Optimal 2-constraint satisfaction via sum-product algorithms

Mikko Koivisto

HIIT Basic Research Unit, Department of Computer Science, University of Helsinki, Finland

Abstract

We show that for a given set of m pairwise constraints over n variables, a variable assignment that satisfies maximally many m constraints (MAX-2-CSP) can be found in $\mathrm{O}^*(nm\, d^{n\omega/3})$ time, where d is the maximum number of states per variable, and $\omega < 2.376$ is the matrix product exponent over a ring; the notation O^* suppresses factors polylogarithmic in m and n. As a corollary, MAX-2-SAT can be solved in $\mathrm{O}^*(nm 1.732^n)$ time. This improves on a recent result by Ryan Williams (ICALP 2004, 1227–1237) by reducing the polynomial factor from nm^3 to about nm.

Key words: Algorithms, Computational complexity, Matrix multiplication, Maximum satisfiability, Sum-product

1 The constraint satisfaction problem

We consider some variants of the maximum constraint satisfaction problem (MAX CSP). An instance of this problem consists of n variables $x_1, \ldots, x_n$, each x_i taking values from a finite domain $\mathcal{X}_i$ of size d_i, and m (weighted) constraints $w_1, \ldots, w_m$ that are integer-valued functions over subsets of the variables. The task is to find an assignment of values to the variables maximizing the sum of weights of the satisfied constraints.

More precisely, the *scope* of the cth constraint w_c is an index subset $S_c \subseteq \{1, \ldots, n\}$ that specifies a variable vector x_{S_c} taking values from $\mathcal{X}_{S_c}$. Here, and throughout this paper, we use indexing with subsets: If $S = \{i_1, \ldots, i_k\}$ with $1 \le i_1 < \cdots < i_k \le n$, we write $\mathcal{X}_S$ for $\mathcal{X}_{i_1} \times \cdots \times \mathcal{X}_{i_k}$ and x_S for $(x_{i_1}, \ldots, x_{i_k})$; for $S = \{1, \ldots, n\}$ we may drop the subscript. Thus, the constraint w_c maps

Email address: `mikko.koivisto@cs.helsinki.fi` (Mikko Koivisto).

each partial assignment x_{S_c} from $\mathcal{X}_{S_c}$ to a number from $\{0, 1, 2, \ldots\}$ (we do not allow negative weights). We say that a variable assignment $x \in \mathcal{X}$ satisfies the constraint w_c if the value $w_c(x_{S_c})$ is greater than zero. The *total weight* of the constraints $w_1, \ldots, w_m$ at x is the sum of the weights, $w_1(x_{S_1}) + \cdots + w_m(x_{S_m})$. Given an instance, the maximum constraint satisfaction problem is to find a variable assignment so as to maximize the total weight.

For analyzing purposes it is useful to consider restricted subclasses of MAX CSP. Most importantly, we bound the cardinality of the scopes by a constant k; in fact, we will focus on the case $k = 2$. We also bound the number of states per variable by d, and the maximum weight by R; however, d and R need not be constants.

MAX-k-CSP
Input: n variables, each with at most d states, m constraints with weights in $\{0, 1, \ldots, R\}$, each over at most k variables.
Output: An assignment x to the n variables such that the total weight of the constraints at x is maximized.

Note that the non-weighted version where each constraint is either satisfied or unsatisfied by a variable assignment is obtained by setting $R = 1$.

It is possible to solve MAX-k-CSP by reducing it to the following counting variant that asks the number of optimal assignments and the total weight achieved [11].

COUNT-k-CSP
Input: n variables, each with at most d states, m constraints with weights in $\{0, 1, \ldots, R\}$, each over at most k variables.
Output: The number of assignments x to the n variables such that the total weight of the constraints at x is maximized, and the maximum total weight W.

An important special case of MAX-k-CSP is MAX-k-SAT, where each variable is Boolean with two possible values, true and false, and each constraint (clause) is a disjunction of at most k literals and has weight 1 if the clause is satisfied (evaluates to true) and 0 otherwise. Thus, if an algorithm solves MAX-k-CSP, it trivially solves MAX-k-SAT as well. Analogously, COUNT-k-SAT, the problem of counting the number of satisfying assignments, is a special case of COUNT-k-CSP. Note also that the (non-weighted) existence versions, k-CSP and k-SAT, which ask for a variable assignment that satisfies every constraint, trivially reduce to MAX-k-CSP.

Another related problem, MAX CUT, is obtained as a special case of MAX-2-CSP. An instance of MAX CUT consists of m edge weights on n vertices (variables). The task is to find a partition of the vertices so as to maximize the sum of the weights of edges that cross the cut (i.e., the endpoints belong to different parts). Thus, an instance of MAX CUT can be represented as an instance of MAX-2-CSP (or MAX-2-SAT as well) with n binary variables (the value specifies the part to which the corresponding vertex belongs).

1.1 Previous research

Much theoretical work has been devoted to developing asymptotically fast exact algorithms for MAX-2-SAT. One of the best results is due to Gramm et al. [5] who give an algorithm that runs in time $\tilde{\mathrm{O}}(2^{m/5})$, where m is the number of clauses. [1] This is better than the trivial bound $\tilde{\mathrm{O}}(2^n)$, when the number of clauses m is less than $5n$. But note that m may well grow quadratically in n. The question, whether one can solve MAX-2-SAT in time $\tilde{\mathrm{O}}((2-\epsilon)^n)$ for some $\epsilon > 0$, has been raised explicitly by many authors (e.g., [8,1,12]).

Recently, Williams [11] answered this question by giving an algorithm that runs in time $\tilde{\mathrm{O}}(2^{n\omega/3})$, where ω is the matrix multiplication exponent, that is, the smallest constant such that two $N \times N$ square matrices can be multiplied in time $\mathrm{O}(N^\omega)$. The current record is $\omega < 2.376$ [2] yielding the time complexity $\tilde{\mathrm{O}}(1.732^n)$ for MAX-2-SAT. Williams's [11] method is rather general and applies to the general MAX-2-CSP as well. In fact, the essential part of Williams's method is his algorithm for solving the counting variant, COUNT-2-CSP. An outline of his approach to MAX-2-CSP is as follows: (1) Split the n variables into three disjoint groups of sizes (about) $n/3$. (2) Then construct a graph on the corresponding $3d^{n/3}$ partial variable assignments such that each (weighted) 3-clique in the graph corresponds to a variable assignment in the COUNT-2-SAT instance. (3) Count the number of optimal 3-cliques using fast matrix multiplication; this solves the COUNT-2-CSP instance. (4) Finally, use self-reducibility to reconstruct an optimal variable assignment. The first phase that solves COUNT-2-SAT runs in time $\mathrm{O}(m^3 d^{n\omega/3})$. Williams's analysis for the last step (4) gives $\mathrm{O}(nm^3 d^{n\omega/3})$ bound for the total running time. (In fact, Williams's construction allows for partitioning the variables into more than three groups, but the complexity bound is obtained with three groups.)

In addition to William's work, constraint satisfaction problems have also been viewed as special classes of a more general family of sum-product problems [10,4,7]. In a sum-product problem the task is to evaluate a multi-dimensional sum of a product of lower-dimensional functions; here the addition and multiplication operations need not be the conventional ones, as long as they form a semiring. The generic algorithm for sum-product problems is known as variable elimination (or bucket elimination) [10,4,7]; this dynamic-programming

[1] The notation $\tilde{\mathrm{O}}$ suppresses polynomial factors: we write $f(x) \in \tilde{\mathrm{O}}(g(x))$ if $f(x) \in \mathrm{O}(p(x) \cdot g(x))$ for some polynomial function $p(x)$.

algorithm solves the summation problem by marginalizing the variables out one by one. The running time depends on the induced treewidth of the (hyper)graph structure determined by the scopes of the given low-dimensional functions (see, e.g., [4]). Applied directly to 2-CSP or COUNT-2-CSP this framework gives only the trivial bound $\tilde{O}(d^n)$ (see, e.g., [4]).

1.2 Contributions and outline of this paper

Based on the sum-product framework, this paper provides a new look at the matrix multiplication approach to COUNT-2-CSP; see [6, Ch. 3] for similar ideas regarding general sum-product problems. Like Williams [11], we partition the variables into three groups and connect any two groups by the constraints that mention variables from these groups (the first step). The key difference is in the second and third steps: we turn to matrix multiplication directly, avoiding the transformation to the clique problem. We do this, in Section 2, by formulating a sum-product expression whose value encodes the number of optimal variable assignments and the associated total weight. This treatment gives the improved upper bound $O(nm \log(nm) \log\log(nm)\, d^{n\omega/3})$ for the time complexity of COUNT-2-CSP (throughout this paper the base of the logarithms is 2).

We also provide a more careful, yet very simple analysis of the self-reduction technique of Williams's. In Section 3, we show that the factor n can be avoided, yielding the bound $O(nm \log(nm) \log\log(nm)\, d^{n\omega/3})$ for the time complexity of MAX-2-CSP.

While the improvements over Williams's results concern the polynomial factor only, for potential practical implementations the improvement is important. If, for example, the number of constraint m grows as $\Theta(n^2)$, then the improvement is roughly by the factor n^4.

In our analysis we do not assume that d and R are constants. This generality results in slightly more complex bounds than the ones stated above for constant d and R.

2 An algorithm for COUNT-2-CSP

Fix an instance of COUNT-k-CSP with the notation given in the previous section. Let $M = mR$ be the maximum total weight. For $r = 0, 1, \ldots, M$, let α_r be the number of variable assignments for which the total weight equals r. We are interested in finding the largest r for which $\alpha_r > 0$.

We next give an algorithm that finds the value α_r for all $r = 0, 1, \ldots, M$. We do this by constructing a sum-product formulation as follows. Let $\beta = 2^l$ where l is an integer satisfying $d^n < 2^l \leq 2d^n$; we will use β as the base of some exponential terms, as described below. For each constraint $c = 1, \ldots, m$ define

$$f_c(x_{S_c}) := \beta^{w_c(x_{S_c})} \in \{1, \beta, \beta^2, \ldots, \beta^R\}\,.$$

For all assignments $x \in \mathcal{X}$ consider the product $\prod_{c=1}^{m} f_c(x_{S_c})$; it is β^r, where r is the total weight at constraints satisfied by x. Denote

$$\gamma := \sum_{x \in \mathcal{X}} \prod_{c=1}^{m} f_c(x_{S_c})\,. \tag{1}$$

It is easy to see the following.

Lemma 1 *We have $\gamma = \alpha_0 + \alpha_1\beta + \alpha_2\beta^2 + \cdots + \alpha_M\beta^M$.*

Proof For each r the term β^r appears in sum (1) exactly α_r times. $\square$

Moreover, the coefficients α_r can be easily read from γ:

Lemma 2 *Given γ, the numbers $\alpha_0, \alpha_1, \ldots, \alpha_M$ are unique and can be found in time linear in $M \log \beta$.*

Proof Observe that each coefficient α_r is smaller than $\beta = 2^l$. Thus, by Lemma 1, the binary representation of γ uses $\mathrm{O}(M \log \beta)$ bits and readily tabulates the numbers $\alpha_0, \alpha_1, \ldots, \alpha_M$. $\square$

We next restrict our attention to the case $k = 2$ where each constraint can mention at most two variables. It turns out that in this case we can apply fast matrix multiplication methods to solve COUNT-2-CSP.

Theorem 3 *COUNT-2-CSP can be solved in $\mathrm{O}(\mu(nM \log d)\, d^{a+n'\omega})$ time, where $\mu(b) = b \log b \log\log b$, $n' = \lfloor n/3 \rfloor$, and $a = n - 3n' \in \{0, 1, 2\}$.*

Proof To prove the claim we may assume that every variable has exactly d states. By the above lemmas (Lemmas 1 and 2) it is sufficient to show that the number γ, given in (1), can be evaluated in $\mathrm{O}(\mu(nM \log d)\, d^{a+n'\omega})$ time.

Namely, from γ we can easily obtain both the largest r for which $\alpha_r > 0$ (the maximum total weight) and the number α_r itself (the number of optimal assignments).

Suppose first that n is divisible by 3. Let $\{I, J, K\}$ be a partition of $\{1, \ldots, n\}$ into three disjoint sets, each of size $n/3$. Define further a corresponding partitioning $\{A, B, C\}$ of the index set $\{1, \ldots, m\}$ of the constraints by

$$\begin{aligned}
A &:= \{c : S_c \subseteq I \cup J\}\,, \\
B &:= \{c : S_c \cap K \neq \emptyset \text{ and } S_c \cap J = \emptyset\}\,, \\
C &:= \{c : S_c \cap K \neq \emptyset \text{ and } S_c \cap J \neq \emptyset\}\,.
\end{aligned}$$

This, indeed, gives a partition of the constraints, since a scope S_c can contain at most two elements, and thus cannot intersect all the three sets I, J, and K; obviously, every scope is included.

We continue by factorizing the product that appears in the sum-production expression (1) of γ. For all variable assignments $x \in \mathcal{X}$ let

$$\mathcal{A}(x_I, x_J) := \prod_{c \in A} f_c(x_{S_c}),\ \mathcal{B}(x_I, x_K) := \prod_{c \in B} f_c(x_{S_c}),\ \mathcal{C}(x_J, x_K) := \prod_{c \in C} f_c(x_{S_c})\,.$$

Note that x_I, x_J, and x_K form a partition of the n variables. The key observation is that

$$\gamma = \sum_{x_I} \sum_{x_J} \sum_{x_K} \mathcal{A}(x_I, x_J)\, \mathcal{B}(x_I, x_K)\, \mathcal{C}(x_J, x_K) = \sum_{x_I} \sum_{x_J} \mathcal{A}(x_I, x_J)\, \mathcal{D}(x_I, x_J)\,,$$

where

$$\mathcal{D}(x_I, x_J) := \sum_{x_K} \mathcal{B}(x_I, x_K)\, \mathcal{C}(x_J, x_K)\,.$$

Let us analyze the time needed to evaluate the above expressions. We view $\mathcal{A}$ (similarly $\mathcal{B}$ and $\mathcal{C}$) as a $d^{n/3} \times d^{n/3}$ matrix, where the row and column indexes are determined by bijections from $\mathcal{X}_I$ and $\mathcal{X}_J$ to $\{1, 2, \ldots, d^{n/3}\}$; constructing such matrix representation introduces only a negligible computational cost. With this interpretation, we observe that $\mathcal{D}$ is obtained as the product of $\mathcal{B}$ and the transpose of $\mathcal{C}$. Thus, $\mathcal{D}$ can be evaluated using $\mathrm{O}(d^{n\omega/3})$ arithmetic operations with numbers that are representable in $\mathrm{O}(nM \log d)$ bits (since $\log \gamma < (M + 2) \log \beta \leq (M + 2) \log(2d^n)$). We know that two b-bit integers can be multiplied in $\mathrm{O}(\mu(b))$ time, where $\mu(b) = b \log b \log \log b$ [9], and that addition and substraction are cheaper. Consequently, $\mathcal{D}$ can be computed in

$O(\mu(nM \log d)\, d^{n\omega/3})$ time. Finally, we notice that computing γ given $\mathcal{D}$ is much easier, since only two thirds of the n variables are involved.

Then suppose that n is not divisible by 3. We choose $a = n - 3\lfloor n/3 \rfloor < 3$ variables and construct d^a new instances for the rest $n-a$ variables. Now $n-a$ is divisible by 3 and by the above described procedure each instance can be solved in $O(\mu(nM \log d)\, d^{(n-a)\omega/3})$ time, yielding the claimed total complexity. $\square$

As a corollary we obtain a simplified bound when the domain size d and the maximum weight per constraint R are constants.

Corollary 4 *For constant d and R, COUNT-2-CSP can be solved in $O(\mu(nm)\, d^{n\omega/3})$ time, where $\mu(b) = b \log b \log\log b$.*

Proof Recall that $M = mR$ and use Theorem 3. $\square$

Remark 5 *Theorem 3 and Corollary 4 rely on the fastest known method for integer multiplication due to Schönhage and Strassen [9]. That algorithm is not particularly suitable for practical implementations. However, it seems that a practical and asymptotically at least as fast algorithm for COUNT-2-CSP can be obtained via the Chinese remainder technique (e.g., [3, Ch. 33]), which allows us to replace the matrix multiplication with large integers by* $o(b)$ *matrix multiplications with small integers representable using only* $O(\log b)$ *bits, where $b = nM \log d$. We omit detailed analysis.*

3 An algorithm for MAX-2-CSP

For simplicity we consider the case of constant d so that we do not have to care whether n is divisible by 3; the generalization to non-constant d is straightforward.

The proof of the following result uses the self-reducibility argument of Williams [11]. Here we give a more detailed description of the algorithm to support our tighter running time analysis.

Theorem 6 *For any constant d, MAX-2-CSP can be solved in* $O(\mu(nM)d^{n\omega/3})$ *time, where $M = mR$ and $\mu(b) = b \log b \log\log b$.*

Proof We show how an optimal assignment is found. Let W be the maximum total weight that can be obtained by some variable assignment. As described in

Section 2, this number can be found in $O(\mu(nM)d^{n\omega/3})$ time. We now construct an optimal assignment by solving iteratively a sequence of smaller problems:

```
(1) For $i = 1, \ldots, n$,
    (a) set $x_i := 0$;
    (b) replace each constraint $w_c$ whose scope $S_c$ contains $i$ by a new constraint $w'_c$ defined by $w'_c(x_{S_c \setminus \{i\}}) := w_c(x_{S_c})$;
    (c) compute the optimal weight $W'$ for the new instance;
    (d) if $W' < W$, set $x_i := 1$.
(2) Return $x$.
```

This algorithm outputs an optimal assignment, since for each i there must be an assignment to $x_1, \ldots, x_{i-1}$ such that the constructed new instance has the optimal weight W.

At the ith iteration of the above algorithm an instance with $n - i$ variables is solved in $O(\mu(nM)d^{(n-i)\omega/3})$ time. As $\sum_{i=0}^{n} d^{(n-i)\omega/3} < d^{n\omega/3}/(1 - d^{-\omega/3})$, the total time complexity is the claimed $O(\mu(nM)d^{n\omega/3})$. □

The MAX-2-SAT problem is a special case of MAX-2-CSP where each variable has two possible values (true and false) and each constraint (disjunction of at most two literals) has weight 1 if the constraint is satisfied, and 0 otherwise. We may assume that the number of constraints m is at most $4n^2$.

Corollary 7 *MAX-2-SAT can be solved in* $O(nm \log n \log\log n \, 1.732^n)$ *time.*

Proof Use Theorem 6 with $d = 2$, $m \le 4n^2$, $R = 1$, and $\omega < 2.376$ [2]. □

Remark 8 *The discussion in Remark 5 applies to Theorem 6 and Corollary 7 as well.*

Acknowledgements

I am grateful to Heikki Mannila for valuable discussions and to an anonymous reviewer for the constructive comments that helped to improve the presentation.

References

[1] J. Alber, J. Gramm, R. Niedermeier, Faster exact algorithms for hard problems: a parameterized point of view, Discrete Math. 229 (1-3) (2001) 3–27.

[2] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, J. Symbolic Comput. 9 (3) (1990) 791–799.

[3] T.H. Cormen, C.L. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, Massachusetts, 1990.

[4] R. Dechter, Bucket elimination: A unifying framework for reasoning, Artificial Intelligence 113 (1–2) (1999) 41–85.

[5] J. Gramm, E.A. Hirsch, R. Niedermeier, P. Rossmanith, Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT, Discrete Appl. Math. 130 (2) (2003) 139–155.

[6] M. Koivisto, Sum-product algorithms for the analysis of genetic risks, Ph.D. thesis, University of Helsinki (January 2004).

[7] F.R. Kschischang, B. Frey, H.-A. Loeliger, Factor graphs and the sum-product algorithm, IEEE Trans. Inform. Theory 47 (2) (2001) 498–519.

[8] V. Raman, B. Ravikumar, S.S. Rao, A simplified NP-complete MAXSAT problem, Inform. Process. Lett. 65 (1) (1998) 1–6.

[9] A. Schönhage, V. Strassen, Schnelle Multiplikation großer Zahlen, Computing 7 (1971) 281–292.

[10] R.E. Stearns, H.B. Hunt III, An algebraic model for combinatorial problems, SIAM J. Comput. 25 (2) (1996) 448–476.

[11] R. Williams, A new algorithm for optimal constraint satisfaction and its implications, in: J. Diaz et al. (Eds.), ICALP 2004, in: Lecture Notes in Comput. Sci., vol. 3142, Springer, Berlin, 2004, pp. 1227–1237.

[12] G.J. Woeginger, Exact algorithms for NP-hard problems: A survey, in: M. Jünger et al. (Eds.), Combinatorial Optimization – Eureka, You Shrink!, in: Lecture Notes in Comput. Sci., vol. 3142, Springer, Berlin, 2003, pp. 185–207.