A Microstructure Based Approach to Constraint Satisfaction Optimisation Problems

Ola Angelsmark[*]
Department of Computer and Information Science
Linköpings Universitet, Sweden
olaan@ida.liu.se

Johan Thapper[†]
Department of Mathematics
Linköpings Universitet, Sweden
jotha@mai.liu.se

Abstract

We study two constraint satisfaction optimisation problems: The MAX VALUE problem for CSPs, which, somewhat simplified, aims at maximising the sum of the (weighted) variable values in the solution, and the MAX IND problem, where the goal is to find a satisfiable *subinstance* of the original instance containing as many variables as possible. Both problems are *NP*-hard to approximate within $n^{1-\epsilon}, \epsilon > 0$, where n is the number of variables in the problems, which implies that it is of interest to find exact algorithms. By exploiting properties of the microstructure, we construct algorithms for solving instances of these problems with small domain sizes, and then, using a probabilistic reasoning, we show how to get algorithms for more general versions of the problems. The resulting algorithms have running times of $\mathcal{O}\left((0.585d)^n\right)$ for MAX VALUE $(d, 2)$-CSP, and $\mathcal{O}\left((0.503d)^n\right)$ for MAX IND $(d, 2)$-CSP. Both algorithms represent the best known theoretical bounds for their respective problem, and, more importantly, the methods used are applicable to a wide range of optimisation problems.

Introduction

Constraint satisfaction problems (CSPs) provide a natural and efficient way of formulating problems in a wide range of areas, such as machine vision, scheduling, temporal reasoning, graph problems and diagnostic reasoning (Kumar 1992), just to name a few. Computer science in general, and artificial intelligence in particular, has benefited greatly from CSPs. Many of the most well-studied computational problems are, in fact, constraint satisfaction problems, e.g. the satisfiability problem for propositional logic, the k-colouring problem for graphs and the graph homomorphism problem, cf. Jeavons (1998). The most thoroughly studied problem regarding CSPs is the following: We are given a set of variables taking their values from a finite domain, and a set of constraints, e.g. relations, restricting which values the different variables can simultaneously assume. The question is now whether the variables can be assigned values consistent with all of the constraints. This is called the *decision problem for CSPs*, and it is known to be *NP*-complete in general (Garey & Johnson 1979). For (d, l)-CSPs, i.e. CSPs with domain size d and constraint arity l, the currently best known algorithms are, for $d < 11$, the $\mathcal{O}\left((0.4518d)^n\right)$ time algorithm by Eppstein (2001), and, for $d \geq 11$, Feder & Motwani's (2002) algorithm which has a running time of $\mathcal{O}\left((d!)^{n/d}\right)$. (Here, and throughout the paper, n is the number of variables and d the size of the domain of a problem.)

We will discuss two *optimisation* problems for CSPs, i.e. problems where we are not satisfied with just any solution, but have some additional requirements. Of course, the algorithms for the classical decision problem are somewhat faster than the ones we present, but the gap is not a large one—and it can certainly be shrunk even further using the methods we describe.

The problems under discussion are MAX IND CSP and MAX VALUE CSP. Both problems are *NP*-hard to approximate within $n^{1-\epsilon}$ (see Jonsson & Liberatore (1999) and Jonsson (1998), respectively) unless $P = NP$. Consequently, it is of interest to find efficient exact algorithms.

The first problem we look at is the MAX IND $(d, 2)$-CSP problem. In this setting, we want to find a satisfiable *subinstance* containing as many variables as possible. A subinstance is here a subset of the variables, together with the constraints that involve all of these, and only these variables. MAX IND $(d, 2)$-CSP is in some sense dual to the classical MAX CSP in that it does not attempt to maximise the number of satisfied *constraints*, but rather tries to maximise the number of *variables* that are assigned values without violating any constraints. It is not difficult to come up with examples where MAX IND CSP is a more natural problem than MAX CSP, e.g. in real-time scheduling. Say the scheduler has discovered that the tasks cannot be scheduled due to, e.g. resource limitations. If it were to consider the problem as a MAX CSP instance, then the resulting schedule might still not be of any use, since the relaxation does not introduce additional resources. If, on the other hand, it were to solve the corresponding MAX IND CSP instance, it would get a partial schedule which would be guaranteed to run—since no constraints are violated. Jonsson & Liberatore (1999) contains several results regarding the complexity of this problem. In particular, it is shown that the problem is either always sat-

[*]Ola Angelsmark is supported in part by the Swedish National Graduate School in Computer Science (CUGS), and in part by the *Swedish Research Council* (VR), under grant 621-2002-4126.

[†]Johan Thapper is supported by the *Programme for Interdisciplinary Mathematics*, Department of Mathematics, Linköpings Universitet, Sweden.

isfiable (and trivial) or *NP*-hard. We exploit properties of the *microstructure graph* (Jégou 1993) of the instance by first observing that a maximum independent set in the microstructure corresponds to a maximum induced subinstance of the original problem. From this observation, we get a specialised algorithm for MAX IND $(2,2)$-CSP, and then we arrive at an algorithm for the more general MAX IND $(d,2)$-CSP which has a running time of $\mathcal{O}\left((0.5029d)^n\right)$. (The necessary definitions can be found in the next section.)

The MAX VALUE CSP problem, which we look at next, can be seen as a generalisation of several *NP*-complete problems, such as MAX ONES and MAX DONES, as well as the restricted variant of INTEGER PROGRAMMING where we have bounded variables and a bounded number of non-zero entries in each matrix row. Strictly speaking, the algorithm we propose is not limited to solving instances of this particular problem; In fact, it is applicable to any problem where we are given a CSP instance together with a function assigning weights to the variables, and the goal is to maximise the sum of these weights—while at the same time finding a satisfying assignment, of course. One problem which matches this description is INTEGER PROGRAMMING, and in the future it would be interesting to see how the algorithm applies to this problem, since it is different from the typical branch-and-bound approach.

In order to solve the general MAX VALUE $(d,2)$-CSP, we first construct a specialised algorithm for the case with domain size 3. Again, the algorithm exploits properties of the microstructure graph of the instances, and, through careful analysis, we get an algorithm for solving MAX VALUE $(3,2)$-CSP within $\mathcal{O}\left(1.7548^n\right)$. We can then use this specialised, deterministic, algorithm to construct a general, probabilistic, algorithm for MAX VALUE $(d,2)$-CSP (see Theorem 3) which has a running time of $\mathcal{O}\left((0.5849d)^n\right)$.

An interesting property shared by both of these problems is that for certain CSPs, we can efficiently extended algorithms for small domains into algorithms for larger domain sizes by selecting a large number of restricted instances and solving these. Both the algorithms in Theorem 1 and 3 use this technique. In Theorem 3 the smaller instances must have domain size 3 but in Theorem 1 they can be of any size, and it turns out we can choose this size optimally.

The algorithms we present for the two problems are both probabilistic and they have a probability of error less than or equal to e^{-1}. While this may seem rather high ($\mathrm{e}^{-1} \approx 0.38$), it is possible to achieve arbitrarily low error probability by iterating the algorithm; For example, after only 5 iterations we have a probability of *success* greater than 99.3%, and after 5 more it is greater than 99.99%. For a given probability of error, the number of iterations is constant, and thus does not add to the time complexity of the algorithm.

Preliminaries

A *(d,l)-constraint satisfaction problem ((d,l)-CSP)* is a triple (X,D,C) with X a finite set of variables, D a finite set of domain values, with $|D| = d$, and C a set of constraints $\{c_1, c_2, \ldots, c_k\}$. Each constraint $c_i \in C$ is a structure $R(x_{i_1}, \ldots, x_{i_j})$, where $j \leq l$, $x_{i_1}, \ldots, x_{i_j} \in X$ and $R \subseteq D^j$. A *solution* to a CSP instance is a function $f : X \to D$ s.t. for each constraint $R(x_{i_1}, \ldots, x_{i_j}) \in C$, $(f(x_{i_1}), \ldots, f(x_{i_j})) \in R$. Given a variable x and a set $E \subseteq D$, we let $(x;E)$ denote the unary constraint $x \in E$. Given a (d,l)-CSP instance, the basic computational problem is to decide whether it has a solution or not. The special case when $d = 2$ and we have binary constraints, i.e. $(2,2)$-CSP, can be viewed as a 2-SAT formula. A 2-SAT formula is a conjunction of a number of clauses, where each clause is on one of the forms $(p \vee q)$, $(\neg p \vee q)$, $(\neg p \vee \neg q)$, (p), $(\neg p)$. The set of variables of a formula F is denoted $\mathrm{Var}(F)$, and an occurrence of a variable or its complement in a formula is termed a *literal*.

Definition 1 (Dahllöf, Jonsson, & Wahlström (2004)) *Let F be a 2-SAT formula, and let L be the set of all literals for all variables occurring in F. Given a vector $\bar{w}$ of weights and a model M for F, we define the weight $W(M)$ of M as*

$$W(M) = \sum_{\{l \in L \mid l \text{ is true in } M\}} \bar{w}(l)$$

The problem of finding a maximum weighted model for F is denoted 2-SAT$_w$.

Dahllöf *et al.* (2004) presents an algorithm for counting the number of maximum weighted solutions to 2-SAT instances. This algorithm has a running time of $\mathcal{O}\left(1.2561^n\right)$, and it can easily be modified to return one of the solutions. We will denote this modified algorithm *2-SAT$_w$*.

A *graph* G consists of a set $V(G)$ of *vertices*, and a set $E(G)$ of *edges*, where each element of $E(G)$ is an unordered pair of vertices. The *size* of a graph, denoted $|G|$ is the number of vertices. The *neighbourhood* of a vertex $v \in V(G)$ is the set of all vertices adjacent to v, v itself excluded, denoted $N_G(v)$; $N_G(v) = \{w \in V(G) \mid (v,w) \in E(G)\}$. The *degree* $\deg_G(v)$ of a vertex v is the size of its neighbourhood, $|N_G(v)|$. When G is obvious from the context, it can be omitted as a subscript. If we pick a subset S of the vertices of a graph together with the edges between them (but no other edges), then we get the *subgraph of G induced by S*, $G(S)$. $G(S)$ has vertex set S and edge set $\{(v,u) \mid v,u \in S, (v,u) \in E(G)\}$. If the induced subgraph has empty edge set, then it forms an *independent set*.

Definition 2 (Jégou (1993)) *Given a binary CSP $\Theta = (X,D,C)$, the* microstructure graph *of Θ is an undirected graph G defined as follows:*

1. *For every variable $x \in X$, and domain value $a \in D$, there is a vertex $x[a] \in V(G)$.*
2. *There is an edge $(x[a_1], y[a_2]) \in E(G)$ iff (a_1, a_2) satisfies the constraint between x and y.*

We assume that there is exactly one constraint between any pair of variables, and any variables with no explicit constraint is assumed to be constrained by the universal constraint which allows all values.

(For technical reasons, we consider the complement of the microstructure graph.) Obviously, the microstructure of a

finite domain problem will use polynomial space, since there will be dn vertices in the graph.

In order to avoid confusing the size of a graph with the number of variables in a problem (both of which are traditionally denoted n), we will explicitly write $|V(G)|$ for the problem size whenever we are using the running time of a graph algorithm. By adding weights to the vertices of the microstructure, we get a *weighted microstructure* and since each vertex corresponds to a variable being assigned a value, we can efficiently move from a 2-SAT$_w$ instances to its weighted microstructure, and back.

In the analysis of the algorithm for MAX VALUE, we will often encounter recursions on the form

$$T(n) = \sum_{i=1}^{k} T(n - r_i) + p(n)$$

where $p(n)$ is a polynomial in n, and $r_i \in \mathbb{N}^+$. These equations satisfy $T(n) \in \mathcal{O}(\lambda(r_1, \ldots, r_k)^n)$, where $\lambda(r_1, \ldots, r_k)$ is the largest real-valued root to $1 - \sum_{i=1}^{k} x^{-r_i}$ (see Kullmann (1999).) Note that this bound does not depend on neither $p(n)$ nor the boundary conditions $T(1) = b_1, \ldots, T(k) = b_k$. We will usually call λ the *work factor* (in the sense of (Eppstein 2001).)

Algorithm for Max Ind CSP

The first problem we turn our attention to is the following:
MAX IND (d, l)-CSP
Instance: A CSP instance $\Theta = (V, D, C)$
Output: A subset $V' \subseteq V$ such that $\Theta|V'$ is satisfiable and $|V'|$ is maximal.

Here, $\Theta|V' = (V', D, C')$ is the *subinstance induced by* V', and

$$C' = \{c \in C \mid c(x_1, x_2, \ldots, x_l) \in C, x_1, \ldots, x_l \in V'\}.$$

Or, in words, C' is the subset of C which contains all the constraints involving only variables from V'. First, we need the following proposition.

Proposition 1 *A binary CSP $\Theta = (V, D, C)$ contains a maximum induced subinstance $\Theta|V' = (V', D, C')$ iff the microstructure graph of Θ contains a maximum independent set α with $|\alpha| = |V'|$.*

This immediately gives us the following corollary:

Corollary 1 *There exists an algorithm for solving* MAX IND $(d, 2)$*-CSP in $\mathcal{O}\left(1.2025^{dn}\right)$ time.*

The combination of Corollary 1 with the following lemma turns out to be a powerful one, as we will see later.

Lemma 1 *Assume there exists an $\mathcal{O}(c^n)$ algorithm for* MAX IND (k, l)*-CSP. Let $\mathcal{I}_k$ denote the set of (d, l)-CSP instances satisfying the following: For each $(X, D, C) \in \mathcal{I}_k$, and every $x \in X$, there exists a unary constraint $(x; S)$ in C s.t. $|S| \leq k$. Then, the* MAX IND *problem restricted to instances in $\mathcal{I}_k$ can be solved in $\mathcal{O}(c^n)$.*

Algorithm 1 Algorithm for MAX IND $(d, 2)$-CSP.

MaxInd(Θ)
1. $s := \emptyset$
2. **repeat** $(d/k)^n$ times
3. Randomly choose $\Delta \in S_k$
4. Let G be the microstructure graph of Θ_Δ
5. $\alpha :=$*MIS*(G)
6. **if** $|s| < |\alpha|$ **then**
7. $s := \alpha$
8. **end repeat**
9. **return** s

Now we use Corollary 1 to solve randomly chosen, restricted instances of the original CSP-instance—and Lemma 1 lets us do this. In Algorithm 1 we assume that we have an algorithm *MIS* for finding maximum independent sets in arbitrary graphs.

Theorem 1 *Given an $\mathcal{O}\left(c^{|V(G)|}\right)$ algorithm for solving* MAXIMUM INDEPENDENT SET *for arbitrary graphs, there exists an algorithm for* MAX IND $(d, 2)$*-CSP, which returns an optimal solution with probability at least $1 - \mathrm{e}^{-1}$, where $\mathrm{e} = 2.7182\ldots$, and has a running time of $\mathcal{O}\left((d/k \cdot c^k)^n\right)$, for $d \geq k$. Furthermore, either $\lceil (\ln c)^{-1} \rceil$ or $\lfloor (\ln c)^{-1} \rfloor$ is the best choice for k.*

Corollary 2 *For $d \geq 5$, there exists a probabilistic algorithm for solving* MAX IND $(d, 2)$*-CSP returning an optimal solution with probability at least $1 - \mathrm{e}^{-1}$ in $\mathcal{O}\left((0.5029d)^n\right)$ time.*

As was observed in the introduction, the probability of error can be made arbitrarily small; By iterating the algorithm $-\lceil \ln \kappa \rceil$ times, we get an error probability of $\kappa > 0$. Since, for fixed κ, the number of iterations is constant, this does not add to the time complexity of the algorithm.

Algorithm for Max Value CSP

The next problem under consideration is the MAX VALUE problem, defined as follows:
MAX VALUE (d, l)-CSP
Instance: (d, l)-CSP instance $\Theta = (X, D, C)$ where the domain $D = \{a_1, \ldots, a_d\} \subseteq \mathbb{R}$, together with a real vector $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$.
Output: A solution $f : X \to D$ maximising $\sum_{i=1}^{n} w_i \cdot f(x_i)$.

Note that even though the domain consists of real-valued elements, we are still dealing with a *finite* domain.

The currently best known algorithm for this problem, by Angelsmark *et al.* (2004), runs in $\mathcal{O}(d/2 \cdot \gamma_l + \epsilon)$, where $\gamma_l = \lambda(1, 2, \ldots, l)$, and $\epsilon > 0$. For $l = 2$, this is $\mathcal{O}(d/2 \cdot 1.6180 + \epsilon) \subseteq \mathcal{O}(0.8091d + \epsilon)$. The algorithm we will now present for the case with binary constraints is significantly faster.

We start by constructing an algorithm for MAX VALUE $(3,2)$-CSP with a running time in $O(1.7458^n)$, which will later form the basis for a probabilistic algorithm for MAX VALUE $(d,2)$-CSP. The algorithm actually solves a more general problem where we are given a weight function $w : X \times D \to \mathbb{R}$ and find the assignment α which maximises $\sum_{x[d]\in\alpha} w(x,d)$. The idea behind the algorithm is to use the microstructure graph and recursively branch on variables with three possible values until all variables are either assigned a value or only have two possible values left. We then transform the remaining microstructure graph to a weighted 2-SAT instance and solve this.

Before we start, we need some additional definitions: A variable having three possible domain values we call a 3-variable, and a variable having two possible domain values will be called a 2-variable. In order to analyse the algorithm we define the size of an instance as $m(\Theta) = n_2 + 2n_3$. Here, n_2 and n_3 denote the number of 2- and 3-variables in Θ, respectively. This means that the size of an instance can be decreased by 1 by either removing a 2-variable or removing one of the possible values for a 3-variable, thus turning it into a 2-variable.

For a variable $x \in X$ with possible values $\{d_1, d_2, d_3\}$ ordered so that $w(x,d_1) > w(x,d_2) > w(x,d_3)$, let $\delta(x) := (c_1, c_2, c_3)$ where $c_i = \deg_G(x[d_i])$, G being the microstructure graph. If x is a 2-variable then, similarly, we define $\delta(v) := (c_1, c_2)$. We will sometimes use expressions such as $\geq c$ or $\cdot$ (dot) in this vector, for example $\delta(v) = (\geq c, \cdot, \cdot)$. This should be read as $\delta(v) \in \{(c_1, c_2, c_3) \mid c_1 \geq c\}$. The *maximal* weight of a variable x, i.e. the domain value d for which $w(x,d)$ is maximal, will be denoted $x_{\max}$. The algorithm is presented as a series of lemmata. Applying these as shown in Algorithm 2 solves the slightly generalised MAX VALUE $(3,2)$-CSP problem.

Lemma 2 *For any instance Θ, we can find an instance Θ' with the same optimal solution as Θ, with size smaller or equal to that of Θ and to which* none *of the following cases apply.*

1. *There is a 2-variable x for which $\delta(x) = (2, \geq 1)$.*
2. *There is a variable x for which the maximal weight is unconstrained.*

Lemma 3 *If there is a variable x with $\delta(x) = (\geq 3, \geq 2)$ then we can reduce the instance with a work factor of $\lambda(3,2)$.*

Lemma 4 *If there is a variable x for which $\delta(x) = (3, \cdot, \cdot)$ then we can reduce the instance with a work factor of $\lambda(3,2)$.*

Lemma 5 *If there is a variable x with $\delta(x) = (\geq 5, \cdot, \cdot)$ then we can reduce the instance with a work factor of $\lambda(5,1)$.*

If none of Lemma 2 to Lemma 5 is applicable, any 3-variable must satisfy $\delta(x) = (5, \cdot, \cdot)$ and any 2-variable must satisfy $\delta(x) = (\geq 4, \cdot)$ or $\delta(x) = (3, 0)$.

Lemma 6 *If none of Lemma 2 to Lemma 5 is applicable then we can remove remaining 3-variable with a work factor of $\lambda(4,4,4)$.*

Algorithm 2 Algorithm for MAX VALUE $(3,2)$-CSP.

$MaxVal(G, w)$

1. If, at any time, the domain of a variable becomes empty, that branch can be pruned.
2. Apply the transformations in Lemma 2, keeping track of eliminated variables.
3. If applicable, return the maximum result of the branches described in Lemma 3 to 5
4. else, if applicable, return the maximum result of the branches described in Lemma 6
5. else return *2-SAT*$_w(G, w)$

The input to Algorithm 2 is the microstructure graph G and a weight function w, and the algorithm returns an assignment with maximal weight. Note that in order to actually get a solution to the original problem, one has to a) keep track of the variables eliminated by Lemma 2, and b) extract them from the solution returned on line 4.

Theorem 2 MAX VALUE $(3,2)$*-CSP can be solved by a deterministic algorithm in $\mathcal{O}(1.7548^n)$ time.*

If we now note that Lemma 1 holds for MAX VALUE CSP as well as MAX IND CSP, we can this time combine it with Theorem 2 and get the following theorem:

Theorem 3 *There exists an algorithm for solving* MAX VALUE $(d,2)$*-CSP which has a running time of $\mathcal{O}((0.5849d)^n)$ and finds an optimal solution with probability $1 - \mathrm{e}^{-1}$, where $\mathrm{e} = 2.7182\ldots$.*

As was the case for the MAX IND $(d,2)$-CSP algorithm earlier, the probability of success can be made arbitrarily high by iterating the algorithm.

Conclusion

We have presented algorithms for two constraint satisfaction optimisation problems; MAX IND CSP and MAX VALUE CSP. The algorithms, while both based on the microstructure of the underlying constraint satisfaction problem, are quite different. The algorithm for MAX IND can be seen as the "quick-and-dirty" approach, since little effort is spent on constructing the base algorithm—we simply apply an existing maximum independent set algorithm to the microstructure—whereas the base algorithm for MAX VALUE is a lot more intricate and utilises properties of the microstructure a lot more efficiently than a general maximum independent set algorithm could. It would of course be interesting to see how much improvement we would get by careful case analysis for the MAX IND algorithm as well.

As was mentioned in the introduction, the algorithm for MAX VALUE is more general than strictly necessary, and it can be applied to any problem where, given a CSP instance together with a weight function, the goal is to maximise the sum of these weights. We have not yet fully explored the possibilities here, but it is worth noting that our approach is quite different from the standard branch-and-bound one

when it comes to, say, INTEGER PROGRAMMING. Consequently, a future direction would be to compare this and other approaches.

To summarise, we have shown that by exploiting the microstructure graph of a CSP we can either a) quickly develop algorithms for optimisation problems (as we did for MAX IND), or b) carefully analyse it in order to get more efficient algorithms (as we did for MAX VALUE.) This would suggest that the microstructure graphs deserve further study.

Acknowledgments

The authors would like to thank Peter Jonsson for helpful comments.

Proofs

In order to keep the discussion flowing, we present the proofs of the theorems in this paper here.

Corollary 1 (Proof). The microstructure graph contains dn vertices, and using the $\mathcal{O}\left(1.2025^{|V(G)|}\right)$ time maximum independent set algorithm by Robson (2001), we can find a solution in $\mathcal{O}\left(1.2025^{dn}\right)$ time.□

Lemma 1 (Proof). For each variable in (X, D, C), we know it can be assigned at most k out of the d possible values. Thus, we can modify the constraints so that every variable picks its value from the set $\{1, \ldots k\}$. This transformation can obviously be done in polynomial time, and the resulting instance is an instance of MAX IND (k, l)-CSP, which can be solved in $\mathcal{O}\left(c^n\right)$ time.□

Theorem 1 (Proof). Combining Lemma 1 with the fact that we can solve MAX IND $(k, 2)$-CSP in $\mathcal{O}\left(c^{kn}\right)$ using microstructures, we can solve a MAX IND $(d, 2)$-CSP instance where the variables are restricted to domains of size k in $\mathcal{O}\left(c^{kn}\right)$ time.

Let $S_k = \{D_1 \times D_2 \times \ldots \times D_n \subseteq D^n \text{ s.t. } |D_i| = k, 1 \leq i \leq n\}$. For $\Delta \in S_k$, define Θ_Δ to be the instance Θ where each variable x_i has been restricted to take its value from the set D_i—i.e. for each $x_i \in X$ we add the constraint $(x_i; D_i)$. No additional solution can be introduced by restricting the domains in this way, and a solution f which is optimal for Θ, is still a solution (and optimal) to any Θ_Δ for which $f(x_i) \in \Delta_i, i \in \{1, \ldots, n\}$.

For a randomly chosen $\Delta \in S_k$, the probability that an optimal solution to Θ is still in Θ_Δ is at least $(k/d)^n$. It follows that the probability of not finding an optimal solution in t iterations is at most $(1 - (k/d)^n)^t < \mathrm{e}^{-t(k/d)^n}$. Therefore, by repeatedly selecting $\Delta \in S_k$ at random and solving the induced MAX IND $(k, 2)$-CSP problem, we can reduce the probability of error to e^{-1} in $(d/k)^n$ iterations. The resulting algorithm, Algorithm 1 has a running time of $\mathcal{O}\left((d/k \cdot c^k)^n\right)$.

Now, let $g(x) = c^x/x$. The function g is convex and assumes its minimum at $x' = (\ln c)^{-1}$. Thus, for a given c, finding the minimum of $g(\lceil x' \rceil)$ and $g(\lfloor x' \rfloor)$ determines the best choice of k.□

Lemma 2 (Proof). The transformation in Fig. 1 takes care of the first case, removing one 2-variable (and therefore decreasing the size of the instance). For the second case, we can simply assign the maximal weight to the variable, leaving us with a smaller instance.□

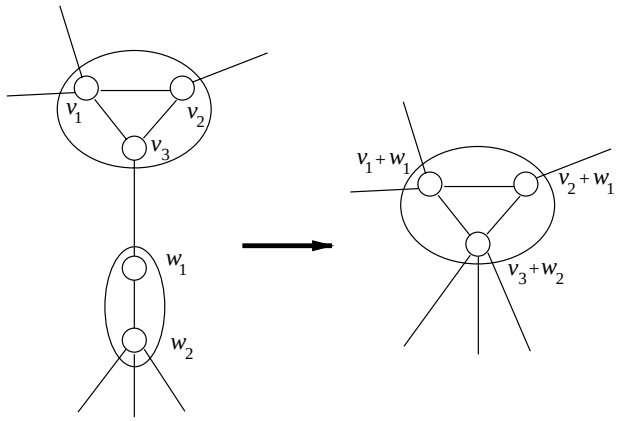

Figure 1: The transformation done in Lemma 2. (Note that v_i, w_j are the *weights* of the vertices.)

Lemma 3 (Proof). We branch on the two possible values of x and propagate the chosen value to the neighbours of x. In one of the branches, the size will decrease by at least 3 and in the other by at least 2.□

Lemma 4 (Proof). In one of the branches, we choose $x = x_{\max}$ and propagate this value, decreasing the size by at least 3. In the other branch, choosing $x \neq x_{\max}$ implies that the value of its exterior neighbour must be chosen in order to force a non-maximal weight to be chosen in x. Therefore, in this branch, the size decreases by at least 2.□

Lemma 5 (Proof). Choosing $x = x_{\max}$ decreases the size of the instance by at least 5. In the other branch, we choose $x \neq x_{\max}$, turning a 3-variable into a 2-variable and thereby decreasing the size by 1.□

Lemma 6 (Proof). Let x_1 be a 3-variable and d_1 its maximal value, with neighbours x_2 and x_3 as in Fig. 2.

If x_2 is a 2-variable and $\delta(x_2) = (3, 0)$ (see Fig. 2a) then we can let one of the branches be $x_1 = d_1$ and the other $x_1 \neq d_1$. This makes $\delta(x_2) = (2, 0)$ in the latter branch, and x_2 can be removed by Lemma 2 which means we can decrease the size by 2 in this branch, giving a work factor of $\lambda(4, 2) < \lambda(4, 4, 4)$. Otherwise if $x_3[d_3]$ has only three neighbours, x_3 must be a 3-variable, which implies that $x_3[d_3]$ can not be maximally weighted. If this holds for 0 or 1 of the neighbours of x_1 (Fig. 2b), we can branch on $x_1 = d_1$, $x_3 = d_3$ and $\{x_2 = d_2, x_3 \neq d_3\}$. In this case, we decrease the size by 4 in each branch. If both of $x_2[d_2]$ and $x_3[d_3]$ are of degree 3 (Fig. 2c) and it is not possible to choose an x_1 without this property, then for every 3-variable remaining, the maximal weighted value can be assigned without branching at all.□

Theorem 2 (Proof). We claim that Algorithm 2 correctly solves MAX VALUE $(3, 2)$-CSP.

First we show the correctness of the algorithm. Lemma 2 does not remove any solutions to the problem, it merely reduces the number of vertices in the microstructure. Lemma 3

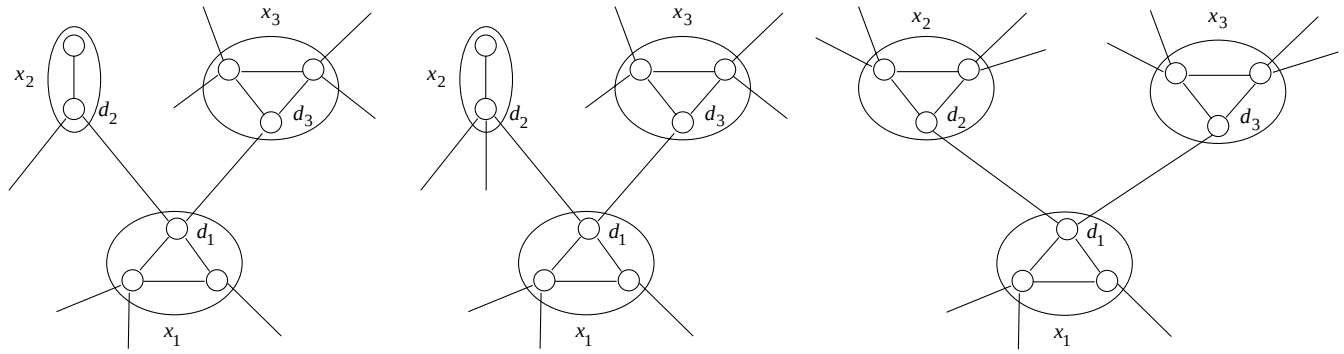

Figure 2: The 3 cases discussed in Lemma 6. (Note that x_i denotes a variable, while d_j denotes a value.)

branches on both possible values for the chosen variable, while Lemmata 4 and 5 try all three possible assignments, as is the case for Lemma 6. This, together with the proof of correctness of the *2-SAT$_w$* algorithm by Dahllöf *et al.* (2004), shows the correctness of the algorithm.

Now, apart from the call to *2-SAT$_w$*, the highest work factor found in the algorithm is $\lambda(3,2) = \lambda(1,5) \approx 1.3247$. Recall that we measure the size of Θ by $m(\Theta) = n_2 + 2n_3$ which, for $(3,2)$-CSP, is $2n$, where n is the number of variables in Θ. If we can solve weighted 2-SAT in $\mathcal{O}(c^n)$, then the entire algorithm will run in $\mathcal{O}\left(\max(1.3247, c)^{2n}\right)$. Using the construction with weighted microstructures mentioned earlier, a weight function $w(x_i, d) = w_i \cdot d$, together with the algorithm for weighted 2-SAT by Dahllöf *et al.* (2004), we get $c \approx 1.2561$, and the result follows.□

Theorem 3 (Proof). Similar to the proof of Theorem 2. For each $\Delta \in S_3$, we again define Θ_Δ to be the instance Θ where each variable x_i has been restricted to take its value from the set D_i, with $|D_i| = 3$. Theorem 2 together with Lemma 1 tells us that we can solve this instance in $\mathcal{O}(1.7548^n)$.

For a randomly chosen $\Delta \in S$, the probability that an optimal solution to Θ is still in Θ_Δ is at least $(3/d)^n$. It follows that the probability of not finding an optimal solution in t iterations is at most $(1-(3/d)^n)^t < \mathrm{e}^{-t(3/d)^n}$. Therefore, by repeatedly selecting $\Delta \in S$ at random and solving the induced MAX VALUE $(3,2)$-CSP problem, we can reduce the probability of error to e^{-1} in $(d/3)^n$ iterations. Consequently, we get a running time of $\mathcal{O}((d/3)^n \cdot 1.7548^n) \approx \mathcal{O}((0.5849d)^n)$.□

References

Angelsmark, O.; Jonsson, P.; and Thapper, J. 2004. Two methods for constructing new CSP algorithms from old. Unpublished manuscript. Available for download at http://www.ida.liu.se/~olaan/papers/twomethods.ps.

Dahllöf, V.; Jonsson, P.; and Wahlström, M. 2004. On counting models for 2SAT and 3SAT formulae. To appear in *Theoretical Computer Science.* DOI:10.1016/j.tcs.2004.10.037.

Eppstein, D. 2001. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA-2001)*, 329–337.

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

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Company.

Jeavons, P. 1998. On the algebraic structure of combinatorial problems. *Theoretical Computer Science* 200:185–204.

Jégou, P. 1993. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *Proceedings of the 11th (US) National Conference on Artificial Intelligence (AAAI-93)*, 731–736. Washington DC, USA: American Association for Artificial Intelligence (AAAI).

Jonsson, P., and Liberatore, P. 1999. On the complexity of finding satisfiable subinstances in constraint satisfaction. Technical Report TR99-38, Electronic Colloquium on Computational Complexity.

Jonsson, P. 1998. Near-optimal nonapproximability results for some NPO PB-complete problems. *Information Processing Letters* 68(5):249–253.

Kullmann, O. 1999. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science* 223(1–2):1–72.

Kumar, V. 1992. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine* 13(1):32–44.

Robson, M. 2001. Finding a maximum independent set in time $O(2^{n/4})$. Technical report, LaBRI, Université Bordeaux I.