New Upper Bounds for MAX-2-SAT and MAX-2-CSP w.r.t. the Average Variable Degree[⋆]

Alexander Golovnev

St. Petersburg University of the Russian Academy of Sciences, St. Petersburg, Russia
alex.golovnev@gmail.com

Abstract. MAX-2-SAT and MAX-2-CSP are important NP-hard optimization problems generalizing many graph problems. Despite many efforts, the only known algorithm (due to Williams) solving them in less than 2^n steps uses exponential space. Scott and Sorkin give an algorithm with $2^{n(1-\frac{2}{d+1})}$ time and polynomial space for these problems, where d is the average variable degree. We improve this bound to $O^*(2^{n(1-\frac{10/3}{d+1})})$ for MAX-2-SAT and $O^*(2^{n(1-\frac{3}{d+1})})$ for MAX-2-CSP. We also prove stronger upper bounds for d bounded from below. E.g., for $d \geq 10$ the bounds improve to $O^*(2^{n(1-\frac{3.469}{d+1})})$ and $O^*(2^{n(1-\frac{3.221}{d+1})})$, respectively. As a byproduct we get a simple proof of an $O^*(2^{\frac{m}{5.263}})$ upper bound for MAX-2-CSP, where m is the number of constraints. This matches the best known upper bound w.r.t. m due to Gaspers and Sorkin.

Keywords: algorithm, satisfiability, maximum satisfiability, constraint satisfaction, maximum constraint satisfaction.

1 Introduction

1.1 Problem Statement

The maximum satisfiability problem (MAX-SAT) is, given a boolean formula in conjunctive normal form (CNF), to find a maximum number of simultaneously satisfiable clauses of this formula. MAX-2-SAT is restricted MAX-SAT, where each clause contains at most two literals. MAX-SAT and MAX-2-SAT are NP-hard problems. Moreover, it is still not known whether MAX-2-SAT can be solved in less than $O^*(2^n)$[1] with polynomial memory.

MAX-2-SAT is a special case of the maximum 2-constraint satisfaction problem (MAX-2-CSP). In MAX-2-CSP problem one is given a graph $G = (V, E)$ along with sets of functions $S_v : \{0,1\} \to \mathbb{Z}$ for each vertex v and $S_e : \{0,1\}^2 \to \mathbb{Z}$ for each edge e. The goal is to find an assignment $\phi : V \to \{0,1\}$ maximizing the sum

$$\sum_{e=(v_1,v_2)\in E} S_e(\phi(v_1), \phi(v_2)) + \sum_{v \in V} S_v(\phi(v)). \tag{1}$$

⋆ Research is partially supported by Yandex, Parallels and JetBrains.

1 As usual, $O^*(\cdot)$ suppresses polynomial factors

It is easy to see that MAX-2-SAT corresponds to the case when all functions from S_e are disjunctions (of variables and their negations). MAX-2-SAT and MAX-2-CSP are important NP-hard optimization problems generalizing many graph problems.

1.2 The Main Definitions

Let F be an instance of MAX-2-SAT or MAX-2-CSP. By $n(F), m(F)$ we denote, respectively, the number of vertices (variables) and the number of edges (clauses) of the formula F. By the degree $deg(x)$ of a vertex x we mean the number of edges incident to x. We say variable y is the neighbor of variable x if there is an edge (x, y) in the graph (i.e. there is a 2-clause with these variables in F). By $\Delta(F)$ we denote the maximum vertex degree. $d(F) = 2m/n$ is the average vertex degree. We omit F if it is clear from the context. By the length $|F|$ of a formula F we mean its number of clauses.

Note that in case of MAX-2-CSP one can assume without loss of generality that the corresponding graph does not contain multiple edges (as any two parallel edges can be replaced by their "sum"). At the same time one cannot exclude multiple edges from a MAX-2-SAT graph by the same argument (e.g., the graph of a formula $(x \vee y)(\neg x \vee y)(y \vee z)$ has two edges between x and y).

By (n, Δ)-MAX-2-SAT and (n, Δ)-MAX-2-CSP we denote, respectively, MAX-2-SAT and MAX-2-CSP problems restricted to instances in which each variable appears in at most Δ 2-clauses. By $Opt(F)$ we denote the maximal value of (1) for F over all possible assignments (for MAX-2-SAT, this is the maximal number of simultaneously satisfiable clauses of the formula F).

Let F be an instance of MAX-2-SAT or MAX-2-CSP, l be a literal of F. By $F[l]$ we denote a formula resulting from F by replacing l by 1 and $\neg l$ by 0. Under this assignment, all 2-clauses containing l or $\neg l$ become 1-clauses.

1.3 Known Results

In this subsection, we review some known results for the considered problems. Williams [1] proved that MAX-2-CSP can be solved in time $O^*(2^{\frac{\omega n}{3}})$, where $\omega \approx 2.376$ is the matrix multiplication exponent. Williams' algorithm beats the 2^n barrier at the cost of requiring exponential space. It is a big challenge of the field to solve MAX-2-CSP in less than 2^n steps with only polynomial space. However the trivial 2^n upper bound was improved for several special cases of the considered problems. Dantsin and Wolpert [2] showed that MAX-SAT for formulas with constant clause density can be solved faster than in $O^*(2^n)$ time with exponential space. Kulikov and Kutzkov [3] developed an algorithm for MAX-SAT with polynomial space (and all the algorithms mentioned below use polynomial space) and running time c^n for formulas with constant clause density, where $c < 2$ is a constant.

Fürer and Kasiviswanathan [4] developed an algorithm for MAX-2-SAT with the running time $O^*(2^{n(1-\frac{1}{d-1})})$. Scott and Sorkin [5] improved this bound to $O^*(2^{n(1-\frac{2}{d+1})})$. For $(n,3)$-MAX-2-SAT, Kojevnikov and Kulikov [6] proved $O^*(2^{\frac{n}{6}})$ bound. This was later improved to $O^*(2^{\frac{n}{6.7}})$ by Kulikov and Kutzkov [3].

Concerning the number of clauses m, the best known upper bound $O^*(2^{\frac{m}{2.465}})$ for MAX-SAT was given by Chen and Kanj [7]. For MAX-2-SAT and MAX-2-CSP, Gaspers and Sorkin [8] proved $O^*(2^{\frac{m}{6.321}})$ and $O^*(2^{\frac{m}{5.263}})$ bounds, respectively.

MAX-CUT is a special case of MAX-2-CSP. Della Croce, Kaminski and Paschos [9] developed an algorithm for MAX-CUT with the running time $O^*(2^{n(1-\frac{2}{\Delta})})$.

1.4 New Upper Bounds

In this paper, we present an elementary algorithm solving MAX-2-SAT and MAX-2-CSP in time $O^*(2^{n(1-\frac{10/3}{d+1})})$ and $O^*(2^{n(1-\frac{3}{d+1})})$, respectively. We show also how to improve these bounds for d bounded from below. E.g., for $d \geq 5$ we get upper bounds $O^*(2^{n(1-\frac{3.40}{d+1})})$ and $O^*(2^{n(1-\frac{3.15}{d+1})})$ and for $d \geq 10$ we get $O^*(2^{n(1-\frac{3.469}{d+1})})$ and $O^*(2^{n(1-\frac{3.221}{d+1})})$. The key point of our algorithm is branching on a vertex of maximal degree that has at least one neighbor with smaller degree.

From these improved upper bounds w.r.t. the average degree d we can derive an upper bound $O^*(2^{\frac{m}{5.263}})$ w.r.t. the number of clauses for MAX-2-CSP. This bound matches the best known upper bound by Gaspers and Sorkin [8]. We also show that any improvement of upper bound for (n,Δ)-MAX-2-CSP for any $\Delta \leq 5$ would improve this record bound.

Since MAX-CUT is a special case of MAX-2-CSP, we also get an improved upper bound $O^*(2^{n(1-\frac{3}{d+1})})$ for MAX-CUT (again, the bound decreases when d increases).

1.5 Organization of the Paper

In Section 2, we construct a simple algorithm for MAX-2-SAT and MAX-2-CSP. The main idea of the algorithm is branching on a vertex of maximal degree. We prove $O^*(2^{n(1-\frac{10/3}{\Delta+1})})$ and $O^*(2^{n(1-\frac{3}{\Delta+1})})$ upper bounds for this algorithm. Section 3 generalizes upper bounds w.r.t. Δ to upper bounds w.r.t. d. Also, we apply this theorem to the algorithm from Section 2. In Section 4, we slightly change the algorithm and get stronger upper bounds for it.

2 A Simple Algorithm for MAX-2-SAT and MAX-2-CSP

In this section, we present a simple algorithm for (n,Δ)-MAX-2-SAT and (n,Δ)-MAX-2-CSP with upper bounds $O^*(2^{n(1-\frac{10/3}{\Delta+1})})$ and $O^*(2^{n(1-\frac{3}{\Delta+1})})$, respectively. To solve an instance of the maximal degree Δ, our algorithm branches on a variable of maximal degree until it gets an instance of maximal degree $\Delta - 1$.

2.1 Removing variables of degree 2

Lemma 1. *Let F be an instance of MAX-2-SAT or MAX-2-CSP containing a vertex u of degree at most 2. Then F can be transformed in polynomial time into a formula F' s.t.*

1. $deg_{F'}(u) = 0$,
2. *for all* v, $deg_{F'}(v) \leq deg_F(v)$,
3. $Opt(F)$ *can be computed from* $Opt(F')$ *in polynomial time.*

This lemma is proved for MAX-2-SAT in [6, Lemma 3.1], and for MAX-2-CSP in [8, Section 5.9]. It allows us to assume that a simplified formula contains variables of degree at least 3 only.

2.2 An Algorithm

The algorithm branches on a vertex of maximal degree Δ until it gets a graph of maximal degree 3. It then calls a known algorithm for $(n,3)$-MAX-2-SAT or $(n,3)$-MAX-2-CSP, respectively.

Denote by n_i the number of vertices of degree i for $i \in \{3,\ldots,\Delta\}$. Consider the problem (n,Δ)-MAX-2-SAT ((n,Δ)-MAX-2-CSP). We use the following formula complexity measure:

$$\mu = \alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta,$$

where α_i denotes the weight of a variable of degree i. The values of α_i's will be determined later. We would like to find α_i's such that for any formula F the algorithm has the running time $poly(|F|) \cdot 2^{\mu(F)}$.

Assume that an algorithm A solves $(n,\Delta-1)$-MAX-2-SAT ($(n,\Delta-1)$-MAX-2-CSP) in time $2^{\alpha_3 n_3+\ldots+\alpha_{\Delta-1}n_{\Delta-1}}$. Consider the following algorithm for (n,Δ)-MAX-2-SAT ((n,Δ)-MAX-2-CSP).

```
MetaAlg
Parameter: Algorithm $A$ for $(n,\Delta-1)$-MAX-2-SAT ($(n,\Delta-1)$-MAX-2-CSP).
Input: $F$ – instance of MAX-2-SAT or MAX-2-CSP.
Output: $Opt(F)$.
Method.
 1. Remove all vertices of degree $< 3$ (using Lemma 1).
 2. If $F$ does not contain 2-clauses, then return the result.
 3. If the maximal vertex degree of $F$ is less than $\Delta$, then return $A(F)$.
 4. Choose a vertex $x$ of maximal degree $\Delta$.
 5. Return $\max(\text{MetaAlg}(A, F[x]), \text{MetaAlg}(A, F[\neg x]))$.
```

Lemma 2. *Let $\Delta > 3$, $\alpha_i < 1$, for all i. If*

$$\delta = \min(\alpha_\Delta - \alpha_{\Delta-1}, \alpha_{\Delta-1} - \alpha_{\Delta-2}, \ldots, \alpha_4 - \alpha_3, \alpha_3) \geq \frac{1-\alpha_\Delta}{\Delta}, \qquad (2)$$

then the running time of the algorithm MetaAlg *for (n,Δ)-MAX-2-SAT ((n,Δ)-MAX-2-CSP) is* $2^{\alpha_3 n_3+\ldots+\alpha_\Delta n_\Delta}$.

Proof. Denote by $T(n_3, \ldots, n_\Delta)$ the running time of the algorithm on a formula that has n_i vertices of degree i, for all $3 \le i \le \Delta$. If there are no vertices of degree Δ (i.e., $n_\Delta = 0$), then MetaAlg just calls A. Then, clearly,

$$T(n_3, \ldots, n_\Delta) \le 2^{\alpha_3 n_3 + \ldots + \alpha_{\Delta-1} n_{\Delta-1}} = 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta}.$$

Assume now that there exists a vertex x of degree Δ. Then MetaAlg at step 4 branches on a vertex of degree Δ. We show that in both branches $F[x]$ and $F[\neg x]$, μ is reduced at least by 1.

Indeed, the measure decreases by α_Δ, because the algorithm branches on a vertex of degree Δ. The degree of each neighbor of x is reduced, so the complexity is decreased at least by δ (as δ is the minimal amount by which μ is decreased when the degree of a vertex is reduced). This causes a complexity decrease of $\Delta \cdot \delta$. Lemma 1 guarantees that removing variables of degree 2 does not increase μ. It follows from (2) that $\Delta \cdot \delta + \alpha_\Delta \ge 1$. Therefore μ decreases at least by 1. Then

$$T(n_3, \ldots, n_\Delta) \le 2 \cdot 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta - 1} + poly(|F|) \le 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta} + poly(|F|).$$

Thus, the running time of the algorithm MetaAlg is $O^*(2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta})$. □

As easy consequence of the just proved lemma is an upper bound $O^*(2^{\alpha n})$, where $\alpha = \max(\alpha_\Delta, \ldots \alpha_3)$. From $\alpha_i < 1$ and (2) we conclude that α_i's increase with i, which means that $\alpha = \alpha_\Delta$.

It is known [3] that $(n,3)$-MAX-2-SAT can be solved in time $O^*(2^{n/6.7})$. Also, the fact that vertices of degree at most 2 can be removed implies that $(n,3)$-MAX-2-CSP can be solved in $O^*(2^{n/4})$. Indeed, when branching on a vertex of degree 3 we can remove all its neighbors in both branches (so, the number of vertices is decreased at least by 4).

Corollary 1. *The following algorithm solves MAX-2-SAT (MAX-2-CSP) in* $O^*(2^{n(1-\frac{10/3}{\Delta+1})})$ *(*$O^*(2^{n(1-\frac{3}{\Delta+1})})$*) time.*

```
SimpleAlg
Input: F – an instance of MAX-2-SAT or MAX-2-CSP.
Output: Opt(F).
Method.
 1. Remove all vertices of degree < 3 (using Lemma 1).
 2. If F does not contain 2-clauses, then return the result.
 3. If the maximal vertex degree of F is 3, then call the known algorithm
    for (n,3)-MAX-2-SAT or (n,3)-MAX-2-CSP, respectively.
 4. Choose a vertex x of maximal degree Δ.
 5. Return max(SimpleAlg(F[x]), SimpleAlg(F[¬x])).
```

Proof. SimpleAlg is obtained from MetaAlg. As a parameter A SimpleAlg takes himself if $i > 3$ and described algorithms if $i = 3$. We will choose α_i satisfying (2).

As mentioned above, $(n,3)$-MAX-2-SAT ($(n,3)$-MAX-2-CSP) can be solved by SimpleAlg in $O^*(2^{n/6.7})$ ($O^*(2^{n/4})$) time. Hence, to minimize $\max(\alpha_3,\alpha_4)$, according to (2), we can choose α_3 and α_4 as follows:

$$\begin{aligned} \text{MAX-2-SAT:} \qquad & \alpha_3=\frac{1}{6}, \alpha_4=\frac{1}{3} \\ \text{MAX-2-CSP:} \qquad & \alpha_3=\frac{1}{4}, \alpha_4=\frac{2}{5}. \end{aligned}$$

Thus, we get upper bounds $O^*(2^{n/3})$ and $O^*(2^{2n/5})$ for $(n,4)$-MAX-2-SAT and $(n,4)$-MAX-2-CSP, respectively. Now let $\Delta > 4$. To (2) to hold we can set α_i as follows:

$$\alpha_i=\frac{1+i\cdot\alpha_{i-1}}{i+1}. \tag{3}$$

Below we state several simple properties of α_i's that will be needed in further analysis.

- $\alpha_i=1-4\frac{1-\alpha_3}{i+1}$.
  By expanding (3), one gets:

$$\alpha_i=\frac{1+i\cdot\alpha_{i-1}}{i+1}=\frac{1+i\cdot\frac{1+(i-1)\alpha_{i-2}}{i}}{i+1}=$$
$$\frac{2+(i-1)\alpha_{i-2}}{i+1}=\ldots=\frac{i-3+4\alpha_3}{i+1}=1-4\frac{1-\alpha_3}{i+1}. \tag{4}$$

- $\alpha_i<1$.
  This follows immediately from the previous property and the fact that $\alpha_3<1$.
- α_i increases with i.
  This follows immediately from $\alpha_i=1-4\frac{1-\alpha_3}{i+1}$.
- $\alpha_i-\alpha_{i-1}=\frac{1-\alpha_i}{i}$.
  This follows from (3).

For MAX-2-SAT ($\alpha_3=\frac{1}{6}$) we get: $\alpha_\Delta=1-\frac{10/3}{\Delta+1}$. For MAX-2-CSP ($\alpha_3=\frac{1}{4}$): $\alpha_\Delta=1-\frac{3}{\Delta+1}$.

Show that these α_i satisfy the condition of the lemma. First, SimpleAlg solves $(n,\Delta-1)$-MAX-2-SAT ($(n,\Delta-1)$-MAX-2-CSP) in $2^{\alpha_3 n_3+\ldots+\alpha_{\Delta-1}n_{\Delta-1}}$ time by induction.

It remains to show that (2) holds. First, show that $\alpha_3\geq\frac{1-\alpha_\Delta}{\Delta}$, for $\Delta\geq 4$.

$$\frac{1-\alpha_\Delta}{\Delta}\leq\frac{1-1+\frac{10/3}{\Delta+1}}{\Delta}\leq\frac{10}{3\Delta(\Delta+1)}\leq\frac{10}{3\cdot4\cdot(4+1)}\leq\frac{1}{6}\leq\alpha_3.$$

Now show that $\alpha_i-\alpha_{i-1}\geq\frac{1-\alpha_\Delta}{\Delta}$ for $i\leq\Delta$.

From properties of α_i we know that $\alpha_i-\alpha_{i-1}=\frac{1-\alpha_{i-1}}{i+1}$. α_i increases, so for $i<\Delta$, $\frac{1-\alpha_{i-1}}{i+1}>\frac{1-\alpha_\Delta}{\Delta}$. Hence, $\alpha_i-\alpha_{i-1}=\frac{1-\alpha_{i-1}}{i+1}>\frac{1-\alpha_\Delta}{\Delta}$ for $i<\Delta$.

For $i=\Delta$, $\alpha_i-\alpha_{i-1}=\frac{1-\alpha_\Delta}{\Delta}$. □

The described algorithm has the running time $O^*(2^{n(1-\frac{10/3}{\Delta+1})})$ and $O^*(2^{n(1-\frac{3}{\Delta+1})})$ for MAX-2-SAT and MAX-2-CSP, respectively. Note that our algorithm is based on upper bound $2^{n/6}$ [6], while a stronger bound $2^{n/6.7}$ [3] is known. The latter bound would not improve our algorithm as for smaller α_3 one needs a larger α_4 to satisfy $\alpha_3 \geq \frac{1-\alpha_4}{4}$ from (2).

The presented algorithm already improves the known bounds $O^*(2^{n(1-\frac{1}{\Delta-1})})$ [4] and $O^*(2^{n(1-\frac{2}{\Delta+1})})$ [5] w.r.t. Δ. In Section 4, we further improve these bounds by changing the algorithm slightly.

3 Going from the Maximal Degree to the Average Degree

In this section, we generalize the results of the previous section from the maximal degree to the average degree. Informally, we show that the worst case of the considered algorithm is achieved in case when all the vertices have the same degree (and so $d = \Delta$). In this section we consider only simplified formulas (i.e. formulas without vertices of degree less than 3).

Theorem 1. *If an algorithm X solves MAX-2-SAT (MAX-2-CSP) in time* $O^*(2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta})$ *and for all* $i > 3$*,*

$$2\alpha_i \geq \alpha_{i+1} + \alpha_{i-1}, \tag{5}$$

then the algorithm X solves MAX-2-SAT (MAX-2-CSP) in time $O^*(2^{n(\alpha_D + \epsilon(\alpha_{D+1} - \alpha_D))})$*, where* $D = \lfloor d \rfloor$*,* $\epsilon = d - D$ *and* $d = \frac{2m}{n}$ *is the average vertex degree.*

Proof. As d is the average degree of the simplified graph,

$$3n_3 + 4n_4 + \ldots \Delta n_\Delta = nd. \tag{6}$$

Subtract from (6) the equation $n_3 + n_4 + \ldots + n_\Delta = n$ multiplied by d:

$$\sum_{i=3}^{\Delta} (i - d) n_i = 0.$$

By substitution d by $D + \epsilon$ in this equality, we get

$$\sum_{i=3}^{\Delta} (i - D) n_i = \epsilon n. \tag{7}$$

Let $\sigma = \alpha_{D+1} - \alpha_D$. Show that

$$n_i \alpha_i \leq n_i \alpha_D + n_i (i - D)\sigma. \tag{8}$$

Condition (5) can be written as follows:

$$\alpha_i - \alpha_{i-1} \geq \alpha_{i+1} - \alpha_i.$$

Then, for all $i \le D$,

$$\alpha_{i+1} - \alpha_i \ge \alpha_{i+2} - \alpha_{i+1} \ge \ldots \ge \alpha_{D+1} - \alpha_D = \sigma.$$

Hence

$$\alpha_i \le \alpha_{i+1} - \sigma \le \alpha_{i+2} - 2\sigma \le \ldots \le \alpha_D + (i-D)\sigma.$$

Therefore $\alpha_i n_i \le \alpha_D n_i + \sigma(i-D)n_i$, for all $i \le D$. By the same argument, $\alpha_i n_i \le \alpha_D n_i + \sigma(i-D)n_i$, for all $i \ge D$.

Then the exponent of the running time of the algorithm is

$$\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta \le \alpha_D n_3 + \sigma(3-D)n_3 + \ldots + \alpha_D n_\Delta + \sigma(\Delta - D)n_\Delta =$$
$$\alpha_D(n_3 + n_4 + \ldots + n_\Delta) + \sigma\sum_{i=3}^{\Delta}(i-D)n_i = \alpha_D n + \sigma\sum_{i=3}^{\Delta}(i-D)n_i \overset{(by (7))}{=}$$
$$\alpha_D n + \sigma\epsilon n = n(\alpha_D + \epsilon(\alpha_{D+1} - \alpha_D)).$$

□

It can be shown that this bound holds also for formulas containing variables of degree less than 3 for MAX-2-CSP with average degree $d \ge 3$ and for MAX-2-SAT with average degree $d \ge 4$.

Corollary 2. *The algorithm* SimpleAlg *solves MAX-2-SAT (MAX-2-CSP) in time* $O^*(2^{n(1-\frac{10/3}{d+1})})$ $(O^*(2^{n(1-\frac{3}{d+1})}))$.

Proof. From (3):

$$\alpha_{i+1} = \frac{1 + \alpha_i(i+1)}{i+2}, \qquad \alpha_{i-1} = \frac{\alpha_i(i+1) - 1}{i}.$$

Then

$$\alpha_{i+1} + \alpha_{i-1} = \frac{\alpha_i(i+1)+1}{i+2} + \frac{\alpha_i(i+1)-1}{i} =$$
$$2\alpha_i + \frac{1-\alpha_i}{i+2} + \frac{\alpha_i - 1}{i} = 2\alpha_i - (1-\alpha_i)(\frac{1}{i} - \frac{1}{i+2}) < 2\alpha_i.$$

Therefore (5) holds. From Theorem 1 it follows that the exponent of the running time is $\alpha_D + \epsilon(\alpha_{D+1} - \alpha_D)$.

To prove $O^*(2^{n(1-\frac{10/3}{d+1})})$ and $O^*(2^{n(1-\frac{3}{d+1})})$ upper bounds for MAX-2-SAT and MAX-2-CSP it remains to show that $\alpha_D + \epsilon(\alpha_{D+1} - \alpha_D) \le 1 - 4\frac{1-\alpha_3}{d+1}$.

We know that $\alpha_i = 1 - 4\frac{1-\alpha_3}{i+1}$.

$$\alpha_D + \epsilon(\alpha_{D+1} - \alpha_D) = (1-\epsilon)\alpha_D + \epsilon\alpha_{D+1} =$$
$$1 - \epsilon - 4\frac{(1-\epsilon)(1-\alpha_3)}{D+1} + \epsilon - 4\frac{\epsilon(1-\alpha_3)}{D+2} = 1 - 4\frac{(1-\alpha_3)(D+2-\epsilon)}{(D+1)(D+2)} =$$
$$1 - 4\frac{(1-\alpha_3)(D+2-\epsilon)(D+1+\epsilon)}{(D+1)(D+2)(D+1+\epsilon)} = 1 - 4\frac{(1-\alpha_3)((D+1)(D+2) + \epsilon - \epsilon^2)}{(D+1)(D+2)(D+1+\epsilon)} <$$
$$1 - 4\frac{1-\alpha_3}{D+1+\epsilon} = 1 - 4\frac{1-\alpha_3}{d+1}.$$

□

4 An Algorithm for MAX-2-SAT and MAX-2-CSP

Recall that $\delta = \min(\alpha_\Delta - \alpha_{\Delta-1}, \alpha_{\Delta-1} - \alpha_{\Delta-2}, \ldots, \alpha_4 - \alpha_3, \alpha_3)$ is the minimal amount by which μ is decreased when the degree of a vertex is decreased. The algorithm from Section 2 at each iteration branches on a vertex of maximal degree Δ. Therefore the complexity measure decreases at least by $\alpha_\Delta + \Delta \cdot \delta$. In Corollary 1 we showed that $\delta = \alpha_\Delta - \alpha_{\Delta-1}$, and in accordance with this we choose α_Δ such that

$$\alpha_\Delta + \Delta \cdot \delta = \alpha_\Delta + \Delta(\alpha_\Delta - \alpha_{\Delta-1}) = 1.$$

Therefore we get $\alpha_\Delta = \frac{1+\Delta\alpha_{\Delta-1}}{\Delta+1}$.

We now improve these bounds. To do this, at each iteration we choose a branching vertex such that it has a neighbor with degree less than Δ. Then we can choose α_Δ based on the following equation:

$$\alpha_\Delta + (\Delta - 1)(\alpha_\Delta - \alpha_{\Delta-1}) + (\alpha_{\Delta-1} - \alpha_{\Delta-2}) = 1.$$

Then

$$\alpha_\Delta = \frac{1 + \alpha_{\Delta-2} + (\Delta - 2)\alpha_{\Delta-1}}{\Delta}. \tag{9}$$

We present an algorithm for which the recurrence (9) holds.

```
Max2Alg
Input: $F$ – instance of MAX-2-SAT or MAX-2-CSP.
Output: $Opt(F)$.
Method.
 1. Remove all vertices of degree $< 3$ (using Lemma 1).
 2. If $F$ does not contain 2-clauses, then return the result.
 3. If the formula $F$ has connected components $F_1$ and $F_2$, then
    return Max2Alg($F_1$) + Max2Alg($F_2$).
 4. If the maximal vertex degree of $F$ is 3, then call the known algorithm
    for $(n,3)$-MAX-2-SAT or $(n,3)$-MAX-2-CSP, respectively.
 5. If $F$ contains a vertex of maximal degree $\Delta$ that has at least
    one neighbor whose degree is not maximal, then let $x$ be
    this vertex. Otherwise, let $x$ be any vertex of maximal degree.
 6. Return max(Max2Alg($F[x]$), Max2Alg($F[\neg x]$)).
```

The algorithm Max2Alg is SimpleAlg extended by two steps (given in bold).

Lemma 3. *If*

1. *Algorithm called at step 4 solves $(n,3)$-MAX-2-SAT ($(n,3)$-MAX-2-CSP) in time $O^*(2^{\alpha_3 n})$,*

2. $\delta_i = \min(\alpha_i - \alpha_{i-1}, \ldots, \alpha_4 - \alpha_3, \alpha_3)$,
for each i:

$$\alpha_i + (i-1)\delta_i + \delta_{i-1} \geq 1, \tag{10}$$

then the algorithm Max2Alg *solves MAX-2-SAT (MAX-2-CSP) in time* $O^*(2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta + \tau})$, *where* $\tau = \sum_i (1 - \tau_i)$,

$$\tau_i = \alpha_i + i\delta_i.$$

Proof. We prove this by induction on the number of vertices. Again we use the formula complexity measure $\mu = \alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta$. By $T(n_3, \ldots, n_\Delta)$ we denote the running time of the algorithm on formula with n_i vertices of degree i, $3 \leq i \leq \Delta$.

It is clear that connected components of an input formula can be handled independently. This is done at the step 4 of the algorithm. So, below we assume that the graph of the formula is connected.

If there is both a vertex of degree Δ and a vertex of degree less than Δ, then recursive calls decrease μ at least by 1. Indeed, we branch on a vertex x of degree Δ and we decrease μ by α_Δ. We choose x such that x has a neighbor of degree less than Δ. This neighbor causes a complexity decrease of at least $\delta_{\Delta-1}$. Each of the remaining $\Delta - 1$ neighbors decrease μ at least by δ_Δ. It follows from (10), that $\alpha_\Delta + (i-1)\delta_\Delta + \delta_{\Delta-1} \geq 1$. Therefore,

$$T(n_3, \ldots, n_\Delta) \leq 2 \cdot 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta + \tau - 1} + poly(|F|) \leq 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta + \tau} + poly(|F|),$$

Now assume that the graph contains degree Δ variables only. Since during the work of the algorithm the degrees of variables can only decrease such a graph cannot appear in this branch again. So, at this iteration the algorithm makes two recursive calls for formulas whose complexity measure is less than the complexity of the initial formula at least by τ_Δ.

$$\begin{aligned} T(n_3, \ldots, n_\Delta) &\leq 2 \cdot 2^k \cdot 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta - k - \tau_\Delta + \sum_{i=3}^{\Delta-1}(1-\tau_i)} + poly(|F|) \\ &= 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta + \sum_{i=3}^{\Delta}(1-\tau_i)} + poly(|F|), \end{aligned}$$

where k is the number of iterations of the algorithm while maximal degree is Δ. $\square$

Corollary 3. *If for all* $i > 3$,

$$2\alpha_i \geq \alpha_{i+1} + \alpha_{i-1}, \quad \alpha_i - \alpha_{i-1} \leq \alpha_3,$$

then the running time of Max2Alg *is*

$$O^*(2^{n(\alpha_D + \epsilon(\alpha_{D+1} - \alpha_D))}),$$

where $D = \lfloor d \rfloor$, $d = D + \epsilon$, $d = \frac{2m}{n}$ *is the average degree of the vertices.*

Proof. It can be shown by induction on i, that $\alpha_i \geq 1 - \frac{4}{i+1} = \frac{i-3}{i+1}$ From the corollary condition it follows that $\alpha_i - \alpha_{i-1}$ decreases with increasing i. Therefore $\delta_i = \alpha_i - \alpha_{i-1}$. Then

$$\tau = \sum_{i=3}^{\Delta}(1-\tau_i) = \sum_{i=3}^{\Delta}(1-\alpha_i - i\delta_i) = \sum_{i=3}^{\Delta}(1-\alpha_i - i\alpha_i + i\alpha_{i-1}) =$$

$$\Delta - 2 - (\Delta+1)\alpha_\Delta = \Delta(1-\alpha_\Delta) + O(1) = \Delta(1 - \frac{\Delta-3}{\Delta+1}) + O(1) = O(1).$$

We use Theorem 1 to complete the proof. □

It is easy to show that α_i's, chosen by (9), satisfy the condition of Corollary 3, so we have the bounds w.r.t. d. We get the following sequence for MAX-2-SAT: $\alpha_3 = 1/6$, $\alpha_4 = 1/3$, $\alpha_5 = 13/30$, $\alpha_6 = 23/45$, and the following sequence for MAX-2-CSP: $\alpha_3 = 1/4$, $\alpha_4 = 3/8$, $\alpha_5 = 19/40$, $\alpha_6 = 131/240$. At each step we can continue computing α_i's with weaker equality (4). This gives us an explicit formula for α_i for all i, if α_k is already computed from a stronger equality (9):

$$\alpha_i = \frac{i-k+(k+1)\alpha_k}{i+1} =$$

$$1 - \frac{k+1-(k+1)\alpha_k}{i+1} = 1 - \frac{k+1}{i+1}(1-\alpha_k). \tag{11}$$

The values of the first α_k for MAX-2-SAT and MAX-2-CSP are shown in Table 1. According to the table we can calculate the running time of MAX2ALG for graphs with the average degree i. The running time is $O^*(2^{n\alpha_i})$.

Table 1. The values of α_k for $3 \leq k \leq 10$

d	MAX-2-SAT		MAX-2-CSP	
3	1/6	≈ 0.1666	1/4	≈ 0.2500
4	1/3	≈ 0.3333	3/8	≈ 0.3750
5	13/30	≈ 0.4333	19/40	≈ 0.4750
6	23/45	≈ 0.5111	131/240	≈ 0.5458
7	359/630	≈ 0.5698	1009/1680	≈ 0.6005
8	1553/2520	≈ 0.6162	8651/13440	≈ 0.6436
9	14827/22680	≈ 0.6537	82069/120960	≈ 0.6784
10	155273/226800	≈ 0.6846	855371/1209600	≈ 0.7071

Consider, e.g., the case $k = 5$. From (11) for MAX-2-SAT $\alpha_i = 1 - \frac{3.4}{d+1}$, for MAX-2-CSP $\alpha_i = 1 - \frac{3.15}{d+1}$, if $d \geq 5$. For $k = 8$, $\alpha_i = 1 - \frac{3.45}{d+1}$ for MAX-2-SAT, $\alpha_i = 1 - \frac{3.20}{d+1}$ for MAX-2-CSP, if $d \geq 8$. So, the running time of MAX2ALG for MAX-2-SAT (MAX-2-CSP) is $O^*(2^{n(1-\frac{3.45}{d+1})})$ $(O^*(2^{n(1-\frac{3.20}{d+1})}))$, if $d \geq 8$.

The upper bounds w.r.t. d imply upper bounds w.r.t. m. Indeed, $n = \frac{2m}{d}$, so the running time is $O^*(2^{\frac{2m(\alpha_D + \epsilon(\alpha_{D+1} - \alpha_D))}{d}})$. For MAX-2-CSP, the minimum of this function is at $d = 5$ and is equal to $O^*(2^{\frac{m}{5.263}})$. This matches the best known upper bound for MAX-2-CSP w.r.t. m [8].

5 Further Directions

To improve the upper bounds for MAX-2-SAT it is enough to improve any α_i, $i \geq 4$. All the subsequent α_i's will also be improved recursively. We can improve bounds from Section 4 for $(n, \varDelta)$-MAX-2-SAT by using item 5 of Lemma 4.1 in [3], saying that a neighbor of a variable x of degree 3 in at least one of two branches $F[x]$ and $F[\neg x]$ is not just eliminated by simplification rules, but is assigned a constant. Using this lemma and Lemma 3 we can get $\alpha_4 = 1/3.43$. Also for small i, α_i can be chosen using the algorithm from [8].

Also, as shown in Section 4, improving an upper bound for either $(n, 3)$-, $(n, 4)$-, or $(n, 5)$-MAX-2-CSP w.r.t. n, gives an improved upper bound for MAX-2-CSP w.r.t. m (the number of clauses).

Acknowledgments

I would like to thank my supervisor Alexander S. Kulikov for help in writing this paper and valuable comments.

References

1. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theoretical Computer Science **348**(2-3) (2005) 357–365
2. Dantsin, E., Wolpert, A.: MAX-SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In: Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing. Volume 4121 of Lecture Notes in Computer Science. (2006) 266–276
3. Kulikov, A., Kutzkov, K.: New upper bounds for the problem of maximal satisfiability. Discrete Mathematics and Applications **19** (2009) 155–172
4. Fürer, M., Kasiviswanathan, S.P.: Exact Max 2-Sat: Easier and Faster. In: Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science. SOFSEM 2007, Springer-Verlag (2007) 272–283
5. Scott, A.D., Sorkin, G.B.: Linear-programming design and analysis of fast algorithms for Max 2-CSP. Discrete Optimization **4**(3-4) (2007) 260–287
6. Kojevnikov, A., Kulikov, A.S.: A new approach to proving upper bounds for MAX-2-SAT. In: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete Algorithm. SODA 2006 (2006) 11–17
7. Chen, J., Kanj, I.: Improved exact algorithms for Max-Sat. Discrete Applied Mathematics **142**(1-3) (2004) 17–27
8. Gaspers, S., Sorkin, G.B.: A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between. (2009) 606–615
9. Croce, F.D., Kaminski, M., Paschos, V.: An exact algorithm for MAX-CUT in sparse graphs. Operations Research Letters **35**(3) (2007) 403–408