3-COL is particularly important because it automatically improves the running time of the best known #k-COL algorithm of Angelsmark *et al.* [1]. The following theorem summarizes the improvement.

Theorem 5. ***(Angelsmark et al. [1])*** *There is an algorithm for solving the #k-COL problem in time* $O((c_k)^n)$*, where, for some* $i \in \mathbb{N}$

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

with β *as the running time of #3-COL.*

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

References

1. O. Angelsmark and P. Jonsson. Improved algorithms for counting solutions in constraint satisfaction problems. In *ICCP: International Conference on Constraint Programming (CP), LNCS*, 2003.
2. O. Angelsmark, P. Jonsson, S. Linusson, and J. Thapper. Determining the number of solutions to binary CSP instances. *Lecture Notes in Computer Science*, 2470:327–340, 2002.
3. V. Dahllöf and P. Jonsson. An algorithm for counting maximum weighted independent sets and its applications. *Proceedings of the thirteen annual ACM-SIAM Symposium On Discrete Algorithms*, pages 292–298, 2002.
4. V. Dahllöf, P. Jonsson, and M. Wahlström. Counting satisfying assignments in 2-SAT and 3-SAT. In *COCOON: Annual International Conference on Computing and Combinatorics*, 2002.
5. V. Dahllöf, P. Jonsson, and M. Wahlström. Counting models for 2SAT and 3SAT formulae. *Theoretical Computer Science, http://www.sciencedirect.com/science/article/B6V1G-4DTBKWK-1/2/dba6357d72b8f629ec960f74e4f396dd*, 332(1-3):265–291, 2005.
6. E. Danstin, E. A. Hirsch, S. Ivanov, and M. Vserminov. Algorithms for sat and upper bounds on their complexity. *Electronic Colloquium on Computational Complexity*, 12(8), 2001.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, July 1962.
8. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of Association Computer Machinery*, 7:201–215, 1960.
9. O. Dubois. Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science*, 81(1):49–64, April 1991.
10. M. Dyer and C. Greenhill. The complexity of counting graph homomorphisms. *RSA: Random Structures and Algorithms*, 17:260–289, 2000.
11. M. Dyer and C. Greenhill. Corrigendum: The complexity of counting graph homomorphisms. *RSA: Random Structures and Algorithms*, 25:346–352, 2004.
12. D. Eppstein. Improved algorithms for 3-Coloring, 3-Edge-Coloring, and constraint satisfaction. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01)*, pages 329–337, New York, January 7–9 2001. ACM Press.
13. T. Feder and R. Motwani. Worst-case time bounds for coloring and satisfiability problems. *J. Algorithms*, 45(2):192–201, 2002.
14. U. Feige, S. Goldwasser, L. Lóvasz, S. Safra, and M. Szegedy. Approximating the clique is almost NP-complete. In *Proc. 32nd IEEE Symp. on Foundations of Comp. Science*, pages 34–39. IEEE, 1991.
15. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.*, 182(1):105–142, 1999.
16. P. G. Jeavons, D. A. Cohen, and J. K. Pearson. Constraints and universal algebra. *Annals of Mathematics and Artificial Intelligence*, 24:51–67, 1998.
17. D. L. Kozen. *The Design and Analysis of Algorithms*. Springer, Berlin, 1992.
18. O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223:1–72, 1999.
19. M. L. Littman, T. Pitassi, and R. Impagliazzo. On the complexity of counting satisfying assignments. In *The Working notes of LICS 2001 Workshop on Satisfiability*, 2001.
20. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical report, LaBRI, Université Bordeaux, 2001.
21. D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2):273–302, 1996.
22. S. P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, April 2002.
23. L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, April 1979.
24. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, August 1979.

25. E. Vigoda. Improved bounds for sampling colorings. *Journal of Mathematical Physics*, 41(3):1555–1569, March 2000.
26. W. Zhang. Number of models and satisfiability of sets of clauses. *Theoretical Computer Science*, 155(1):277–288, February 1996.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'