Electronic Colloquium on Computational Complexity, Report No. 32 (2004)

A new algorithm for optimal constraint satisfaction and its implications

Ryan Williams[*]
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

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, it is a constant factor improvement in the base of the runtime exponent. In the case where constraints have arbitrary weights, there is a $(1 + \epsilon)$-approximation with roughly the same runtime, modulo polynomial factors. Our algorithm may be used to 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. This is the first known algorithm solving MAX-2-SAT and MAX-CUT in provably less than c^n steps in the worst case, for some $c < 2$; similar new results are obtained for related problems. Our main construction may also be used to show that any 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. As a corollary, we prove that an $n^{o(k)}$-time k-clique algorithm implies SNP $\subseteq$ DTIME$[2^{o(n)}]$, for any $k(n) \in o(\sqrt{n/\log n})$.

Further extensions of our technique yield connections between the complexity of some (polynomial time) high dimensional geometry problems and that of some general 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 $(2-\epsilon)^n$ algorithm for MAX-LIN. Such results may be construed as either lower bounds on these high-dimensional problems, or hope that better algorithms exist for more general NP-hard problems.

1 Introduction

The extent to which NP-hard problems are indeed hard to solve is still 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 significantly better algorithms; one popular example in the literature is MAX-2-SAT.

There has been high recent theoretical interest in finding a way to solve MAX-2-SAT that would run in $O((2 - \epsilon)^n)$ steps on all instances, for some $\epsilon > 0$. Unlike other problems such as Vertex

[*]Email: `ryanw@cs.cmu.edu`. Supported by a NSF Graduate Research Fellowship and the NSF ALADDIN Center (`http://www.aladdin.cs.cmu.edu/`).

ISSN 1433-8092

Cover and k-SAT, where a sophisticated analysis of branch-and-bound techniques (or random choice of assignments) sufficed for improving the naive bounds (*e.g.* [33, 23, 29]), MAX-SAT has been surprisingly difficult to attack. Previous work has only been able to show either a $(2-\epsilon)^m$ time bound, where m is the number of clauses, or an approximation scheme running in $(2-\epsilon)^n$ [13, 12] (but $\epsilon \to 0$ as the 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 $(2-\epsilon)^m$ 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 [11] (so for $m/n > 5$, this is no better than 2^n). This result followed a long line of previous papers on bounds in terms of m [14, 26, 10, 6, 4, 22, 23]; a similar line has formed for MAX-CUT [38, 21]. A partial answer to the problem (of a $(2-\epsilon)^n$ algorithm for MAX-SAT) was recently given by [39], who proved that *sparse instances* of MAX-k-SAT can be solved in $(2-\epsilon)^n$. That is, for every constant k and constant Δ there is a constant $c_{k,\Delta}$ such that MAX-k-SAT instances with at most Δn clauses are solvable in $(c_{k,\Delta})^n$ steps (however, $c_{k,\Delta} \to 2$ as $\Delta \to \infty$ or $k \to \infty$). [31] 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 still appeared distant from an improved exponential algorithm. As a result, several researchers (*e.g.* [31, 1, 12, 40]) explicitly proposed a $(2-\epsilon)^n$ algorithm for MAX-2-SAT (and/or MAX-CUT) as a benchmark open problem in the area.

1.1 Outline of our approach: Split and List.

Previous exact algorithms in the literature for MAX-2-SAT (indeed, most known algorithms for exactly solving NP-hard problems) involve either a case analysis of a branch-and-bound strategy [11], repeated random choice of assignments [29], or local search [35]. Our design is a major departure from these approaches; its pre-processing step resembles earlier algorithms from the 70's, and a recent (toy) algorithm of Pudlák [30, 16, 36]. We *split* the set of n variables into k partitions (for $k \geq 3$) of (roughly) equal size, and *list* the (roughly) $2^{n/k}$ assignments for the variables of 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 counting algorithm on undirected graphs yields the improvement over 2^n. To get a $(1+\epsilon)$-approximation when constraints have arbitrary weights, we can adapt results concerning approximate all pairs shortest paths [41] for our purposes.

Our reduction can also be used to prove stronger complexity results concerning the fixed-parameter tractability of k-clique; cf. Section 4.2. Finally, we 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 in 0-1 variables). 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 $O(poly(d) \cdot n^{2-\epsilon})$ time), then there exists a $(2-\epsilon)^n$ split-and-list algorithm for MAX-LIN-2.

1.2 Notation.

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

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

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 $\epsilon > 0$, matrix multiplication over a ring can be performed in $O(n^{\omega+\epsilon})$ time. 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 [25, 3] for detecting if a graph has a k-clique in less than n^k steps.

Theorem 2.1 *([25]) Let $r \in \mathbb{Z}^+$. Then $3r$-clique on undirected graphs is solvable in $O(n^{\omega r})$ time.*

Proof. First consider $k = 3$. Given $G = (V, E)$ with $n = |V|$, let $A(G)$ be its adjacency matrix. Recall that $tr(M)$, the trace of a matrix M, is the sum of the diagonal entries. $tr(A(G)^3)$ is computable in two matrix multiplications; it is easy to see that $tr(A(G)^3)$ 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 $tr(A(G_r)^3) \neq 0$ 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 2.1 $tr(A(G_r)^3) = 6|C_{3r}(G)|$.

Proof. In $tr(A(G)^3)$, 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 $v_i \to v_k \to v_k \to v_i$). Similar reasoning shows that each $3r$-clique is counted six times in $tr(A(G_r)^3)$. □

3 Algorithm for 2-CSP optimization.

Let us explicitly define the problem our optimization algorithm 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 ($0 \leq A \leq \prod_{i=1}^{n} d_i$) such that exactly N constraints of C are satisfied.

Let $\kappa(k)$ be such that the number of k-cliques on n-node undirected graphs can be found in $O(n^{\kappa(k)})$ time. (One may think of $\kappa(k)$ as the "k-clique exponent", analogous to the mysterious matrix multiplication exponent ω. But note also that k may be a function of n, in which case $\kappa(k)$ might be as well.) Assume for simplicity that $|d_i|$ is the same for all $i = 1, \ldots, n$, and is equal to d.

Theorem 3.1 *Let $k(n) \geq 3$ be monotone non-decreasing and time-constructible. Then*

COUNT-2-CSP *is in* $O\left(\binom{N+\binom{k(n)}{2}-1}{\binom{k(n)}{2}-1}[k(n)d^{n/k(n)}]^{\kappa(k(n))}\right)$ *time,*

where m is the number of constraints, n is the number of variables, and d is the domain size.

Corollary 3.1 MAX-2-SAT *and* MAX-CUT *solutions can be counted in $O(m^3 1.732^n)$ time; an explicit optimal assignment can be found in $O(nm^3 1.732^n)$ time.*

Proof of Corollary 3.1. 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 a $O(n)$ multiplicative factor: assign $x_1 := 1$ in the set of constraints and test if there is still an assignment satisfying N constraints. If yes, keep $x_1 := 1$; try assignments for x_2, x_3, ... similarly. If not, assign $x_1 := 0$; try assignments for x_2, x_3, ... similarly. $\square$

Proof of Theorem 3.1. 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, ..., 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* $r : C \to ([k] \cup \binom{k}{2})$ from constraints to partitions and 2-sets of partitions. Intuitively $r(c) = i$ ($r(c) = \{i, j\}$) means that P_i is (P_i and P_j are) "responsible" for satisfying c.

- $r(c) := i \in [k]$, if $\mathsf{vars}(c) \subseteq P_i$,
- $r(c) := \{i, j\} \in \binom{k}{2}$, if $\mathsf{vars}(c) \cap P_i \neq \varnothing$ and $\mathsf{vars}(c) \cap P_j \neq \varnothing$.

Observe that r is well-defined assuming c is dependent on at most two variables ($|\mathsf{vars}(c)| \leq 2$).

Step 2: Weighting accordingly.

Next, build a weighted graph G with

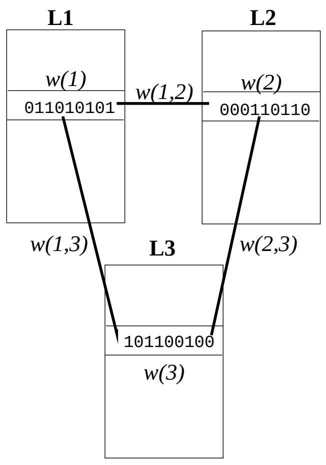
$$V = \bigcup_{i=1}^{k} L_i \text{ and } E = \{\{u, v\} \in V \ : \ u \in L_i, v \in L_j \Rightarrow i \neq j\}.$$

That is, G is a complete k-partite graph, with $d^{n/k}$ nodes per partition. For a vertex v in some list L_i, let a_v denote the partial assignment to which v refers. The weight function w for G is on nodes and edges of G, and is defined:

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

(Given a partial assignment a, "$c(a)$" denotes the value of c when a is assigned to its variables. Assuming all variables in $\mathsf{vars}(c)$ are assigned in a, $c(a)$ is well-defined; for us this is the case, by definition of r.) Generally speaking, $w(t)$ tells the number of constraints $c \in C$ for which (a) t is in a partition/pair of partitions responsible for c, and (b) the partial assignment that t denotes satisfies c.

When $k = 3$ and $d = 2$, G resembles the picture below, for vertices $1 \in L_1$, $2 \in L_2$, $3 \in L_3$.



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.

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

$$w(K_a) = \sum_{i=1}^{k} |\{r(c) = i : v \in L_i, c(a_v) = 1\}| + \sum_{\{i,j\}\in\binom{k}{2}} |\{r(c) = \{i,j\} : u \in L_i, v \in L_j, c(a_u\ a_v) = 1\}|.$$

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 k-cliques with N weight correspond to assignments satisfying N constraints, the claim is proven. $\square$

Step 3: Reduction from weighted to unweighted graphs.

There is a slight difficulty: we want to count k-cliques in the above, but the efficient algorithm for counting k-cliques only works for unweighted graphs. We can remove this difficulty and tack on a multiplicative factor that is polynomial in $N \le m$ 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 graph $G_{(i_{1,2}, i_{1,3}, \ldots, i_{k-1,k})}$ where (unweighted) edges are placed between $v_j \in L_j$ and $v_l \in L_l$ 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 $w(v_j)$ and $w(\{v_j, v_l\})$ have values in $[m]$; this bounds the possible $i_{j,l}$ values.) Then, for each k-clique $K = \{v_1, \ldots, v_k\}$ appearing in the unweighted graph $G_{(i_{1,2}, i_{1,3}, \ldots, i_{k-1,k})}$ (it takes $O([kd^{n/k}]^{\kappa(k)})$ time to count them), it follows that N equals

$$\sum_{\{j,l\}\in\binom{k}{2}} i_{j,l} = [w(v_1) + w(v_2) + w(\{v_1, v_2\})] + \sum_{\{1,l\}\in\binom{k}{2}, l>2} [w(v_l) + w(\{v_j, v_l\})] + \sum_{\{j,l\}\in\binom{k}{2}, j>1} w(\{v_j, v_l\}) = w(K).$$

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, every distinct $G_{(i_{1,2},\ldots,i_{k-1,k})}$ represents a new (disjoint) set of weight N cliques in G. Therefore 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 [34]. □

4 General Remarks.

4.1 On triangles and matrix multiplication in the algorithm.

The current $O(n^{3-\epsilon})$ 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 very desirable to find a practical efficient algorithm which detects if an undirected graph has a triangle in $O(n^{3-\epsilon})$ time; on the other hand, this problem has been open since the 70's [18, 3]. We can in fact show that if one only wishes to detect a k-clique in a graph, 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.) We prove the result for triangles; the generalization to k-clique follows from the reduction in Theorem 2.1.

Theorem 4.1 *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 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 1/2, for all i,j. Creating $k \geq \log(n^2/\epsilon)$ different A's (call them $A'_1, \ldots, A'_k$), we have that with probability $1-\epsilon$

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

holds for all i,j.

This motivates the following procedure. 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.) 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 A complexity-theoretic implication.

[9] showed that if one can efficiently find cliques of size $\log n$ in an n node graph, then a substantially faster deterministic simulation of nondeterministic time is possible.

Theorem 4.2 *([9]) If $(\log n)$-clique $\in P$, then $NTIME[t] \subseteq DTIME[2^{O(t^{1/2} polylog(t))}]$ for any time-constructible function t.*

Define $SE := TIME[2^{o(n)}] = \bigcap_{\epsilon>0} TIME[2^{\epsilon n}]$. SE is typically called *sub-exponential time* [17]. The class SNP is defined in [27] and its set of complete problems includes most well-studied NP-complete problems. One may derive a simple proposition from results of [9] and [17]: if k-clique is *fixed-parameter tractable*, then SAT (and thus, all of SNP) is in sub-exponential time (sub-exponential in both m and n, the number of clauses and variables, respectively).

Proposition 4.1 *(Follows from [9], [17]) If there exists $c > 0$ and some function f such that for all k, k-clique is in $O(n^c f(k))$ time, then $SNP \subseteq SE$.*

Using Theorem 3.1, a significantly weaker hypothesis still yields the above conclusion.

Corollary 4.1 *Let $k(n) \in o(\sqrt{n/\log n})$ be unbounded and time-constructible. If $k(n)$-clique is in $n^{o(k(n))}$ time, then $SNP \subseteq SE$.*

Proof. (Sketch) First, it is known that if Vertex Cover is in $2^{o(n)}$ time (n being the number of vertices) then the consequent holds [17]. Given the assumption, we prove Vertex Cover is quickly solvable using a variant of the COUNT-2-CSP algorithm. Suppose we want to find a vertex cover of size at most s. Essentially the algorithm for COUNT-2-CSP is executed (vertices are partitioned into $k(n)$ sets, all subsets of vertices are listed, etc.) except the reduction to undirected k-clique is run over all k-tuples $(j_1, \ldots, j_k) \in [s]$ such that $\sum_{i=1}^{k} j_i \leq s$. A graph $G_{(j_1,\ldots,j_k)}$ is constructed by removing all $v_i \in L_i$ where the number of 1's in a_{v_i} is *not* j_i; now k-cliques in this graph represent vertex subsets of size $\leq s$. An additional scheme (over all $\binom{k}{2}$ tuples of non-negative integers summing up to m, the number of edges in the Vertex Cover instance) is used to determine how edges are added to $G_{(j_1,\ldots,j_k)}$. The total runtime is bounded from above by $[k2^{n/k}]^{o(k)} \cdot (s+k)^k \cdot (m+k^2)^{k^2}$, which is at most $(\sqrt{n/\log n})^{o(\sqrt{n/\log n})} 2^{o(n)} \cdot (n + \sqrt{n/\log n})^{o(\sqrt{n/\log n})} \cdot (m + n/\log n)^{o(n/\log n)} \leq 2^{o(n)+o(n)} \cdot 2^{o(n)} \cdot 2^{\log(m+n)\cdot o(n/\log n)} \leq 2^{o(n)}$ (note $m \leq n^2$, and tighter upper bounds on the number of tuples will not improve the outcome by much). □

Remark 1 *We could have also used the Max Clique problem in the above; that is, for any unbounded $k(n) \in o(\sqrt{n/\log n})$, an $n^{o(k(n))}$ algorithm for $k(n)$-Clique implies a $2^{o(n)}$ algorithm for* general *Max Clique.*

4.3 On the arbitrary weight case.

We may also employ our main algorithm to get a $(1+\epsilon)$-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 minimized/maximized.

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 $O(poly(m) \cdot 1.732^n)$, as we try all possible weight-sums for the clauses satisfied. When the weights are independent of m and n, it is possible to use components from an approximate all-pairs shortest paths algorithm of [41] to get a $(1+\epsilon)$-approximation to the optimum in roughly $O(nm^3 1.732^n)$ time (setting $k = 3$ in Theorem 3.1).

Recall $\otimes_d$ (defined earlier) is the distance product on matrices. If A is the adjacency matrix of a weighted (on edges) graph G, then $\min_{i=1}^{n}(A \otimes_d A \otimes_d A)[i,i]$ 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, $C[i,j] \leq D[i,j] \leq aC[i,j]$. Then the following theorem implies an efficient $(1+\epsilon)$-approximation to our problem.

Theorem 4.3 *([41]) Let $A, B \in (\mathbb{Z} \cup \{-\infty, \infty\})^{n \times n}$. $A \otimes_d B$ has a $(1+\epsilon)$-approximation computable in $O((n^\omega/\epsilon)\log(W/\epsilon))$ 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.)

4.4 Improvement on the quadratic assignment problem.

The quadratic assignment problem (*QAP*) arises in facility location and scheduling. One has a set of n sites and n facilities; each facility must be built at one of the sites. Between pairs of facilities there are demands that must be met, and between pairs of sites there are distances. Informally speaking, we wish to build facilities with high pairwise demands close to one another. Formally, there are $n \times n$ matrices C and D of costs and demands (respectively), and we wish to find a permutation $\pi \in S_n$ such that $\sum_{i,j} D[i,j]C[\pi(i), \pi(j)]$ is minimized.

In practice, the *QAP* is quite difficult; solving instances with $n > 20$ is state-of-the-art [28]. Hence it is an example where a significant improvement in the theoretical upper bound could have immediate practical impact. Prior to our work, there was no better exact algorithm than the obvious $O(poly(n) \cdot n!)$ one; we can gain a little over the non-trivial bound, since the *QAP* may be construed as a 2-CSP with variable domain n. (Care must be taken so that only permutations have k-cliques in the constructed graph, but this is fairly easy to add into our framework.)

Corollary 4.2 *If the entries of C and D are describable in $O(\log n)$ bits, then* QAP *on n nodes is solvable (and the number of optimal permutations can be counted) in $n^{\omega n/3+O(1)}$ time.*

5 Further Directions.

Is it possible to solve general problems like SAT significantly faster than the trivial algorithm, using a "split-and-list" method akin to the above? Depending on your outlook, the following results may interpreted as lower bounds on solving various high-dimensional problems, or they may be taken to be promising signs that much better algorithms exist for general NP-complete problems, using split-and-list methods. We will give a few (polytime solvable) problems such that if their trivial solutions can be improved time-wise, then there exist $(2-\epsilon)^n$ algorithms for SAT and MAX-LIN.

5.1 Cooperative subset queries and orthogonal vectors

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* ([20], p.557): 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 q of the form: "Is q 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 [32, 5], but all known solutions for the problem still require either $\Omega(|D|)$ query time (search the whole database) or $2^{\Omega(|U|)}$ space (store all the possible subsets) in general. For the subset query problem, [19] recently proved a lower bound of $\Omega(|U|/\log|D|)$ probes in the cell-probe model, assuming a polynomial number of cells of polynomial size. 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, queries are explicitly given to us as part of the input, and all we want is to determine if one of the queries will get an *yes* answer. The cooperative version ought to be significantly easier than the general one– all query information is known to us. Of course, it is trivially solvable in $O(|U| \cdot |D_1| \cdot |D_2|)$ time; try all the possible pairs of sets from D_1 and D_2. Can it be solved much faster? If so, then SAT in conjunctive normal form can be solved much faster than the 2^n bound.

Theorem 5.1 *Let f be time constructible. If the cooperative subset query problem with $d = |U|$ and $k = \max\{|D_1|, |D_2|\}$ is solvable in $\tilde{O}(f(d)k^{2-\epsilon})$ time, then CNF-SAT is in $\tilde{O}(f(m)2^{(1-\epsilon/2)n})$ time, where m is the number of clauses and n is the number of variables.*

Proof. (Sketch) Recall that CNF-SAT is a constraint satisfaction problem where the constraints are arbitrary *clauses*, which are OR's of negated and non-negated variables. Suppose the clauses are indexed $c_1, \ldots, c_m$. Partition variables into two sets P_1, P_2 of size $n/2$ each, making lists L_1, L_2 of the $2^{n/2}$ assignments. Associate with each $p \in L_1$ a set S_p, defined by putting $c_j \in S_p$ iff p does not satisfy c_j. For $p' \in L_2$, make a set $S_{p'}$, defined by putting $c_j \in S_{p'}$ iff p' satisfies c_j. Now determine if there is $S_p \in L_1$ and $S_{p'} \in L_2$ whereby $S_p \subseteq S_{p'}$. It is not hard to show that the set of clauses is satisfiable iff the cooperative subset query instance has a "yes" answer. $\Box$

This is either evidence that the trivial algorithm for the cooperative problem cannot be improved upon significantly (and therefore the *general* subset query problem is also hard), or evidence that SAT can be solved significantly faster than 2^n. (The current best SAT algorithm is randomized, and runs in $2^{n-\Omega(n/\log n)}$ time [37].) Note the cooperative subset query problem is equivalent to the following linear algebra problem, which also has a similar non-cooperative version that has been extensively studied (*e.g.* [8]).

Orthogonal Vectors Problem: Given two lists L_1 and L_2 of m-bit vectors with $|L_i| \leq N$ for $i \in \{1, 2\}$, is there $u \in L_1$ and $v \in L_2$ such that $\langle u, v \rangle = 0$?

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: 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 is sometimes called the *diameter problem* as well. (It may be trivially seen as a cooperative version of a problem where, given a query, one reports points furthest from it.) Of course, trying all pairs takes $O(dn^2)$ time; using an isometric embedding from ℓ_1 in d dimensions to ℓ_∞ in 2^d dimensions, one can find a pair of points with largest ℓ_1 distance in $O(2^d n)$ time. We shall prove if ℓ_1-Furthest-Pair can be solved in (for example) $O(2^{o(d)} n^{2-\epsilon})$, then there is a better algorithm for the general MAX-LIN-2 problem. 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 5.2 *If ℓ_1-Furthest-Pair (of n points in d-dimensions) is in $O(f(d)n^{2-\epsilon})$ time, then MAX-LIN-2 is in $O(f(m+1)2^{(1-\epsilon/2)n})$ time.*

Proof. Rewrite each GF(2) linear equation as a constraint: an equation $\sum_i a_i x_i = d$ becomes $c(x_1, \ldots, x_n) = \sum_i a_i x_i + d + 1$. 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

$$v_a = (\ c_1|_{P_i}(a),\ \ \ldots,\ \ c_m|_{P_i}(a), 2m \cdot i\ )$$

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

Claim: The assignment $(x_1, \ldots, x_n) = (a, a')$ satisfies a maximum number of constraints in the original instance.

Proof. (Sketch) Use the fact that addition in GF(2) is subtraction in GF(2); the last component of v_a ensures that the furthest pair of points consists of one point from L_1 and one from L_2.

$\square$

6 Conclusion.

We have presented the first improved exponential algorithm (in n) for solving and counting solutions to 2-constraint optimization. Our techniques 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). Let us describe a few directions worth following.

• How does one extend our results for k-CSPs, when $k \geq 3$? A straightforward generalization of the above for 3-CSPs results in a weighted hypergraph of edges and 3-edges. (A 3-edge seems needed, in the event that three partitions contain variables from a single constraint.) It is conjectured that matrix multiplication can be done in $O(n^{2+o(1)})$ time, in which case the above algorithm (setting $k(n) = O(1)$) runs in $\tilde{O}(2^{2n/3})$ time. In our investigation of 3-CSPs, it appears that $2^{3n/4}$ might be possible using an extension of our ideas. Let us therefore conjecture hopefully that for all k, COUNT-k-CSP is in $\tilde{O}(2^{n(1-\frac{1}{k+1})})$ time.

• Are there faster algorithms for 2-CSP optimization that only use polynomial space?

• Is there a randomized k-clique detection/counting algorithm running in $O(n^{2+o(1)})$? (Is there merely a good one that doesn't use matrix multiplication?)

• We avoided brute-force search for a class of NP-hard problems by splitting feasible solutions into pieces, listing possible settings to the pieces, associating different pieces, then constructing an optimal solution from the pieces efficiently. What other NP-hard problems are amenable to this general idea?

7 Acknowledgements.

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

References

[1] J. Alber, J. Gramm and R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics* 229:3–27, Elsevier, 2001.

[2] N. Alon and M. Naor. Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. *Algorithmica* 16:434–449, 1996.

[3] N. Alon, R. Yuster, and U. Zwick. Color-Coding. *JACM* 42(4):844–856, 1995.

[4] N. Bansal and V. Raman. Upper bounds for Max-Sat: Further Improved. In *Proceedings of ISAAC99*, Springer LNCS 1741:247–258, 1999.

[5] M. Charikar, P. Indyk, and R. Panigrahy. New Algorithms for Subset Query, Partial Match, Orthogonal Range Searching, and Related Problems. In *Proceedings of ICALP*, 451–462, 2002.

[6] J. Chen and I. Kanj. *Improved Exact Algorithms for MAX-SAT.* In *Proceedings of LATIN*, Springer LNCS 2286:341–355, 2002.

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

[8] E. Cohen and D. D. Lewis. Approximating Matrix Multiplication for Pattern Recognition Tasks. In *Proceedings of ACM-SIAM SODA*, 682–691, 1997.

[9] U. Feige and J. Kilian. On Limited versus Polynomial Nondeterminism. *Chicago Journal of Theoretical Computer Science* 1, 12 March 1997.

[10] J. Gramm and R. Niedermeier. Faster exact solutions for Max2Sat. In *Proceedings of the 4th Conference on Algorithms and Complexity*, Springer LNCS 1767: 174-186, 2000.

[11] J. Gramm, E.A. Hirsch, R. Niedermeier and P. Rossmanith. Worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. *Discrete Applied Mathematics* 130(2):139–155, 2003. Preliminary version appeared in *SAT 2000.*

[12] E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev. MAX-SAT approximation beyond the limits of polynomial-time approximation. *Annals of Pure and Applied Logic* 113(1-3): 81–94, 2001.

[13] E. A. Hirsch, Worst-case study of local search for MAX-k-SAT. *Discrete Applied Mathematics* 130(2): 173-184, 2003. First appeared in *Proceedings of RANDOM 2000.*

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

[15] T. Hofmeister, U. Schöning, R. Schuler, O. Watanabe. A probabilistic 3-SAT algorithm further improved. In *Proceedings of STACS02*, 192-202, 2002.

[16] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *JACM* 21: 277–292, 1974.

[17] R. Impagliazzo, R. Paturi, and F. Zane. Which Problems Have Strongly Exponential Complexity? *JCSS* 63(4): 512–530, 2001.

[18] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4): 413-423, 1978.

[19] T. S. Jayram, S. Khot, R. Kumar, and Y. Rabani. Cell-probe lower bounds for the partial match problem. In *Proceedings of ACM STOC*, 667–672, 2003.

[20] D. Knuth. *The Art of Computer Programming: Sorting and Searching.* 1973.

[21] A. S. Kulikov and S. S. Fedin. A $2^{|E|/4}$-time Algorithm for MAX-CUT. *Zapiski nauchnyh seminarov POMI*, No.293, 2002, pp.129-138.

[22] M. Mahajan and V. Raman. Parameterizing above Guaranteed Values: MAXSAT and MAXCUT. *J. Algorithms* 31(2): 335-354, 1999. Preliminary version in *Electronic Colloquium on Computational Complexity*, 1997.

[23] B. Monien, E. Speckenmeyer, 3-satisfiability is testable in $O(1.62^r)$ steps. Bericht Nr. 3/1979, Reihe Theoretische Informatik, Universität-Gesamthochschule-Paderborn.

[24] B. Monien and E. Speckenmeyer. Upper bounds for covering problems. Bericht Nr. 7/1980, Reihe Theoretische Informatik, Universität-Gesamthochschule-Paderborn.

[25] J. Nesetril and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2): 415–419, 1985.

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

[27] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *JCSS* 43:425–440, 1991.

[28] P. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In *Proceedings of the DIMACS Workshop on Quadratic Assignment Problems*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 16, 1–42, 1994.

[29] R. Paturi, P. Pudlak, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th IEEE FOCS*, 628-637, 1998.

[30] P. Pudlak. Satisfiability - Algorithms and Logic. In *Mathematical Foundations of Computer Science*, Springer Lecture Notes in Computer Science 1450, 129–141, 1998.

[31] V. Raman, B. Ravikumar, and S. Srinivasa Rao. A Simplified NP-Complete MAXSAT problem. *Information Processing Letters* 65:1–6, 1998.

[32] R. Rivest. Partial match retrieval algorithms. SIAM J. on Computing, 5:19–50, 1976.

[33] M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[34] F. Ruskey. Simple combinatorial Gray codes constructed by reversing sublists. In *Proceedings of International Symposium on Algorithms and Computation*, Springer LNCS 762:201–208, 1993.

[35] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th IEEE FOCS*, 410–414, 1999.

[36] R. Schroeppel and A. Shamir. A $T{=}O(2^{n/2})$, $S{=}O(2^{n/4})$ Algorithm for Certain NP-Complete Problems. *SIAM J. Comput.* 10(3): 456–464, 1981.

[37] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. Accepted in *J. Algorithms*, 2003. `ftp://theorie.informatik.uni-ulm.de/pub/papers/ti/cnf4sat.ps`

[38] A. Scott and G. Sorkin. Faster Algorithms for MAX CUT and MAX CSP, with Polynomial Expected Time for Sparse Instances. In *Proceedings of RANDOM 2003*, to appear.

[39] R. Williams. On Computing k-CNF Formula Properties. To appear in Springer-Verlag LNAI. Preliminary version in *SAT 2003.*

[40] G.J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization - Eureka! You shrink!*. M. Juenger, G. Reinelt and G. Rinaldi (eds.). LNCS 2570, Springer, 185–207, 2003.

[41] U. Zwick. All Pairs Shortest Paths using bridging sets and rectangular matrix multiplication. *JACM* 49(3):289–317, May 2002.