New Algorithms for the Maximum Hamming Distance Problem

Ola Angelsmark[1]* and Johan Thapper[2]**

[1] olaan@ida.liu.se
Department of Computer and Information Science
Linköpings Universitet
S-581 83 Linköping, Sweden
[2] jotha@mai.liu.se
Department of Mathematics
Linköpings Universitet
S-581 83 Linköping, Sweden

Abstract. We study the problem of finding two solutions to a constraint satisfaction problem which differ on the assignment of as many variables as possible – the MAX HAMMING DISTANCE problem for CSPs – a problem which can be seen as a domain independent way of quantifying "ignorance." The first algorithm we present is an $\mathcal{O}(1.7335^n)$ microstructure based algorithm for MAX HAMMING DISTANCE $(2,2)$-CSP, which is an improvement over the previously best known algorithm for this problem, which has a running time of $\mathcal{O}(1.8409^n)$. We also give algorithms for MAX HAMMING DISTANCE $(2,l)$-CSP and the general MAX HAMMING DISTANCE (d,l)-CSP, which are enumerative in nature. The main result for these problems is that if we can solve $(2,l)$-CSP (i.e. l-SAT) in $\mathcal{O}(a^n)$ and (d,l)-CSP in $\mathcal{O}(b^n)$, then the corresponding Max Hamming problems can be solved in $\mathcal{O}((2a)^n)$ and $\mathcal{O}(b^n(1+b)^n)$, respectively.

1 Introduction

Ever since they were introduced in the 1970's [13], constraint satisfaction problems (CSPs) have proven to be an important tool in the computer scientist's tool box. CSPs provide a natural and efficient way of formulating problems in a wide variety of fields, cf [9, 12], and, because of this, they have received a lot of attention by the research community.

In its most basic form, a CSP consists of a collection of variables taking values from some domain, and a collection of constraints restricting the values different variables can simultaneously assume. Now the question is: Does there exist an assignment of values to these variables which does not violate any of the constraints? While this is certainly the most thoroughly studied problem for

* Supported in part by the National Graduate School in Computer Science, Sweden, and in part by the *Swedish Research Council* (VR), grant 621-2002-4126.
** Supported by the *Programme for Interdisciplinary Mathematics*, Department of Mathematics, Linköpings Universitet.

CSPs, there are a number of alternative, equally interesting, questions one can ask about a CSP.

The question we will study in this paper asks us to find two solutions that are as far away from each other as possible; i.e. we want to find two satisfying assignments that disagree on the values for as many variables as possible. This is known as the MAX HAMMING DISTANCE problem, and was first introduced in Crescenzi & Rossi [3], where it was suggested as a domain independent measure of ignorance, quantifying how much we do not know of the world.

We present three different algorithms in this paper. The first one is a microstructure based algorithm for the special case when the domains have size two and we have binary constraints, denoted MAX HAMMING DISTANCE $(2,2)$-CSP. (We will exclusively consider CSPs over finite domains, denoted (d,l)-CSP, where d is the domain size and l the arity of the constraints.) Even in this restricted form, the problem is NP-complete. In the microstructure graph of a CSP [10], a vertex corresponds to an assignment of a value to a variable in the original problem (see Section 2 for definitions.) The algorithm exploits this by searching for a set of vertices where each vertex either does not have an edge to any other vertex – and thus can be interpreted as an assignment – or is part of a connected component with 2 or 4 vertices. Each vertex (i.e. assignment) in this set is then given a weight, and the original instance together with these weights is given to a weighted 2-SAT solver. This algorithm returns a solution with maximum weight W, and we can then construct a solution which differs on W variables.

By using the weighted 2-SAT algorithm from [4], we arrive at a running time of $\mathcal{O}(1.7335^n)$, where n is the number of variables in the problem. This is an improvement over the MAX HAMMING DISTANCE $(2,2)$-CSP algorithm presented in [1], which runs in $\mathcal{O}(1.8409^n)$.

When we allow domains with more than 2 elements, or constraints with arity higher than 2, it turns out that microstructures are not as successful, and, consequently, the algorithms we present for these cases are quite different. Intuitively, the algorithm for MAX HAMMING DISTANCE (d,l)-CSP works as follows:

1. Pick a subset of the variables which should assume different values in the two solutions, duplicate and rename them and the constraints they are involved in.
2. Add a constraint for each of these new variables, preventing it from assuming the same value as the one it is a copy of.
3. Solve this new, larger, instance.

Starting with assuming all variables are different in the two solutions, and then working downwards, the algorithm will, by trying out the different possible subsets of variables, arrive at a pair of solutions with maximum hamming distance. Given that we can solve the (d,l)-CSP in the last step in time $\mathcal{O}(a^n)$, the entire algorithm will have a running time of $\mathcal{O}(a^n(1+a)^n)$.

The final algorithm is for the case when the domain has two elements and the constraints have arity l, MAX HAMMING DISTANCE $(2,l)$-CSP. Here, we

note that since there are only two possible choices of values for a variable, it is unnecessary to duplicate the variables that should take different values – instead, only the constraints they are involved in are duplicated, and then any occurrence of a variable which should assume different values in the two solutions is replaced by its negation in these constraints. The resulting algorithm will have a running time of $\mathcal{O}((2a)^n)$, where $\mathcal{O}(a^n)$ is the time needed to solve the $(2,l)$-CSP problem in each step.

1.1 Overview of the paper

Section 2 contains most of the definitions we will need in the discussion. For convenience, it has been split into three parts, where Section 2.1 contains the definitions related to CSPs, Section 2.2 the graph and microstructure definitions, while Section 2.3 formally defines the problem we will be studying, MAX HAMMING DISTANCE. The algorithm for MAX HAMMING DISTANCE $(2,2)$-CSP, together with its analysis, is presented in Section 3, while Section 4 contains the algorithms for MAX HAMMING DISTANCE (d,l)-CSP and MAX HAMMING DISTANCE $(2,l)$-CSP.

2 Preliminaries

This section is divided into three parts in order simplify the search for a particular definition. In Section 2.1 we have the definitions related to constraint satisfaction problems, while Section 2.2 is devoted to graphs and the microstructure of CSPs. Finally, in Section 2.3, we define the problem we will be discussing in this paper; MAX HAMMING DISTANCE (d,l)-CSP. Note that Section 3 contains additional definitions specific to that part of the paper.

2.1 Constraint satisfaction problems

A *(d,l)-constraint satisfaction problem ((d,l)-CSP)* is a triple (X,D,C) where

- X is a finite set of variables,
- D a finite set of domain values, with $|D| = d$, and
- C is 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 instances 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 (d,l)-CSP, the basic computational problem is to decide whether it has a solution or not – to determine if it is *satisfiable*.

The special case when $d = 2$ and we have binary constraints, i.e. $(2,2)$-CSP, will often be viewed as 2-SAT formulae. 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*. Determining whether a 2-SAT formula is satisfiable can be done in polynomial time [2], while, in contrast, the more general l-SAT (i.e, the clauses consist of at most l literals) is known to be NP-complete for $l \geq 3$ [7].

Definition 1 ([4]). *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.

In [4], an algorithm for counting the number of maximum weighted solutions to 2-SAT instances is presented which has a running time of $\mathcal{O}(1.2561^n)$, and it can easily be modified to return one of the solutions.

2.2 Graphs and microstructures

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 *neighbourhood* of a vertex $v \in V(G)$ is the set of all vertices adjacent to v, excluding v itself, and is denoted $N_G(v)$, $N_G(v) := \{u \in V(G) \mid (v,w) \in E(G)\}$. If, by following the edges of the graph, we can get from a vertex v to v', then v' is *reachable* from v. The *connected components* of a graph are the equivalence classes of vertices under the "is reachable from" relation.

Definition 2 ([10]). *Given a binary CSP $\Theta = (X, D, C)$, the microstructure of Θ is an undirected graph G, defined as follows:*

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

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

For convenience, we will work exclusively with the complement of the graph in Def. 2. The complement of a (microstructure) graph G is a graph containing exactly those edges which are not present in G, i.e. a graph with edge set $\{(v,u) \mid v \neq u \wedge (v,u) \notin E(G)\}$.

A variable with domain size d will in the microstructure graph be a clique of size d. When the domain has two elements and we have a clique of size 2, we will use $x[i]$ to denote an arbitrary value for x, and $x[1-i]$ to denote the other possible value.

algorithm $MH1(\alpha, G, \Theta)$

1. **if** $\delta(x) \in \{(3,1),(2,2),(2,1),(1,1)\}$ for all variables x in G **then**
2. **return** $MH2(\alpha, G, \Theta)$
3. **end if**
4. Choose a variable x in G with $\delta(x) \in \{(\geq 3, \geq 2),(\geq 4,1)\}$
5. $(\alpha_0, \beta_0) = MH1(\alpha \cup \{x[0]\}, G - N_G(x[0]) - \{x[0]\}, \Theta)$
6. $(\alpha_1, \beta_1) = MH1(\alpha \cup \{x[1]\}, G - N_G(x[1]) - \{x[1]\}, \Theta)$
7. **return** $(\alpha_i, \beta_i), i \in \{0,1\}$ maximising $d_H(\alpha_i, \beta_i)$

Fig. 1. The main algorithm for MAX HAMMING DISTANCE $(2,2)$-CSP.

2.3 Hamming distance of CSPs

The algorithms we present in this paper are all designed to solve different variants of the MAX HAMMING DISTANCE problem for constraint satisfaction problems [3]. Since our algorithms are not limited to problems with two valued domains, the following definition differs somewhat from the one given in [3]:

Definition 3. *Given a set of variables X over finite domains, the* Hamming distance *between a pair f_1 and f_2 of assignments of values to the variables in X, denoted $d_H(f_1, f_2)$, is the number of variables on which f_1 and f_2 disagree.*

For example, consider the 2-SAT formula $(x \vee y) \wedge (\neg x \vee z)$, and the two assignments $f_1 = \{x \mapsto 0, y \mapsto 1, z \mapsto 0\}$, $f_2 = \{x \mapsto 1, y \mapsto 1, z \mapsto 1\}$. Clearly, both f_1 and f_2 satisfy the formula, and their Hamming distance is 2, since they disagree on the values for x and z.

The following formalises the problem:

Definition 4 (Maximum Hamming Distance of (d,l)-CSPs). *Let $\Theta = (X, D, C)$ be an instance of (d,l)-CSP. The* MAX HAMMING DISTANCE *(d,l)-CSP problem is to find two satisfying assignments f and g to Θ which maximises $d_H(f,g)$.*

A naïve enumeration algorithm for this problem would have a time complexity of $\mathcal{O}\left(d^{2n}\right)$. In the following sections we will present ways to significantly improve this running time.

3 Algorithm for Max Hamming Distance $(2,2)$-CSP

In this section we will discuss and analyse our algorithm for MAX HAMMING DISTANCE $(2,2)$-CSP. Since the formulae for the time complexity of the algorithm can be rather lengthy, the final step, that of calling a weighted 2-SAT solver for every leaf in the search tree, has been left out unless otherwise noted. Consequently, when, in the discussion, we say "$\mathcal{O}(a^n)$," this should be read as

algorithm *MH2*(α, G, Θ)

1. **if** $\delta(x) \in \{(2,1),(1,1)\}$ for all variables x in G **then**
2. **return** *MH3*(α, G, Θ)
3. **end if**
4. **if** G contains a cycle **then**
5. **if** all variables x has $\delta(x) = (2,2)$ in a cycle **then**
6. Choose x in this cycle
7. **else if** there is a variable z with $\delta(z) = (2,2)$ in a cycle **then**
8. Choose x in a cycle s.t $\delta(x) = (3,1)$ and $x[i]$ has a neighbour y with $\delta(y) = (2,2)$
9. **else**
10. Choose x with $\delta(x) = (3,1)$ in a cycle
11. **end if**
12. **else** *% G is cycle-free*
13. Choose x which is two variables from the end of a chain, if possible, otherwise, choose x one variable from the end of a chain
14. **end if**
15. $(\alpha_0, \beta_0) = MH1(\alpha \cup \{x[0]\}, G - N_G(x[0]) - \{x[0]\}, \Theta)$
16. $(\alpha_1, \beta_1) = MH1(\alpha \cup \{x[1]\}, G - N_G(x[1]) - \{x[1]\}, \Theta)$
17. **return** $(\alpha_i, \beta_i), i \in \{0,1\}$ maximising $d_H(\alpha_i, \beta_i)$

Fig. 2. The helper function *MH2*.

"$\mathcal{O}(a^n \cdot b^n)$ where $\mathcal{O}(b^n)$ is the running time of a weighted 2-SAT solver." Furthermore, we will omit polynomial factors in the time complexities.

Before we start the discussion of the algorithms, we will need some additional definitions.

The *degree* of a vertex v in a graph, usually denoted $\deg(v)$, is the size of its neighbourhood, i.e. $|N_G(v)|$. However, we are not really interested in the degree of a single vertex, but rather in the degrees of the two vertices that make up a variable. Thus let $\Theta = (X, D, C)$ be a $(2,2)$-CSP and, for $x \in X$, define the *variable degree* $\delta(x)$ as a tuple $(\deg(x[i]), \deg(x[1-i]))$, where $x[i]$ is the vertex with highest degree. If we are interested in variables with degrees higher than a certain number, we write $\delta(x) = (\geq i, \geq j)$.

In the analysis of the algorithm in this section, we will often encounter recursions on the form

$$T(n) = \sum_{i=0}^{k} T(n - r_i) + p(n)$$

where $p(n)$ is a polynomial in n and $r_i \in \mathbb{Z}^+$. These equations satisfy $T(n) \in \mathcal{O}(\tau(r_1, r_2, \ldots, r_k)^n)$, where $\tau(r_1, r_2, \ldots, r_k)$ is the largest real-valued solution to the equation $1 - \sum_{i=1}^{k} x^{-r_i} = 0$ (see Kullman [11].) Note that this bound depends on neither $p(n)$ nor the boundary conditions $T(1) = b_1, \ldots, T(k) = b_k$.

With that in mind, we are now ready to discuss the algorithm.

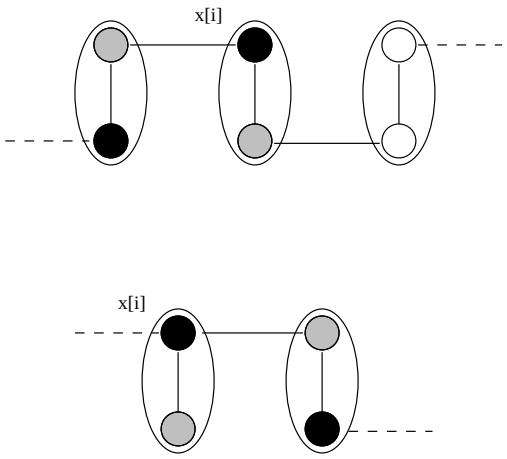

Fig. 3. Branching on $x[i]$ will remove the shaded values and force the black values.

The main algorithm, *MH1*, takes as input a partial assignment α, a microstructure graph G, and the original problem instance Θ. If every variable in the microstructure is involved in less than 3 constraints, the helper function $MH2$ is called. In the graph, this translates to every variable x having $\delta(x)$ in the set $\{(3,1),(2,2),(2,1),(1,1)\}$. Otherwise, a variable without this property is chosen, and the algorithm branches on the two possible values. We note that for $\delta(x) = (3,2)$, there will be at least 3 variables less in one branch and 2 variables less in the second branch, and for $\delta(x) = (4,1)$, there are at least 4 and 1 variables less, respectively. Thus the time complexity is described by the following two recurrences:

$$T_{(3,2)}(n) = T(n-3) + T(n-2) + p(n)$$

and

$$T_{(4,1)}(n) = T(n-4) + T(n-1) + p(n)$$

where $p(n)$ is a polynomial. Using the method by Kullman [11] described earlier, the two cases have running times of $T_{(3,2)} \in \mathcal{O}(\tau(3,2)^n)$ and $T_{(4,1)} \in \mathcal{O}(\tau(4,1)^n)$. Of these two, the latter grows faster, and will dominate the time complexity.

The first helper function, *MH2*, found in Fig. 2, takes over when no variable is involved in more than 2 constraints. Apart from lines 1 to 3, which we will come back to later, the algorithm starts with checking for cycles. If there is a cycle in the graph, we need to break it, and this is done on lines 4 to 11. First of all, if there is a cycle where every variable has a degree of $(2,2)$, then selecting one value for a variable in this cycle will propagate through the entire cycle, as is shown in the upper portion of Fig. 3. On line 8, by choosing a variable x with

algorithm *MH3*(α, G, Θ)

1. Let $\bar{w}$ be a vector of weights, initially all set to 0
2. **for each** $x[i] \in \alpha$ **do**
3. add weight $\bar{w}(x[1-i]) := 1$
4. **for each** connected component of G **do**
5. Add weights to $\bar{w}$, as shown in Fig. 5.
6. $(\beta, W) :=$ *2-SAT*$_w(\Theta, \bar{w})$
7. **for each** variable x in G **do**
8. **if** $x[i]$ in β **then**
9. If possible, add $x[1-i]$ to α, otherwise, add $x[i]$.
10. **end if**
11. **end for**
12. **return** (α, β)

Fig. 4. The helper function *MH3*.

$\delta(x) = (3,1)$ with a neighbour y with $\delta(y) = (2,2)$, one of the values for x will propagate to y. (See lower part of Fig. 3.) Consequently, 4 variables are removed in one branch, and one in the other, giving a running time of $\mathcal{O}(\tau(4,1)^n)$ for this case. The obvious exception to this case is when the cycle contains only 3 variables, as is shown in the left part of Fig. 6. Note that the coloured vertex $x[i]$ is the only possible choice – the other assignment would lead to an inconsistency.

Now if every variable x in the cycle has $\delta(x) = (3,1)$, we get a number of different possibilities, but before we discuss them, we need to make some observations. The first one is that once a variable has no neighbours, see Fig. 5, we can choose any of the two values for it. Additionally, we get a similar situation when a component has been reduced to two variables with one edge between then, a 'hurdle.' This means that when a component of $(3,1)$ variables has at most 3 variables, as in Fig. 6, to the right, when choosing such a variable, in effect, the entire component is removed from the problem and need no longer be considered – in one case we get a unique assignment for the remaining (black) vertices in the other case we get a hurdle. If there are no cycles in the component, e.g we have a 'comb-like' structure (see Fig. 7), then choosing any of the three variables to branch on will, again, remove the entire component, giving a running time of $\mathcal{O}(\tau(3,3)^n)$. This also holds for cycle-free components of size 4 and 5. When there are more than 5 variables in the component, by choosing a variable which is two variables removed from the end of the comb (marked in Fig. 7), the chain is broken and we remove 3 variables in one branch and 4 in the other. As was seen in the case for cycles where all variables have degree $(2,2)$, the number of removed variables increases if a neighbour of the branching variable has this property. Consequently, we will focus on the combs and merely note that the time complexity will not be worse if we have more variables with degree $(2,2)$.

Getting back to discussing cycles; When we reach line 10 of algorithm *MH2*, every cycle consists exclusively of variables with degree $(3,1)$, and since no vertex in the graph has degree higher than 3, there can be at most one cycle in a

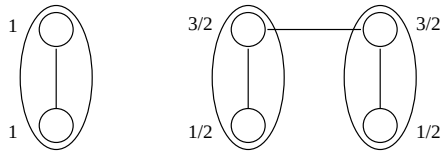

Fig. 5. Variables with no or exactly one neighbour, and the weights they are given by algorithm *MH3*.

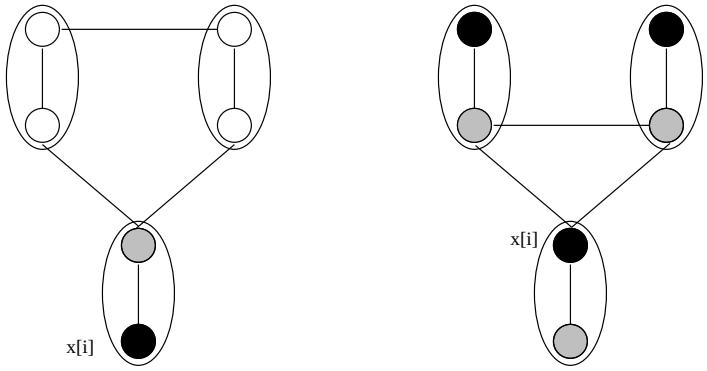

Fig. 6. The case when a component is a cycle with 3 variables.

component. The case with cycles containing 3 variables was discussed earlier, and for the case with 4 variables we get one branch where the entire component is removed, and one where we get a comb with 3 variables, which can be removed in its entirety when we branch. There can be no more than $n/4$ cycles with 4 variables in the graph at this point. For each of these cycles, we choose one variable to branch on, and in one branch the entire component is removed, while in the other, we get a component with 3 variables. Since we want to look at all of these cycles, and both branches, this is equivalent to selecting k cycles where we remove the entire component, and then examine the remaining $n/4-k$ components. In other words, it will require

$$\sum_{k=0}^{n/4} \binom{n/4}{k} \left(1^k \cdot \tau(3,3)^{3(n/4-k)}\right)$$

steps to examine all the cycles. Using the binomial theorem, this can be simplified to $(1+\tau(3,3)^3)^{n/4}$.

For cycles with 5 variables, the situation is similar, but for 6 we no longer remove the entire component in one of the branches. Instead, we get one branch with 5 variables, and one with 3, which, using the same reasoning as above, gives

$$\sum_{k=0}^{n/6} \binom{n/4}{k} \left(\tau(3,3)^{3k} \cdot \tau(5,5)^{5(n/6-k)}\right) = (\tau(3,3)^3 + \tau(5,5)^5)^{n/6}.$$

Similarly, for cycles of length 7, we get $(\tau(4,4)^4 + \tau(3,4)^6)^{n/7}$. In general, if we have cycles of length c, one branch will have one variable less, and the other three variables less, giving the following general running time:

$$\sum_{k=0}^{n/c} \binom{n/c}{k} \left(\tau(3,4)^{(c-3)k} \cdot \tau(3,4)^{(c-1)(n/c-k)}\right) = \left(\tau(3,4)^{c-1} + \tau(3,4)^{c-3}\right)^{n/c} <$$

$$< (2\tau(3,4)^c)^{n/c} = (2^{1/c}\tau(3,4))^n$$

Finally, when algorithm *MH3* (see Fig. 4) is called, the graph G only contains variables involved in zero or one constraint, i.e. every variable will be of one of the forms found in Fig. 5. The weights shown in the figure is now added to the corresponding assignments in the original problem, Θ, and the resulting weighted 2-SAT problem is given to a *2-SAT*$_w$ solver. If the solution β returned by the solver has weight W, this means that we can add assignments (i.e. vertices) to α to create a solution which differs from β on W assignments. First of all, since all assignments in α are given weight 0, if any of these are chosen, they will not add anything to the distance, while the other possible value for all these variables will add one to the distance (and are consequently given a weight of 1 on line 3.) For the free variables in G, i.e., all vertices x with $\delta(x) = (1,1)$, we can choose freely which value they should assume, and thus we can always find an assignment which adds one to the distance from β by choosing the other value for α. The remaining components then consist of pairs of variables with one edge between them, i.e. hurdles. If β contains both assignments with weight $1/2$, then obviously, we have to add one of them to α, since not both assignments with weight $3/2$ are allowed simultaneously – and thus we get a distance of 1, which is the sum of the weights in β. On the other hand, if β contains one $3/2$ and one $1/2$ assignments, then we can choose the opposing value for both of these and get a distance of 2. Consequently, the pair returned on line 12 will have a Hamming distance equal to the weight of β, and with α and G given, no pair with greater Hamming distance can exist.

Except for the call to *2-SAT*$_w$ on line 6, every step of algorithm *MH3* can be carried out in polynomial time, thus the time complexity is fully determined by that of the *2-SAT*$_w$ algorithm.

To summarise this section, we state the following theorem:

Theorem 1. *Algorithm MH correctly solves* Max Hamming Distance $(2,2)$-*CSP and has a running time of* $\mathcal{O}\left((a \cdot 1.3803)^n\right)$*, where* n *is the number of variables in the problem, and* $\mathcal{O}\left(a^n\right)$ *is the time complexity of solving a weighted 2-SAT problem.*

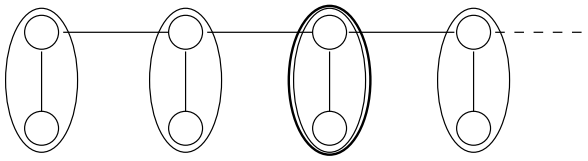

Fig. 7. Choosing a variable in a comb with more than 4 variables.

Proof. The correctness follows from the previous discussion, and among the steps in the algorithm, $\mathcal{O}(\tau(4,1)^n) \in \mathcal{O}(1.3803^n)$ dominates. Since the 2-SAT$_w$ algorithm is called for every leaf in the search tree, we get a total time complexity of $\mathcal{O}((a \cdot 1.3803)^n)$. □

Corollary 1. MAX HAMMING DISTANCE $(2,2)$-*CSP can be solved in* $\mathcal{O}(1.7338^n)$.

Proof. Dahllöf et al. [4] presents an algorithm for solving weighted 2-SAT in $\mathcal{O}(1.2561^n)$, and this together with Theorem 1 gives the result. □

4 Algorithm for Max Hamming Distance (d, l)-CSP

For problems where the arity of the relations is greater than 2, microstructures are not as convenient and we have to find a different approach.

Let us first consider the following problem: Given a CSP instance $\Theta = (X, D, C)$, can we find a pair of solutions with Hamming distance equal to k? One obvious way of doing this is the following:

1. Pick a subset Y of X with $|Y| = k$
2. Create a copy $\Theta' = (X', D, C')$ of Θ over variables X'
3. For each $x \in Y$, add the constraint $x \neq x'$ to C', and
4. for each $x \notin Y$, add the constraint $x = x'$ to C'.
5. If Θ' is satisfiable with solution s
   - Solve the instance $(X \cup X', D, C')$, giving a satisfying assignment s.
   - For each $x \in X$, add $s(x)$ to α
   - For each $x' \in X'$, add $s(x')$ to β
   - Return (α, β)

There are 2^n ways to choose Y on the first line, so if we can solve the satisfiability problem for Θ in time $\mathcal{O}(h(n))$, then, since the number of variables in Θ' is twice that of Θ, we can find a pair of solutions with maximum Hamming distance in $\mathcal{O}(2^n h(2n))$. For example, since 2-SAT can be solved in linear time, we would, using this approach, get a running time of $\mathcal{O}(2^n)$ for the MAX HAMMING DISTANCE $(2,2)$-CSP. This does give a slower running time than the algorithm we presented in the previous section, but it can be applied to CSP instances with domain size and constraint arity greater than 2.

algorithm MAX HAMMING DISTANCE (d,l)-CSP $(\Theta = (X, D, C))$

```
1.  for k := |X| down to 0 do
2.      for each χ ⊆ X, |χ| = k do
3.          Let Θ' = (X', D, C') be a copy of Θ
4.          Let γ ⊆ C be all constraints involving variables from χ
5.          Create γ' by exchanging all variables not in χ with
            their counterparts from X'
6.          C' := C' ∪ γ'
7.          for each x ∈ χ do
8.              C' := C' ∪ {x ≠ x'}
9.          if (X ∪ X', D, C') is satisfiable then
10.             Let α, β be the two assignments found in a solution to Θ'
11.             return (α, β)
12.         end if
13.     end for
14. end for
```

Fig. 8. Algorithm for MAX HAMMING DISTANCE (d,l)-CSP.

Now note that in the algorithm sketch, it is actually unnecessary to make a copy of all the variables. Having selected k variables that should be different in the two solutions, we only need to make copies of those, leaving the remaining $n-k$ variables unchanged. Thus, we get the algorithm for MAX HAMMING DISTANCE (d,l)-CSP given in Fig. 8.

Theorem 2. *If we can solve (d,l)-CSP in $\mathcal{O}(a^n)$, then there exists an algorithm for* MAX HAMMING DISTANCE (d,l)*-CSP which runs in $\mathcal{O}((a(1+a))^n)$.*

Proof. In the algorithm presented in Fig. 8, the instance Θ' will contain $2n-k$ variables, and there are $\binom{n}{k}$ ways of choosing χ. Consequently, given that we can solve (d,l)-CSP in $\mathcal{O}(a^n)$, the algorithm has a total running time of

$$\mathcal{O}\left(\sum_{k=0}^{n}\binom{n}{k}a^{2n-k}\right) = \mathcal{O}\left(a^n\sum_{k=0}^{n}\binom{n}{k}a^{n-k}\right) = \mathcal{O}\left(a^n(1+a)^n\right)$$

and the result follows. □

Using previous results on algorithms for (d,l)-CSPs, we get the following corollary.

Corollary 2. *For $\epsilon > 0$, there exists an algorithm for solving* MAX HAMMING DISTANCE (d,l)*-CSP in*

$$\mathcal{O}\left(\left(\left(d-\frac{d}{l}\right)^2 + d - \frac{d}{l} + \epsilon\right)^n\right)$$

algorithm MAX HAMMING DISTANCE $(2,l)$-CSP (F)

1. **for** $k := |\mathrm{Var}(F)|$ **down to** 0 **do**
2. **for each** $\chi \subseteq \mathrm{Var}(F)$ with $|\chi| = k$ **do**
3. Let γ be all the clauses of F containing variables from χ
4. Create $\bar{\gamma}$ by negating all occurrences of a variable $x \in \chi$ in γ
5. Let $F^+ := F \cup \{\bar{\gamma}\}$
6. **if** F^+ is satisfiable **then**
7. Let α, β be the two found in a solution to F^+.
8. **return** (α, β)
9. **end if**
10. **end for**
11. **end for**

Fig. 9. The MAX HAMMING DISTANCE $(2,l)$-CSP algorithm.

Proof. Follows from Theorem 2 together with the $\mathcal{O}\left((d - d/l + \epsilon)^n\right)$ (d,l)-CSP algorithm from [14]. □

There exist a number of algorithms for special cases of (d,l)-CSPs, especially for binary constraints. The currently best ones are due to Eppstein for $3 \leq d \leq 10$ [5], and Feder & Motwani for $d \geq 11$ [6]. These algorithms are probabilistic, but since they would only be applied once each time line 10 of the algorithm is reached, the probability of error is not increased.

For problems with domain size 2 and l-ary constraints, i.e. l-SAT, we can do even better than this. First, consider the following formula:

$$(x \vee y) \wedge (\neg y \vee z)$$

If we want to find out if there are two solutions which differ on the assignment on x and agree on y and z, we would, using the algorithm in Fig. 8, get the following formula:

$$(x \vee y) \wedge (\neg y \vee z) \wedge (x' \vee y) \wedge (x \vee \neg x') \wedge (\neg x \vee x')$$

But there is a better way. Since there are only two possible domain values, and we force x' to always assume the opposite of x, there is no reason to create new variables. Instead, we duplicate the clauses containing variables on which the two solutions should differ, and among these clauses, we replace every literal containing one of these variables with its negation. In our example, we would get:

$$(x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee y)$$

This formula has a solution $\{x \mapsto 0, y \mapsto 1, z \mapsto 1\}$, and from this we can derive two solutions to the original formula which differ on the assignment of x, and agree on y and z.

As can be seen in Fig. 9, the algorithm for MAX HAMMING DISTANCE $(2,l)$-CSP is similar to the one for the general case, but it does not add any variables to the problem.

Theorem 3. *If we can solve* $(2,l)$*-CSP in* $\mathcal{O}(a^n)$*, then there exists an algorithm for solving* MAX HAMMING DISTANCE $(2,l)$*-CSP which runs in* $\mathcal{O}((2a)^n)$*.*

Proof. The algorithm in Fig. 9 considers all subsets of variables of the problem, as discussed in this section. Consequently, it will deliver a solution in $\mathcal{O}((2a)^n)$ time. □

Corollary 3. *For every* $\epsilon > 0$*, there exists an algorithm for solving the* MAX HAMMING DISTANCE $(2,l)$*-CSP in* $\mathcal{O}((4 - 4/l + \epsilon)^n)$ *time. Additionally, the special case of* MAX HAMMING DISTANCE $(2,3)$*-CSP can be solved in* $\mathcal{O}(2.6604^n)$ *time.*

Proof. Using the probabilistic $\mathcal{O}((2 - 2/l + \epsilon)^n)$ algorithm for l-SAT found in [14] on line 7 of the algorithm in Fig. 9 gives the first result, and replacing it with the (similarly probabilistic) 3-SAT algorithm from [8] gives the second. □

As could be expected, the algorithms we get from Corollary 3 offer a large improvement over the more general Corollary 2; Rather than an $\mathcal{O}(3.1111^n)$ algorithm for MAX HAMMING DISTANCE $(2,3)$-CSP, we get an $\mathcal{O}(2.6604^n)$. (Note that the gain we get from using the 3-SAT algorithm in [8] is not too impressive, since $4 - 4/3 = 2.6666\ldots$)

5 Conclusions

We have presented two different ways of approaching the MAX HAMMING DISTANCE problem for CSPs. The first, and perhaps most complicated of them, which solves the restricted case of when domains have 2 values, and the constraints are binary, is based on search in the microstructure graph of the $(2,2)$-CSP instance. There are several ways to improve the $\mathcal{O}(1.7338^n)$ running time of this algorithm. The first way is obviously to find a faster algorithm for solving weighted 2-SAT, since this algorithm is applied an exponential number of times during the search. The other way is to identify additional special cases and treat them separately during the search – the prime candidate being variables with degree $(4,1)$ – and work is underway.

Originally, we had hoped that the techniques used for the first algorithm would generalise to arbitrary domains, but it appears the size of the microstructure graph becomes too cumbersome as soon as we have more than two domain values. This, together with the fact that microstructures are limited to problems with binary constraints, forced us to come up with a completely different approach for larger domains and higher arity constraints. There is probably little hope of getting around the size of the microstructure for arbitrary domains, but it would be interesting to see if anything can be gained by restricting the domain, but allowing constraints with arity higher than 2.

6 Acknowledgments

The authors would like to thank Peter Jonsson for useful comments and interesting discussions during the writing of this paper.

References

1. O. Angelsmark and J. Thapper. Microstructure based algorithms for three constraint satisfaction optimisation problems, 2004. Unpublished manuscript. Available for download at http://www.ida.liu.se/~olaan/papers/three_algorithms.ps.
2. B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, Mar. 1979.
3. P. Crescenzi and G. Rossi. On the hamming distance of constraint satisfaction problems. *Theoretical Computer Science*, 288:85–100, 2002.
4. V. Dahllöf, P. Jonsson, and M. Wahlström. On counting models for 2SAT and 3SAT formulae, 2003. Unpublished manuscript. Available for download at http://www.ida.liu.se/~magwa/research/merge23sat.ps.
5. D. Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA-2001)*, pages 329–337, 2001.
6. T. Feder and R. Motwani. Worst-case time bounds for coloring and satisfiability problems. *Journal of Algorithms*, 45(2):192–201, Nov. 2002.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
8. T. Hofmeister, U. Schöning, R. Schuler, and O. Watanabe. A probabilistic 3-SAT algorithm further improved. In H. Alt and A. Ferriera, editors, *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS-2002)*, pages 192–202, Antibes Juan-les-Pins, France, 2002. Springer-Verlag, Berlin, Heidelberg.
9. P. G. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.
10. P. Jégou. 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)*, pages 731–736, Washington DC, USA, July 1993. AAAI.
11. O. Kullman. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1–2):1–72, 1999.
12. V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, Spring 1992.
13. U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.
14. U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (FOCS-1999)*, pages 410–414. IEEE Computer Society, 1999.