Maximum 2-satisfiability (2004; Williams)

Ryan Williams, Carnegie Mellon University, `www.cs.cmu.edu/~ryanw`

INDEX TERMS: Satisfiability, exact algorithms, constraint satisfaction

SYNONYMS: Max 2-SAT

1 PROBLEM DEFINITION

In the maximum 2-satisfiability problem (abbreviated as Max 2-Sat), one is given a Boolean formula in conjunctive normal form, such that each clause contains at most two literals. The task is to find an assignment to the variables of the formula such that a maximum number of clauses is satisfied.

Max 2-Sat is a classic optimization problem. Its decision version was proved NP-complete by Garey, Johnson, and Stockmeyer [7], in stark contrast with 2-Sat which is solvable in linear time [2]. To get a feeling for the difficulty of the problem, the NP-completeness reduction is sketched here. One can transform any 3-Sat instance F into a Max 2-Sat instance F', by replacing each clause of F such as

$$c_i = (\ell_1 \vee \ell_2 \vee \ell_3),$$

where ℓ_1, ℓ_2, and ℓ_3 are arbitrary literals, with the collection of 2-CNF clauses

$$(\ell_1), (\ell_2), (\ell_3), (c_i), (\neg\ell_1 \vee \neg\ell_2), (\neg\ell_2 \vee \neg\ell_3), (\neg\ell_1 \vee \neg\ell_3), (\ell_1 \vee \neg c_i), (\ell_2 \vee \neg c_i), (\ell_3 \vee \neg c_i),$$

where c_i is a new variable. The following are true:

- If an assignment satisfies c_i, then exactly seven of the ten clauses in the 2-CNF collection can be satisfied.
- If an assignment does not satisfy c_i, then exactly six of the ten clauses can be satisfied.

If F is satisfiable then there is an assignment satisfying $7/10$ of the clauses in F', and if F is not satisfiable then no assignment satisfies $7/10$ of the clauses in F'. Since 3-Sat reduces to Max 2-Sat, it follows that Max 2-Sat (as a decision problem) is NP-complete.

Notation A CNF formula is represented as a set of clauses.

The symbols $\mathbb{R}$ and $\mathbb{Z}$ denote the sets of reals and integers, respectively. The letter ω denotes the smallest real number such that for all $\epsilon > 0$, n by n matrix multiplication over a ring can be performed in $O(n^{\omega+\epsilon})$ ring operations. Currently, it is known that $\omega < 2.376$ [4]. The ring matrix product of two matrices A and B is denoted by $A \times B$.

Let A and B be matrices with entries from $\mathbb{R} \cup \{\infty\}$. The *distance product* of A and B (written shorthand as $A \otimes_d B$) is the matrix C defined by the formula

$$C[i,j] = \min_{k=1,\ldots,n} \{A[i,k] + B[k,j]\}.$$

A word on m's and n's: in reference to graphs, m and n denote the number of edges and the number of nodes in the graph, respectively. In reference to CNF formulas, m and n denote the number of clauses and the number of variables, respectively.

2 KEY RESULT

The primary result of this article is a procedure solving Max 2-Sat in $O(m \cdot 2^{\omega n/3})$ time. The method can be generalized to *count* the number of solutions to *any* constraint optimization problem with at most two variables per constraint (cf. [17]), though the presentation in this article shall be somewhat different from the reference, and much simpler. There are several other known exact algorithms for Max 2-Sat that are more effective in special cases, such as sparse instances [9, 8, 3, 13, 15, 16, 11, 12]. The procedure described below is the only one known (to date) that runs in $O(poly(m) \cdot 2^{\delta n})$ time (for some fixed $\delta < 1$) in all possible cases.

2.1 KEY IDEA

The algorithm gives a reduction from Max 2-Sat to the problem Max Triangle, in which one is given a graph with integer weights on its nodes and edges, and the goal is to output a 3-cycle of maximum weight. At first, the existence of such a reduction sounds strange, as Max Triangle can be trivially solved in $O(n^3)$ time by trying all possible 3-cycles. The key is that the reduction exponentially increases the problem size, from a Max 2-Sat instance with m clauses and n variables, to a Max Triangle instance having $O(2^{2n/3})$ edges, $O(2^{n/3})$ nodes, and weights in the range $\{-m, \ldots, m\}$.

Note that if Max Triangle required $\Theta(n^3)$ time to solve, then the resulting Max 2-Sat algorithm would take $\Theta(2^n)$ time, rendering the above reduction pointless. However, it turns out that the brute-force search of $O(n^3)$ for Max Triangle is not the best one can do– using fast matrix multiplication, there is an algorithm for Max Triangle that runs in $O(Wn^{\omega})$ time on graphs with weights in the range $\{-W, \ldots, W\}$.

2.2 MAIN ALGORITHM

First, a reduction from Max 2-Sat to Max Triangle is described, arguing that each triangle of weight K in the resulting graph is in one-to-one correspondence with an assignment that satisfies K clauses of the Max 2-Sat instance. Let a, b be reals, and let $\mathbb{Z}[a,b] := [a,b] \cap \mathbb{Z}$.

Lemma 1. *If* Max Triangle *on graphs with n nodes and weights in $\mathbb{Z}[-W, W]$ is solvable in $O(f(W) \cdot g(n))$ time, for polynomials f and g, then* Max 2-Sat *is solvable in $O(f(m) \cdot g(2^{n/3}))$ time, where m is the number of clauses and n is the number of variables.*

PROOF: Let C be a given 2-CNF formula. Assume without loss of generality that n is divisible by 3. Let F be an instance of MAX 2-SAT. Arbitrarily partition the n variables of F into three sets P_1, P_2, P_3, each having $n/3$ variables. For each P_i, make a list L_i of all $2^{n/3}$ assignments to the variables of P_i.

Define a graph $G = (V, E)$ with $V = L_1 \cup L_2 \cup L_3$ and $E = \{(u, v) | u \in P_i, v \in P_j, i \neq j\}$. That is, G is a complete tripartite graph with $2^{n/3}$ nodes in each part, and each node in G corresponds to an assignment to $n/3$ variables in C. Weights are placed on the nodes and edges of G as follows. For a node v, define $w(v)$ to be the number of clauses that are satisfied by the partial assignment denoted by v. For each edge $\{u, v\}$, define $w(\{u, v\}) = -W_{uv}$, where W_{uv} is the number of clauses that are satisfied by *both* u and v.

Define the weight of a triangle in G to be the total sum of all weights and nodes in the triangle.

Claim 1. *There is a one-to-one correspondence between the triangles of weight K in G and the variable assignments satisfying exactly K clauses in F.*

PROOF: Let a be a variable assignment. Then there exist unique nodes $v_1 \in L_1$, $v_2 \in L_2$, and $v_3 \in L_3$ such that a is precisely the concatenation of v_1, v_2, v_3 as assignments. Moreover, any triple of nodes $v_1 \in L_1$, $v_2 \in L_2$, and $v_3 \in L_3$ corresponds to an assignment. Thus there is a one-to-one correspondence between triangles in G and assignments to F.

The number of clauses satisfied by an assignment is exactly the weight of its corresponding triangle. To see this, let $T_a = \{v_1, v_2, v_3\}$ be the triangle in G corresponding to assignment a. Then

$$\begin{aligned} w(T_a) &= w(v_1) + w(v_2) + w(v_3) + w(\{v_1, v_2\}) + w(\{v_2, v_3\}) + w(\{v_1, v_3\}) \\ &= \sum_{i=1}^{3} |\{c \in F \mid v_i \text{ satisfies } F\}| - \sum_{i,j\ :\ i \neq j} |\{c \in F \mid v_i \textit{ and } v_j \text{ satisfy } F\}| \\ &= |\{c \in F \mid a \text{ satisfies } F\}|, \end{aligned}$$

where the last equality follows from the inclusion-exclusion principle. ■

Notice that the number of nodes in G is $3 \cdot 2^{n/3}$, and the absolute value of any node and edge weight is m. Therefore, running a MAX TRIANGLE algorithm on G, a solution to MAX 2-SAT is obtained in $O(f(m) \cdot g(3 \cdot 2^{n/3}))$, which is $O(f(m) \cdot g(2^{n/3}))$ since g is a polynomial. This completes the proof of Lemma 1. ■

Next, a procedure is described for finding a maximum triangle faster than brute-force search, using fast matrix multiplication. Alon, Galil, and Margalit [1] (following Yuval [20]) showed that the distance product for matrices with entries drawn from $\mathbb{Z}[-W, W]$ can be computed using fast matrix multiplication as a subroutine.

Theorem 2 (Alon, Galil, Margalit [1]). *Let A and B be $n \times n$ matrices with entries from $\mathbb{Z}[-W, W] \cup \{\infty\}$. Then $A \otimes_d B$ can be computed in $O(Wn^\omega \log n)$ time.*

PROOF: (Sketch) One can replace ∞ entries in A and B with $2W + 1$ in the following. Define matrices A' and B', where

$$A'[i, j] = x^{3W - A[i,j]}, \quad B'[i, j] = x^{3W - B[i,j]},$$

and x is a variable. Let $C = A' \times B'$. Then

$$C[i,j] = \sum_{k=1}^{n} x^{6W - A[i,k] - B[k,j]}.$$

The next step is to pick a number x that makes it easy to determine, from the sum of arbitrary powers of x, the largest power of x appearing in the sum; this largest power immediately gives the minimum $A[i,k] + B[k,j]$. Each $C[i,j]$ is a polynomial in x with coefficients from $\mathbb{Z}[0,n]$. Suppose each $C[i,j]$ is evaluated at $x = (n+1)$. Then each entry of $C[i,j]$ can be seen as an $(n+1)$-ary number, and the position of this number's most significant digit gives the minimum $A[i,k]+B[k,j]$.

In summary, $A \otimes_d B$ can be computed by constructing

$$A'[i,j] = (n+1)^{3W - A[i,j]}, \quad B'[i,j] = (n+1)^{3W - B[i,j]}$$

in $O(W \log n)$ time per entry, computing $C = A' \times B'$ in $O(n^\omega \cdot (W \log n))$ time (as the sizes of the entries are $O(W \log n)$), then extracting the minimum from each entry of C, in $O(n^2 \cdot W \log n)$ time. Note if the minimum for an entry $C[i,j]$ is at least $2W+1$, then $C[i,j] = \infty$. ■

Using the fast distance product algorithm, one can solve Max Triangle faster than brute-force. The following is based on an algorithm by Itai and Rodeh [10] for detecting if an unweighted graph has a triangle in less than n^3 steps. The result can be generalized to *counting* the number of *k-cliques*, for arbitrary $k \geq 3$. (To keep the presentation simple, the counting result is omitted. Concerning the k-clique result, there is unfortunately no asymptotic runtime benefit from using a k-clique algorithm instead of a triangle algorithm, given the current best algorithms for these problems.)

Theorem 3. Max Triangle *can be solved in $O(Wn^\omega \log n)$, for graphs with weights drawn from $\mathbb{Z}[-W,W]$.*

Proof: First, it is shown that a weight function on nodes and edges can be converted into an equivalent weight function with weights on only edges. Let w be the weight function of G, and redefine the weights to be:

$$w'(\{u,v\}) = \frac{w(u)+w(v)}{2} + w(\{u,v\}), \quad w'(u) = 0.$$

Note the weight of a triangle is unchanged by this reduction.

The next step is to use a fast distance product to find a maximum weight triangle in an edge-weighted graph of n nodes. Construe the vertex set of G as the set $\{1,\ldots,n\}$. Define A to be the $n \times n$ matrix such that $A[i,j] = -w(\{i,j\})$ if there is an edge $\{i,j\}$, and $A[i,j] = \infty$ otherwise. The claim is that there is a triangle through node i of weight at least K if and only if $(A \otimes_d A \otimes_d A)[i,i] \leq -K$. This is because $(A \otimes_d A \otimes_d A)[i,i] \leq -K$ if and only if there are distinct j and k such that $\{i,j\}$, $\{j,k\}$, $\{k,i\}$ are edges and $A[i,j] + A[j,k] + A[k,i] \leq -K$, *i.e.*, $w(\{i,j\}) + w(\{j,k\}) + w(\{k,i\}) \geq K$.

Therefore, by finding an i such that $(A \otimes_d A \otimes_d A)[i,i]$ is minimized, one obtains a node i contained in a maximum triangle. To obtain the actual triangle, check all m edges $\{j,k\}$ to see if $\{i,j,k\}$ is a triangle. ■

Theorem 4. MAX 2-SAT *can be solved in* $O(m \cdot 1.732^n)$ *time.*

PROOF: Given a set of clauses C, apply the reduction from Lemma 1 to get a graph G with $O(2^{n/3})$ nodes and weights from $\mathbb{Z}[-m, m]$. Apply the algorithm of Theorem 3 to output a max triangle in G in $O(m \cdot 2^{\omega n/3} \log(2^{n/3})) = O(m \cdot 1.732^n)$ time, using the $O(n^{2.376})$ matrix multiplication of Coppersmith and Winograd [4]. ■

3 APPLICATIONS

By modifying the graph construction, one can solve other problems in $O(1.732^n)$ time, such as MAX CUT, MINIMUM BISECTION, and SPARSEST CUT. In general, any constraint optimization problem for which each constraint has at most two variables can be solved faster using the above approach. For more details, see [17] and the recent survey by Woeginger [19]. Techniques similar to the above algorithm have also been used by Dorn [6] to speed up dynamic programming for some problems on planar graphs (and in general, graphs of bounded branchwidth).

4 OPEN PROBLEMS

- Improve the space usage of the above algorithm. Currently, $\Theta(2^{2n/3})$ space is needed. A very interesting open question is if there is a $O(1.99^n)$ time algorithm for MAX 2-SAT that uses only *polynomial* space. This question would have a positive answer if one could find an algorithm for solving the k-CLIQUE problem that uses polylogarithmic space and $n^{k-\delta}$ time for some $\delta > 0$ and $k \geq 3$.

- Find a faster-than-2^n algorithm for MAX 2-SAT that does not require fast matrix multiplication. The fast matrix multiplication algorithms have the unfortunate reputation of being impractical.

- Generalize the above algorithm to work for MAX k-SAT, where k is any positive integer. The current formulation would require one to give an efficient algorithm for finding a small hyperclique in a hypergraph. However, no general results are known for this problem. It is conjectured that for all $k \geq 2$, MAX k-SAT is in $\tilde{O}(2^{n(1-\frac{1}{k+1})})$ time, based on the conjecture that matrix multiplication is in $n^{2+o(1)}$ time [17].

5 CROSS REFERENCES

Matrix Multiplication and All Pairs Shortest Paths, CNF SAT and MAXSAT Algorithms, Max Cut, Minimum Bisection, Sparsest Cut

6 RECOMMENDED READING

References

[1] Alon N., Galil Z., and Margalit O. On the exponent of the all-pairs shortest path problem. *J. Computer and System Sciences* 54:255–262, 1997.

[2] Aspvall B., Plass M. F., and Tarjan R. E. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Proc. Letters* 8(3):121–123, 1979.

[3] Bansal N. and Raman V. Upper bounds for Max Sat: Further Improved. In *Proceedings of ISAAC*, Springer-Verlag LNCS 1741:247–258, 1999.

[4] Coppersmith D. and Winograd S. Matrix Multiplication via Arithmetic Progressions. *JSC* 9(3):251–280, 1990.

[5] Dantsin E. and Wolpert A. Max SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In *Proc. of the 9th International Conference on Theory and Applications of Satisfiability Testing*, Springer-Verlag LNCS 4121:266–276, 2006.

[6] Dorn F. Dynamic Programming and Fast Matrix Multiplication. In *Proceedings of 14th Annual European Symposium on Algorithms*, Springer-Verlag LNCS 4168:280–291, 2006.

[7] Garey M., Johnson D., and Stockmeyer L. Some simplified NP-complete graph problems. *Theor. Comput. Sci.* 1:237–267, 1976.

[8] Gramm J. and Niedermeier R. Faster exact solutions for Max2Sat. In *Proceedings of CIAC*, Springer-Verlag LNCS 1767:174–186, 2000.

[9] Hirsch E. A. A $2^{m/4}$-time Algorithm for Max 2-SAT: Corrected Version. *Electronic Colloquium on Computational Complexity* Report TR99-036, 2000.

[10] Itai A. and Rodeh M. Finding a Minimum Circuit in a Graph. *SIAM J. Comput.* 7(4):413–423, 1978.

[11] Kneis J., Mölle D., Richter S., and Rossmanith P. Algorithms Based on the Treewidth of Sparse Graphs. In *Proc. Workshop on Graph Theoretic Concepts in Computer Science*, Springer-Verlag LNCS 3787:385–396, 2005.

[12] Kojevnikov A. and Kulikov A. S. A New Approach to Proving Upper Bounds for Max 2-SAT. In *Proc. of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 11–17, 2006.

[13] Mahajan M. and Raman V. Parameterizing above Guaranteed Values: MAXSAT and MAXCUT. *J. Algorithms* 31(2): 335–354, 1999.

[14] Niedermeier R. and Rossmanith P. New upper bounds for maximum satisfiability. *J. Algorithms* 26:63–88, 2000.

[15] Scott A. and Sorkin G. Faster Algorithms for MAX CUT and MAX CSP, with Polynomial Expected Time for Sparse Instances. In *Proceedings of RANDOM-APPROX 2003*, Springer-Verlag LNCS 2764:382–395, 2003.

[16] Williams R. On Computing k-CNF Formula Properties. In *Theory and Applications of Satisfiability Testing*, Springer-Verlag LNCS 2919:330–340, 2004.

[17] Williams R. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.* 348(2-3): 357–365, 2005.

[18] Woeginger G. J. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization - Eureka! You shrink!*, Springer-Verlag LNCS 2570:185–207, 2003.

[19] Woeginger G. J. Space and time complexity of exact algorithms: some open problems. In *Proc. 1st Int. Workshop on Parameterized and Exact Computation* (IWPEC 2004), Springer-Verlag LNCS 3162, 281-290, 2004.

[20] Yuval G. An Algorithm for Finding All Shortest Paths Using $N^{2.81}$ Infinite-Precision Multiplications. *Inf. Process. Lett.* 4(6):155–156, 1976.