Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 348 (2005) 357–365
Theoretical Computer Science
www.elsevier.com/locate/tcs

A new algorithm for optimal 2-constraint satisfaction and its implications☆

Ryan Williams

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract

We present a novel method for exactly solving (in fact, counting solutions to) general constraint satisfaction optimization with at most two variables per constraint (e.g. MAX-2-CSP and MIN-2-CSP), which gives the first exponential improvement over the trivial algorithm. More precisely, the runtime bound is a constant factor improvement in the base of the exponent: the algorithm can count the number of optima in MAX-2-SAT and MAX-CUT instances in $O(m^3 2^{\omega n/3})$ time, where $\omega < 2.376$ is the matrix product exponent over a ring. When the constraints have arbitrary weights, there is a $(1+\varepsilon)$-approximation with roughly the same runtime, modulo polynomial factors. Our construction shows that improvement in the runtime exponent of either *k-clique* solution (even when $k = 3$) or *matrix multiplication over GF(2)* would improve the runtime exponent for solving 2-CSP optimization.

Our approach also yields connections between the complexity of some (polynomial time) high-dimensional search problems and some NP-hard problems. For example, if there are sufficiently faster algorithms for computing the diameter of n points in ℓ_1, then there is an $(2-\varepsilon)^n$ algorithm for MAX-LIN. These results may be construed as either lower bounds on the high-dimensional problems, or hope that better algorithms exist for the corresponding hard problems.

Keywords: Exact algorithms; Constraint satisfaction; MAX-2-SAT; MAX-CUT

1. Introduction

The extent to which NP-hard problems are indeed hard to solve remains largely undetermined. For some problems, it intuitively seems that the best one can do is examine every candidate solution, but this intuition has been shown to fail in many scenarios. The fledgling development of improved exponential algorithms in recent times suggests that for many hard problems, something substantially faster than brute-force search can be done, even in the worst case. However, some fundamental problems have persistently eluded researchers from better algorithms. One popular example in the literature is MAX-2-SAT.

There has been notable theoretical interest in discovering a procedure for MAX-2-SAT running in $O((2-\varepsilon)^n)$ steps on all instances, for some $\varepsilon > 0$. Unlike problems such as Vertex Cover and k-SAT, where analysis of branch-and-bound techniques (or random choice of assignments) sufficed for improving the naïve time bounds (e.g. [23,20,13]), MAX-SAT has been surprisingly difficult to attack. Previous work has only been able to show either a $(2-\varepsilon)^m$ time bound, where m is the number of clauses, or an approximation scheme running in $(2-\varepsilon)^n$ [12,8] (but $\varepsilon \to 0$ as the

☆ Supported by the NSF ALADDIN Center (NSF Grant No. CCR-0122581) and an NSF Graduate Research Fellowship.
E-mail address: ryanw@cs.cmu.edu.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.09.023

quality of the approximation goes to 1). Of course, the number of clauses can, in general, be far higher than the number of variables, so the $(2-\varepsilon)^m$ bounds only improve the trivial bound on some instances. The current best worst-case bound for MAX-2-SAT explicitly in terms of m is $\tilde{O}(2^{m/5})$, [1] by [9] (so for $m/n > 5$, this is no better than 2^n). This result followed a line of previous papers on bounds in terms of m [11,19,10,6,4,17]; a similar line has formed for MAX-CUT [28,16]. In terms of partial answers to the problem, [21] showed that when every variable occurs at most three times, MAX-2-SAT remains NP-complete, but has an $O(n3^{n/2})$ algorithm. While definite stepping stones, these results were still distant from an improved exponential algorithm. Consequently, several researchers (e.g. [21,1,8,29]) explicitly proposed a $(2-\varepsilon)^n$ algorithm for MAX-2-SAT (and/or MAX-CUT) as a benchmark open problem in exact algorithms.

1.1. Outline of our approach: split and list

Most exact algorithms for NP-hard problems in the literature involve either a case analysis of a branch-and-bound strategy [9], repeated random choice of assignments [20], or local search [25]. Our design is a major departure from these approaches, resembling earlier algorithms from the 1970s [14,26]. We *split* the set of n variables into k partitions (for $k \geqslant 3$) of (roughly) equal size, and *list* the $2^{n/k}$ variable assignments for each partition. From these $k2^{n/k}$ assignments, we build a graph with weights on its nodes and edges, arguing that a optimum weight k-clique in the graph corresponds to a optimum solution to the original instance. The weights are eliminated using a polynomial reduction, and a fast k-clique algorithm on undirected graphs yields the improvement over 2^n. To get a $(1+\varepsilon)$-approximation when constraints have arbitrary weights, we can adapt results concerning approximate all pairs shortest paths [30] for our purposes.

We will also investigate the possibility of efficient split-and-list algorithms for more general problems such as SAT and MAX-LIN-2 (satisfying a maximum number of linear equations modulo 2). In particular, we will demonstrate some connections between this question and problems in high-dimensional geometry. For example, if a furthest pair out of n d-dimensional points in ℓ_1 norm can be found faster than its known solutions (say, in $O(poly(d) \cdot n^{2-\varepsilon})$ time), then there exists a $(2-\varepsilon)^n$ split-and-list algorithm for MAX-LIN-2.

1.2. Notation

$\mathbb{Z}^+$ and $\mathbb{R}^+$ denote the set of positive integers and the set of positive real numbers, respectively.

Let $V = \{x_1, \ldots, x_n\}$ be a set of variables over (finite) domains $D_1, \ldots, D_n$, respectively. A *k-constraint* on V is defined as a function $c : D_1 \times \cdots \times D_n \to \{0, 1\}$, where c only depends on k variables of V (one might say c is a *k-junta*). For a k-constraint c, define $\mathsf{vars}(c) \subseteq V$ to be this k-set of variables. Partial assignments a to variables of V are given by a sequence of assignments $x_{i_1} := v_2, x_{i_2} := v_2, \ldots, x_{i_k} := v_k$, where $i_j \in [n]$ and $v_{i_j} \in D_{i_j}$. A partial assignment *a satisfies* a constraint c if $\mathsf{vars}(c)$ is a subset of the variables appearing in a, and $c(a) = 1$ (the restriction of c to the variables in a evaluates to 1, on the variable assignments given by a).

Given a set S, $S^{m \times n}$ is the set of $m \times n$ matrices with entries taken from S.

Throughout, ω refers to the smallest real number such that for all $\varepsilon > 0$, matrix multiplication over a ring can be performed in $O(n^{\omega+\varepsilon})$ time. Note that $\omega < 2.376$, cf. [7]. We will discuss three types of matrix product in the paper; unless otherwise specified, the default is matrix product over the ring currently under discussion. The other two are the distance product ($\otimes_d$) on $\mathbb{Z} \cup \{-\infty, \infty\}$, and Boolean matrix product ($\otimes_b$) on 0–1 matrices. Let $A, B \in (\mathbb{Z} \cup \{-\infty, \infty\})^{n \times n}$. $A \otimes_d B$ is matrix product over the (min, +)-semiring; that is, the usual + in matrix product is replaced with the min operator, and $\times$ is replaced with addition. When A and B are 0–1 matrices, the Boolean product $\otimes_b$ replaces + with $\vee$ (OR) and $\times$ with $\wedge$ (AND).

2. Fast *k*-clique detecting and counting

We briefly review an algorithm [18] for detecting if a graph has a k-clique in less than n^k steps.

Theorem 1 (*Nesetril and Poljak [18]*). *Let $r \in \mathbb{Z}^+$. Then $3r$-clique on undirected graphs is solvable in $O(n^{\omega r})$ time.*

[1] The $\tilde{O}$ suppresses polynomial factors.

Proof. First consider $k = 3$. Given $G = (V, E)$ with $n = |V|$, let $A(G)$ be its adjacency matrix. Recall that $\mathrm{tr}(M)$, the trace of a matrix M, is the sum of the diagonal entries. $\mathrm{tr}(A(G)^3)$ is computable in two matrix multiplications, and it is easy to see that $\mathrm{tr}(A(G)^3)$ is non-zero if and only if there is a triangle in G. For $3r$-cliques when $r > 1$, build a graph $G_r = (V_r, E_r)$ where V_r is the collection of all r-cliques in G, and $E_r = \{\{c_1, c_2\} : c_1, c_2 \in V_r,\ c_1 \cup c_2 \text{ is a } 2r\text{-clique in } G\}$. Observe that each triangle in G_r corresponds to a unique $3r$-clique in G. Therefore $\mathrm{tr}(A(G_r)^3) \neq 0$ if and only if there is a $3r$-clique in G, which is determined in $O(n^{\omega r})$ time. Finding an explicit $3r$-clique given that one exists may be done by using an $O(n^{\omega})$ algorithm for finding witnesses to Boolean matrix product [2]; details omitted. □

In fact, the above approach may be used to *count* the number of k-cliques as well. Let $C_k(G)$ be the set of k-cliques in G, and G_r be as defined in the previous proof.

Proposition 1. $\mathrm{tr}(A(G_r)^3) = 6|C_{3r}(G)|$.

Proof. In $\mathrm{tr}(A(G)^3)$, each triangle $\{v_i, v_j, v_k\}$ is counted once for each vertex v (say, v_i) in the triangle, times the two paths traversing the triangle starting from that v (for v_i, they are $v_i \to v_j \to v_k \to v_i$ and $v_i \to v_k \to v_j \to v_i$). Similar reasoning shows that each $3r$-clique is counted six times in $\mathrm{tr}(A(G_r)^3)$. □

3. Algorithm for 2-CSP optimization

Let us explicitly define the problem we will tackle, in its full generality.

Problem COUNT-2-CSP:

Input: A set of m 1-constraints and 2-constraints C on n variables $x_1, \ldots, x_n$ with domains of size $d_1, \ldots, d_n$ (respectively), and a parameter $N \in \{0, 1, \ldots, m\}$.

Output: The number A of variable assignments ($0 \leqslant A \leqslant \prod_{i=1}^{n} d_i$) such that exactly N constraints of C are satisfied.

For $k, n \in \mathbb{Z}^+$ with $k \leqslant n$, let $\kappa(k, n) \in \mathbb{R}$ be the smallest real number such that the number of k-cliques on n-node undirected graphs can be found in $O(n^{\kappa(k,n)})$ time. One may think of $\kappa(k, n)$ as the "k-clique exponent", similar to the matrix multiplication exponent ω. Note that Theorem 1 implies that $\kappa(k, n) \leqslant \omega \cdot k/3$, for all constant k. (We include n as a parameter in κ to account for the possibility that k is a function of n.) For simplicity, let us assume that $|d_i|$ is the same for all $i = 1, \ldots, n$, and is equal to d.

Theorem 2. *Let $k(n) \geqslant 3$ be monotone non-decreasing and time-constructible. Then COUNT-2-CSP is in $O\left(\binom{N+\binom{k(n)}{2}-1}{\binom{k(n)}{2}-1}\left[k(n)d^{n/k(n)}\right]^{\kappa(k(n),n)}\right)$ time, where m is the number of constraints, n is the number of variables, and d is the domain size.*

Corollary 1. *The number of optima for* MAX-2-SAT *and* MAX-CUT *instances can be determined in $O(m^3 1.732^n)$ time, and an optimal assignment can be found in $O(nm^3 1.732^n)$ time.*

Proof of Corollary 1. Set $d = 2$ and $k = 3$. Search for the largest $N \in [m]$ such that the number of assignments a satisfying N constraints is non-zero and return a. This incurs an extra $O(m)$ factor. An explicit assignment can be found using self-reducibility, increasing the runtime by an $O(n)$ factor. □

Proof of Theorem 2. We reduce the problem to counting k-cliques in a large graph. Assume w.l.o.g. that n is divisible by k. Let C be a given instance. Arbitrarily partition the n variables of C into sets $P_1, P_2, \ldots, P_k$ with n/k variables each. For each P_i, make a list L_i of all $d^{n/k}$ assignments to the variables of P_i.

Step 1: *Delegating responsibility.*

Certain partitions (or pairs of partitions) will be responsible for satisfying certain constraints of C. Let $[k] = \{1, \ldots, k\}$, and $\binom{k}{2}$ denote the collection of 2-sets from $[k]$. We define a *responsibility map* $r : C \to \left([k] \cup \binom{k}{2}\right)$ from constraints to partitions and 2-sets of partitions: $r(c) := i \in [k]$, if $\mathsf{vars}(c) \subseteq P_i$, and $r(c) := \{i, j\} \in \binom{k}{2}$, if $\mathsf{vars}(c) \cap P_i \neq \emptyset$ and $\mathsf{vars}(c) \cap P_j \neq \emptyset$. Observe that r is well-defined assuming c is dependent on at most two variables ($|\mathsf{vars}(c)| \leqslant 2$).

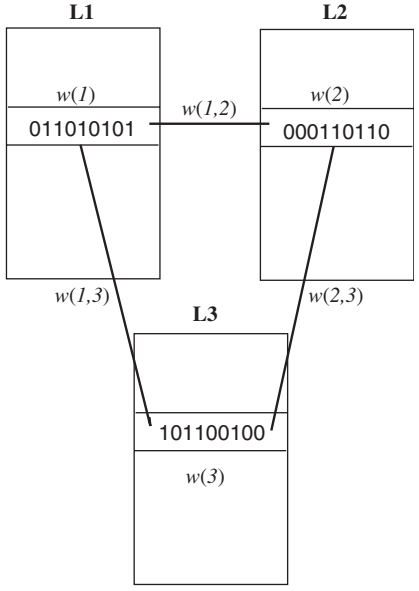
Step 2: *Weighting accordingly.*

Next, we will consider the $L_1, \ldots, L_k$ as a weighted k-partite complete graph $G = (V, E)$, having $d^{n/k}$ nodes per partition. For a vertex $v \in L_i$, let a_v denote the partial assignment to which v refers. For a partial assignment a, $c(a)$ is the value of c when a is assigned to its variables. The weight function w for G is on nodes and edges of G, and is defined:

- $w(v) := |\{r(c) = i \ : \ c \in C, v \in L_i, c(a_v) = 1\}|$,
- $w(\{u, v\}) := |\{r(c) = \{i, j\} \ : \ c \in C, u \in L_i, v \in L_j, c(a_u\, a_v) = 1\}|$.

(Assuming the variables of $\mathsf{vars}(c)$ are assigned in a, $c(a)$ is well-defined; this is the case, by definition of r.) In general, $w(t)$ tells the number of $c \in C$ for which (a) t is in a partition/pair of partitions responsible for c, and (b) the partial assignment that t denotes satisfies c.[2]

Let $K = \{v_1, \ldots, v_k\}$ be a k-clique in G. Define $w(K) := \sum_{i=1}^{k} w(v_k) + \sum_{\{i,j\} \in \binom{k}{2}} w(\{v_i, v_j\})$, i.e. the weight of all nodes and edges involved in K. When $k = 3$ and $d = 2$, G resembles the picture below, for $1 \in L_1, 2 \in L_2, 3 \in L_3$.



Claim. *The number of k-cliques of weight N in G is equal to the number of assignments satisfying exactly N constraints in C.*

Proof. Let a be an assignment to all n variables of C, and suppose a satisfies exactly N constraints of C. Clearly, there exist unique $v_i \in L_i$ for $i = 1, \ldots, k$ such that $a = a_{v_1} a_{v_2} \cdots a_{v_k}$, i.e. a corresponds to a unique clique $K_a = \{v_1, \ldots, v_k\}$ in G. We have that $w(K_a)$ equals $\sum_{i=1}^{k} |\{r(c) = i : v \in L_i, c(a_v) = 1\}| + \sum_{\{i,j\} \in \binom{k}{2}} |\{r(c) = \{i, j\} : u \in L_i, v \in L_j, c(a_u\, a_v) = 1\}|$. That is, $w(K_a)$ counts the number of $c \in C$ that are satisfied by a, such that either $r(c) = i \in [k]$ for some i, or $r(c) = \{i, j\} \in \binom{k}{2}$ for some i, j. But as we argued above, $r(c)$ is well-defined over all $c \in C$, therefore $w(K_a)$ is precisely the number of constraints satisfied by a. As there is a 1-to-1 correspondence between k-cliques in G and assignments to C, and as k-cliques with N weight correspond to assignments satisfying N constraints, the claim is proven. □

[2] **Example.** For MAX-CUT, the formulation is especially simple: the $v \in L_i$ denote all $2^{n/k}$ possible cuts with a distinct "left" and "right" side, on the subgraph of n/k vertices P_i, $w(v)$ is the number of edges crossing the "sub-cut" defined by v, and $w(\{u, v\})$ is the number of edges crossing from one side (say, "left") of u to the opposite side (say, "right") of v.

Step 3: *Reduction from weighted to unweighted graphs.*

There is a slight problem: we want to count k-cliques in the above, but the above method for counting only works on unweighted graphs. We can remove this difficulty and tack on a multiplicative factor that is polynomial in $N \leqslant m$ but exponential in k. Consider the $\binom{k}{2}$-tuples $(i_{1,2}, i_{1,3}, \ldots, i_{k-1,k})$ where $i_{j,l} \in [N]$, and $i_{1,2} + i_{1,3} + \cdots + i_{k-1,k} = N$. For each tuple, construct a unweighted graph $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ where edges are placed according to the rules:

- If $j = 1$ and $l = 2$, put $\{v_1, v_2\}$ in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ iff $w(v_1) + w(v_2) + w(\{v_1, v_2\}) = i_{1,2}$,
- If $j = 1$ and $l > 2$, put $\{v_1, v_l\}$ in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ iff $w(v_l) + w(\{v_j, v_l\}) = i_{1,l}$,
- If $j > 1$, put $\{v_j, v_l\}$ in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ iff $w(\{v_j, v_l\}) = i_{j,l}$.

(Note $w(v_j)$, $w(\{v_j, v_l\}) \in [m]$; this bounds the possible $i_{j,l}$ values.) Then, for each k-clique $K = \{v_1, \ldots, v_k\}$ in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ (it takes $O([kd^{n/k}]^{\kappa(k)})$ time to count them), a short verification shows that N equals $w(K)$. That is, each k-clique counted in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ is a k-clique of weight N in G. Moreover, each distinct $G_{(i_{1,2},\ldots,i_{k-1,k})}$ represents a distinct collection of weight-N cliques in G. Hence the total sum of k-clique counts over all $G_{(i_{1,2},\ldots,i_{k-1,k})}$ is the number of weight-N k-cliques in G. The possible $\binom{k}{2}$-tuples correspond to all non-negative integral solutions to $x_1 + x_2 + \cdots + x_{\binom{k}{2}} = N$, which is $\binom{N+\binom{k}{2}-1}{\binom{k}{2}-1}$. A list of these solutions may be formed in such a way that each solution appears exactly once, with $O(1)$ (amortized) time to generate each one [24].

4. General remarks

4.1. On triangles and matrix multiplication

The current $O(n^{3-\varepsilon})$ matrix multiplication algorithms only begin to outperform Gaussian elimination (in practice) for very large cases. This coincides nicely with the fact that the size of our matrices are exponential in n. Still, it would be quite desirable to find a more *practical* algorithm for detecting triangles in $O(n^{3-\varepsilon})$ time, but this has been an open problem since the 1970s [15]. We can in fact show that if one only wishes to detect a k-clique quickly, it suffices to matrix multiply quickly *over GF*(2). To us, this gives some hope that triangles can be found more quickly, as GF(2) is the "simplest" possible field and matrix multiplication could potentially be easier over it. We prove the result for triangles; the generalization to k-clique follows from the reduction in Theorem 1.

Theorem 3. *If $n \times n$ matrices can be multiplied over $GF(2)$ in $O(n^c)$ time, then there is a randomized algorithm for detecting if a graph has a triangle, running in $O(n^c \log n)$ time and succeeding with high probability.*

Proof. Adi Shamir (unpublished) proposed a method for reducing Boolean matrix product ($\otimes_b$) to GF(2) matrix product ($\otimes_2$). Given two 0–1 matrices A and B to be Boolean-multiplied, randomly change each 1 in A to 0 with probability $\frac{1}{2}$, getting a matrix A'. Then compute $A' \otimes_2 B$; it turns out that $(A' \otimes_2 B)[i, j] = (A \otimes_b B)[i, j]$ with probability $\frac{1}{2}$, for all i, j. Creating $k \geqslant \log(n^2/\varepsilon)$ different A's (call them $A'_1, \ldots, A'_k$), we have that with probability $1 - \varepsilon$

$$(A'_1 \otimes_2 B)[i, j] \vee (A'_2 \otimes_2 B)[i, j] \vee \cdots \vee (A'_k \otimes_2 B)[i, j] = (A \otimes_b B)[i, j]$$

holds for all i, j.

This motivates the following. Similar to [3], randomly color each vertex of G with an element from $\{1, 2, 3\}$, removing all edges connecting nodes with the same color. There is still a triangle in this graph with constant probability, if G has one. (Observe in our split-and-list algorithm, the graph already has this tripartite structure when $k = 3$, so we need not perform this step there.) Say a vertex v is in the i-partition if its color is i. Now orient the edges between the 1-partition and 2-partition to point from the 1-partition to the 2-partition; do similarly for edges from 2 to 3, and edges from 3 to 1. Make matrices $A_{i,j}$ representing connections between nodes from the i-partition to the j-partition: $A_{i,j}[x, y] = 1$ if there is an edge from the xth node in the i-partition to the yth node in the jth partition; $A_{i,j}[x, y] = 0$ otherwise. The theorem follows from the fact that (with constant probability) there is an i such that $(A_{1,3} \otimes_b A_{2,3} \otimes_b A_{3,1})[i, i] = 1$ if and only if there is a triangle in G. □

4.2. The arbitrary weight case

We may also employ our main algorithm to get a $(1+\varepsilon)$-approximation to MAX-2-CSP and MIN-2-CSP with arbitrary weights on the constraints. The approximation will have similar runtime to the exact algorithm in the unweighted case. Formally, the arbitrary weight problem is

Problem OPT-WEIGHT-2-CSP:

Input: A 2-CSP instance with $C = \{c_1, \ldots, c_m\}$, n variables, and weight function $w : C \to \mathbb{Z}^+$.

Output: An assignment a such that $\sum_{i \in \{j \,:\, c_j(a)=1\}} w(c_i)$ is minimal/maximal.

If the constraints have weights describable in $O(\log m)$ bits, we could simply modify the above algorithm (where the weight of an assignment node is now the sum of weights of clauses) and get runtime $O(poly(m) \cdot 1.732^n)$. When weights are independent of m and n, it is possible to use an approximate all-pairs shortest paths algorithm of [30] to get a $(1+\varepsilon)$-approximation in roughly $O(nm^3 1.732^n)$ time (setting $k = 3$ in Theorem 2). Recall $\otimes_{\mathrm{d}}$ (defined earlier) is the distance product on matrices. If A is the adjacency matrix of a weighted (on edges) graph G, then $\min_{i=1}^{n}(A \otimes_{\mathrm{d}} A \otimes_{\mathrm{d}} A)[i,i]$ gives the length of a smallest triangle in G (and is 0 if there is no triangle in G). Say that C is an a-approximation to D iff for all i, j, $C[i,j] \leqslant D[i,j] \leqslant aC[i,j]$. Then the following theorem of Zwick implies an efficient $(1+\varepsilon)$-approximation to OPT-WEIGHT-2-CSP.

Theorem 4 (*Zwick [30]*). *Let* $A, B \in (\mathbb{Z} \cup \{-\infty, \infty\})^{n \times n}$. $A \otimes_{\mathrm{d}} B$ *has a* $(1+\varepsilon)$*-approximation computable in* $O((n^\omega/\varepsilon)\log(W/\varepsilon))$ *time, where*

$W = \max\{A[i,j], B[i,j] \,:\, i, j \in [n]\}$.

(The minimum case is obvious; to get the maximum case, just negate all constraints.)

4.3. Finding a minimum graph bisection

Our algorithm can be easily extended to solve other optimization problems on graphs that have additional constraints on the solution space. We illustrate this point with the problem of MIN BISECTION: *given a graph* $G = (V, E)$ *on* $2k$ *nodes, find a partition of* V *into sets* S *and* T *such that* $|S| = |T| = k$ *and the number of edges crossing the cut* (S, T) *is minimized*. To solve this problem, we add another layer of polynomial time reduction.

Given a graph $G = (V, E)$ with $n = |V|$, construct a corresponding weighted tripartite graph with $2^{n/3}$ nodes per partition, as in the previous algorithm.

Over all triples $(i_1, i_2, i_3) \in [n/3]^3$ such that $i + j + k = n/2$:

For $j = 1, 2, 3$, delete nodes from the jth partition which do not have exactly i_j 1's in their partial assignment. On this reduced graph, find a *minimum* weight triangle using the previous algorithm, where the weighting scheme is the same as for MAX-CUT.

Take the best assignment found over all $O(n^3)$ triples. Now, any minimum weight triangle found corresponds to an minimum vertex cut with exactly $n/2$ vertices on both sides of the cut, and we find the best possible assignment over all possible ways this could happen. The resulting algorithm runs in $O(n^3 m^3 2^{\omega n/3})$ time.

5. Further directions

Is it possible to solve general problems like SAT much faster than the trivial algorithm, using a "split-and-list" method akin to the above? Depending on one's outlook, the results below may be viewed as lower bounds on solving certain high-dimensional problems, or they may be promising signs that much better algorithms exist for general NP-complete problems, using split-and-list methods.

5.1. Cooperative subset queries and SAT

Usually in query problems, one considers a database D of objects, and an adversarial list of queries $q_1, \ldots, q_k$ about D. Such a model often leads to very difficult problems, in particular the *subset query problem*: given a database D of sets $S_1, \ldots, S_k$ over a universe U, build a space-efficient data structure to answer (time-efficiently) queries of the

form "is $q \in 2^U$ a subset of some $S_j \in D$?". This problem is a special case of the *partial matching problem*; that is, supporting queries with wildcards (e.g. "S**RCH") in a database of strings. Non-trivial algorithms exist [22,5], but all known solutions for the problem require either $\Omega(|D|)$ query time (search the whole database) or $2^{\Omega(|U|)}$ space (store all possible subsets) in general. Our *cooperative* version of the subset query problem is the following:

Given two databases D_1 and D_2 of subsets over U, find $s_1 \in D_1$ and $s_2 \in D_2$ such that $s_1 \subseteq s_2$.

That is, we merely want to determine if one of the given queries has a *yes* answer. Intuitively, the cooperative version ought to be significantly easier than the general one. Of course, it admits a trivial $O(|U| \cdot |D_1| \cdot |D_2|)$ time algorithm, but if this solution can be significantly improved, then SAT in conjunctive normal form can be solved faster than the 2^n bound. Note the current best SAT algorithm is randomized, and runs in $2^{n-\Omega(n/\log n)}$ time [27].

Theorem 5. *Let f be time constructible. If the cooperative subset query problem with $d = |U|$ and $k = \max\{|D_1|, |D_2|\}$ is solvable in $\tilde{O}(f(d)k^{2-\varepsilon})$ time, then CNF-SAT is in $\tilde{O}(f(m)2^{(1-\varepsilon/2)n})$ time, where m is the number of clauses and n is the number of variables.*

Proof. Without loss of generality, assume n is divisible by 2. Recall CNF-SAT is a constraint satisfaction problem where constraints are arbitrary *clauses*, i.e. ORs of negated and non-negated variables. Suppose the clauses are indexed $c_1, \ldots, c_m$. Arbitrarily partition the n variables into two sets P_1, P_2 of size $n/2$ each, and build respective lists L_1, L_2 of $2^{n/2}$ assignments for the P_i. Next, we build two collection of sets C_1 and C_2. Associate with each $p \in L_1$ a set $S_p \in C_1$, defined by the condition

$$c_j \in S_p \text{ if and only if } p \text{ does not satisfy } c_j.$$

For $p' \in L_2$, make a set $S_{p'} \in C_2$, defined by

$$c_j \in S_{p'} \text{ if and only if } p' \text{ satisfies } c_j.$$

Now, suppose we determine if there is $S_p \in C_1$ and $S_{p'} \in C_2$ whereby $S_p \subseteq S_{p'}$. It is easy to see that the collection of clauses $\{c_1, \ldots, c_m\}$ is satisfiable if and only if this cooperative subset query instance has a "yes" answer. □

5.2. Furthest pair of points in ℓ_1 and MAX-LIN

Recall that the ℓ_1 distance between two d-dimensional points $(x_1, \ldots, x_d)$ and $(y_1, \ldots, y_d)$ is $|x - y|_1 = \sum_{i=1}^{d} |x_i - y_i|$. A classic high-dimensional geometry problem is ℓ_1-Furthest-Pair, or ℓ_1-Diameter: given a set $S \subseteq \mathbb{R}^d$ of n points, find a pair $x, y \in S$ for which $|x - y|_1$ is maximized. (It may be seen as a cooperative version of a problem where, given a query, one reports points furthest from it.) Beyond the naïve $O(dn^2)$ solution, an isometric embedding from ℓ_1 in d dimensions to ℓ_∞ in 2^d dimensions yields a $O(2^d n)$ time algorithm. We will prove if ℓ_1-Furthest-Pair can be solved in (for example) $O(2^{o(d)}n^{2-\varepsilon})$, then there is a better algorithm for MAX-LIN-2. Recall the MAX-LIN-2 problem is, given a system of m linear equations over n variables in GF(2), to find a variable assignment that maximizes the number of equations satisfied.

Theorem 6. *If ℓ_1-Furthest-Pair (of n points in d-dimensions) is in $O(f(d)n^{2-\varepsilon})$ time, then MAX-LIN-2 is in $O(f(m+1)2^{(1-\varepsilon/2)n})$ time.*

Proof. Rewrite each GF(2) linear equation as a constraint: an equation $\sum_i a_i x_i = d$ becomes

$$c(x_1, \ldots, x_n) = \left(\sum_i a_i x_i + d + 1\right) \bmod 2,$$

so that $c(a) = 1$ if and only if a satisfies the corresponding equation. Let $c_1, \ldots, c_m$ be the resulting constraints.

As before, split the variables into two $n/2$ sets P_1, P_2, and have two lists of assignments L_1 and L_2. For each c_j, let $c_j|_{P_i}$ be the *restriction* of c_j to the variables of P_i. In the case where $+1$ appears in c_j, add $+1$ to $c_j|_{P_1}$. For example, suppose we had the partitions $P_1 = \{x_1, x_2\}$ and $P_2 = \{x_3, x_4\}$; if $c = \sum_{i=1}^{4} x_i + 1$, then $c|_{P_1} = x_1 + x_2 + 1$, $c|_{P_2} = x_3 + x_4$. Clearly, $c_j(x_1, \ldots, x_n) = c_j|_{P_1}(x_1, \ldots, x_{n/2}) + c_j|_{P_2}(x_{n/2+1}, \ldots, x_n)$. For $i \in \{1, 2\}$ and for all assignments $a \in L_i$, make an $(m+1)$-dimensional point

$$v_a = (c_1|_{P_i}(a), \ldots, c_m|_{P_i}(a), (m+1) \cdot (i-1)).$$

Let v_a and $v_{a'}$ be a furthest pair of points out of these, with respect to ℓ_1 distance.

First, we claim that any furthest pair of points is of the form $\{v_a, v_{a'}\}$, where $a \in L_1$ and $a' \in L_2$. Let $d_1(u, v)$ denote the ℓ_1 distance between u and v. For any v_a and $v_{a'}$ with $a, a' \in L_i$ (for some $i \in \{1, 2\}$), we have that $d_1(v_a, v_{a'}) \leqslant m$, since the last components of v_a and $v_{a'}$ are the same, and the other m components of v_a and $v_{a'}$ are Boolean-valued. For any v_a and $v_{a'}$ with $a \in L_1$ and $a' \in L_2$, $d_1(v_a, v_{a'}) \geqslant m + 1$ (the last component of v_a is 0, and the last component of v_a is $m + 1$). Therefore our claim follows.

Secondly, let $\{v_a, v'_a\}$ be a furthest pair of points with $a \in L_1$ and $a' \in L_2$. We claim that the assignment $(x_1, \ldots, x_n) = (a, a')$ satisfies a maximum number of constraints in the original instance. Over GF(2), addition is equivalent to subtraction, hence

$$\begin{aligned}\max_{a\in L_1,\, a'\in L_2} d_1(v_a, v_{a'}) &= (m+1) + \max \sum_{i=1}^{m} |\, c_i|_{P_1}(a) - c_i|_{P_2}(a)\,| \\ &= (m+1) + \max \sum_{i=1}^{m} c_i|_{P_1}(a) + c_i|_{P_2}(a) \\ &= (m+1) + \max_{a\in L_1,\, a'\in L_2} \sum_{i=1}^{m} c_i(x_1, \ldots, x_n). \qquad \square\end{aligned}$$

6. Conclusion

We have presented the first improved exponential algorithm (in n) for solving and counting solutions to 2-constraint optimization. The techniques employed here are general enough to be possibly employed on a variety of problems. We have also established interesting connections between the complexity of some efficiently solvable problems and some hard problems (matrix multiplication and counting 2-CSP optima, furthest pair of points and MAX-LIN-2, subset queries and SAT). Several interesting questions are left open by our work.

- How does one extend our results for k-CSPs, when $k \geqslant 3$? A straightforward generalization to 3-CSPs results in a weighted hypergraph of edges and 3-edges. It is conjectured that matrix multiplication can be done in $O(n^{2+o(1)})$ time, and in our investigation of 3-CSPs, it appears a $2^{3n/4}$ bound might be possible. We therefore conjecture that for all $k \geqslant 2$, MAX-k-SAT is in $\tilde{O}(2^{n(1-1/(k+1))})$ time.
- Are there fast algorithms for 2-CSP optimization using only polynomial space?
- Is there a randomized k-clique detection algorithm running in $O(n^{3-\varepsilon})$? (Is there merely a good one that does not use matrix multiplication?)
- What other problems are amenable to the split-and-list approach?

Acknowledgements

I am indebted to my advisor Manuel Blum, to Nikhil Bansal, and to Virginia Vassilevska, for enlightening conversations on this work.

References

[1] J. Alber, J. Gramm, R. Niedermeier, Faster exact algorithms for hard problems: a parameterized point of view, Discrete Math. 229 (2001) 3–27.
[2] N. Alon, M. Naor, Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions, Algorithmica 16 (1996) 434–449.
[3] N. Alon, R. Yuster, U. Zwick, Color-coding, JACM 42 (4) (1995) 844–856.
[4] N. Bansal, V. Raman, Upper bounds for Max-Sat: further improved, in: Proc. ISAAC, Lecture Notes in Computer Science, Vol. 1741, Springer, Berlin, 1999, pp. 247–258.
[5] M. Charikar, P. Indyk, R. Panigrahy, New algorithms for subset query, partial match, orthogonal range searching, and related problems, in: Proc. of ICALP, Lecture Notes in Computer Science, Vol. 2380, Springer, Berlin, 2002, pp. 451–462.
[6] J. Chen, I. Kanj, Improved exact algorithms for MAX-SAT, in: Proc. of LATIN, Lecture Notes in Computer Science, Vol. 2286, Springer, Berlin, 2002, pp. 341–355.
[7] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, JSC 9 (3) (1990) 251–280.

[8] E. Dantsin, M. Gavrilovich, E.A. Hirsch, B. Konev, MAX-SAT approximation beyond the limits of polynomial-time approximation, Ann. Pure Appl. Logic 113 (1–3) (2001) 81–94.
[9] J. Gramm, E.A. Hirsch, R. Niedermeier, P. Rossmanith, Worst-case upper bounds for MAX-2-SAT with application to MAX-CUT, Discrete Appl. Math. 130 (2) (2003) 139–155.
[10] J. Gramm, R. Niedermeier, Faster exact solutions for Max2Sat, in: Proc. of CIAC, Lecture Notes in Computer Science, Vol. 1767, Springer, Berlin, 2000, pp. 174–186.
[11] E.A. Hirsch. A $2^{m/4}$-time algorithm for MAX-2-SAT: corrected version, Electronic Colloquium on Computational Complexity Report TR99-036, 2000.
[12] E.A. Hirsch, Worst-case study of local search for MAX-k-SAT, Discrete Appl. Math. 130 (2) (2003) 173–184.
[13] T. Hofmeister, U. Schöning, R. Schuler, O. Watanabe, A probabilistic 3-SAT algorithm further improved, in: Proc. of STACS, Lecture Notes in Computer Science, Vol. 2285, Springer, Berlin, 2002, pp. 192–202.
[14] E. Horowitz, S. Sahni, Computing partitions with applications to the knapsack problem, JACM 21 (1974) 277–292.
[15] A. Itai, M. Rodeh, Finding a minimum circuit in a graph, SIAM J. Comput. 7 (4) (1978) 413–423.
[16] S.S. Fedin, A.S. Kulikov, A $2^{|E|/4}$-time algorithm for MAX-CUT, J. Math. Sci. 126 (3) (2005) 1200–1204.
[17] M. Mahajan, V. Raman, Parameterizing above guaranteed values: MAXSAT and MAXCUT, J. Algorithms 31 (2) (1999) 335–354.
[18] J. Nesetril, S. Poljak, On the complexity of the subgraph problem, Comment. Math. Univ. Carolin. 26 (2) (1985) 415–419.
[19] R. Niedermeier, P. Rossmanith, New upper bounds for maximum satisfiability, J. Algorithms 26 (2000) 63–88.
[20] R. Paturi, P. Pudlak, M.E. Saks, F. Zane, An improved exponential-time algorithm for k-SAT, JACM 52 (3) (2005) 337–364.
[21] V. Raman, B. Ravikumar, S. Srinivasa Rao, A simplified NP-complete MAXSAT problem, Inform. Process. Lett. 65 (1998) 1–6.
[22] R. Rivest, Partial match retrieval algorithms, SIAM J. Comput. 5 (1976) 19–50.
[23] M. Robson, Algorithms for maximum independent sets, J. Algorithms 7 (3) (1986) 425–440.
[24] F. Ruskey. Simple combinatorial Gray codes constructed by reversing sublists, in: Proc. of Interat. Symp. on Algorithms and Computation, Lecture Notes in Computer Science, Vol. 762, Springer, Berlin, 1993, pp. 201–208.
[25] U. Schöning, A probabilistic algorithm for k-SAT and constraint satisfaction problems, in: Proc. of the 40th IEEE FOCS, 1999, pp. 410–414.
[26] R. Schroeppel, A. Shamir, A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems, SIAM J. Comput. 10 (3) (1981) 456–464.
[27] R. Schuler, An algorithm for the satisfiability problem of formulas in conjunctive normal form, J. Algorithms 54 (1) (2005) 40–44.
[28] A. Scott, G. Sorkin, Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances, in: Proc. of RANDOM-APPROX 2003, Lecture Notes in Computer Science, Vol. 2764, Springer, Berlin, 2003, pp. 382–395.
[29] G.J. Woeginger, Exact algorithms for NP-hard problems: a survey, in: Combinatorial Optimization—Eureka! You shrink!, Lecture Notes in Computer Science, Vol. 2570, Springer, Berlin, 2003, pp. 185–207.
[30] U. Zwick, All pairs shortest paths using bridging sets and rectangular matrix multiplication, JACM 49 (3) (2002) 289–317.