# An LP-Designed Algorithm for Constraint Satisfaction

Alexander D. Scott[1] and Gregory B. Sorkin[2]

[1] Mathematical Institute, University of Oxford, Oxford OX1 3LB, UK.
scott@maths.ox.ac.uk
[2] Department of Mathematical Sciences, IBM T.J. Watson Research Center, Yorktown Heights NY 10598, USA. sorkin@watson.ibm.com

**Abstract.** The class Max $(r, 2)$-CSP consists of constraint satisfaction problems with at most two $r$-valued variables per clause. For instances with $n$ variables and $m$ binary clauses, we present an $\widetilde{O}(r^{19m/100})$-time algorithm. It is the fastest algorithm for most problems in the class (including Max Cut and Max 2-Sat), and in combination with "Generalized CSPs" introduced in a companion paper, also allows counting, sampling, and the solution of problems like Max Bisection that escape the usual CSP framework. Linear programming is key to the design as well as the analysis of the algorithm.

## 1 Introduction

A recent line of research has been to speed up exponential-time algorithms (deterministic or randomized) for maximization problems such as Max 2-Sat and Max Cut. For example, Gramm, Hirsch, Niedermeier and Rossmanith solve Max 2-Sat in time $\widetilde{O}(2^{m/5})$ and use this to solve Max Cut in time $\widetilde{O}(2^{m/3})$ [GHNR03], while Kulikov and Fedin solve Max Cut in time $\widetilde{O}(2^{m/4})$ [KF02], where $m$ is the number of constraints or edges. (The $\widetilde{O}(\cdot)$ notation is defined in Section 2.)

The typical method is to repeatedly *transform* an instance to a smaller one or *split* it into several smaller ones (whence the exponential running time) until trivial instances are reached; the reductions are then reversed to recover a solution to the original instance. In [SS03] we introduced a new such method, distinguished by the fact that reducing an instance of Max Cut, for example, results in a problem that is not Max Cut, but where the reductions are closed over the larger class Max 2-CSP. This allowed the reductions to be simpler, fewer, and more powerful. The algorithm ran in time $\widetilde{O}(r^{m/5})$ (time $\widetilde{O}(2^{m/5})$ for binary-valued problems), making it the fastest for Max Cut, but tied for Max 2-Sat.

**Results** The present $\widetilde{O}(r^{19m/100})$ algorithm is the fastest for Max Cut, Max 2-Sat, Max Dicut, weighted versions of these problems, less often considered problems like Max Ones 2-Sat, Max 2-Lin, and of course general Max 2-CSP; more efficient algorithms are known only for a few problems such as Maximum

Independent Set (MIS). (For discussion of MIS and references, see the full version, which also addresses polynomial factors in an efficient implementation.) In combination with a "Generalized CSP" approach described in a companion paper [SS06], the algorithms here also enable (still in time $\widetilde{O}(r^{19m/100})$) counting CSP solutions of each possible cost; randomly sampling from optimal solutions, or according to the Gibbs measure or other distributions; and solving problems that do not fall into the Max 2-CSP framework, like Max Bisection, Sparsest Cut, judicious partitioning, Max Clique (without blowing up the input size), and multi-objective problems.

**Techniques** We focus throughout on the graph supporting a CSP instance. The key step in our earlier $\widetilde{O}(r^{m/5})$ analysis was to use a linear program (LP) to show that the number of splitting reductions for an $m$-edge graph is $\leq m/5$. Consideration of an example which achieves that bound shows that any improvement must exploit *connected components* of the CSP's underlying graph. Conceptually, treatment of separate components sits uneasily with the LP analysis, which considers the (indivisible) degree sequence of the full graph: the usual argument that in case of component division "we are done, by induction" cannot be applied. However, a simple observation will sweep away the difficulty.

The LP was essential in the *design* of the new algorithm as well as its analysis. Its *primal* solution shows which reductions contribute to the worst case. We can easily exclude a bad reduction from the LP to see if an improved bound would result, and only then think hard about whether the reduction can be avoided.

The LP method presented is certainly applicable to reductions other than our own, and we hope to see it applied to algorithm design and analysis in contexts other than exponential-time algorithms and CSPs. (For a different use of LPs in automating an extremal construction, see [TSSW00].)

**Literature survey** The $(a, b)$-CSP model is extensively exploited for example in Beigel and Eppstein's [BE05]. An early version of our results was given in technical report [SS04]. We have already mentioned the Max 2-Sat algorithm of [GHNR03] and the Max Cut algorithm of [KF02]; [SS03] improved on the latter, and the present result improves on both. The lovely algorithm of Williams [Wil04], like ours, applies to all of 2-CSP. It runs in time $\widetilde{O}(n^{\omega/3})$, where $\omega < 2.376$ is the matrix-multiplication exponent. Depending on $n$ rather than $m$, this algorithm is faster than ours if the average degree is above $200/(19\omega) < 4.430$. However, it requires exponential *space* of order $2^{2n/3}$.

**Outline** In the next section we define the class Max 2-CSP, and in Section 3 we introduce the reductions our algorithms will use. In Section 4 we define and analyze the $\widetilde{O}(r^{m/5})$ algorithm as a relatively gentle introduction to the tools, including the LP analysis. The $\widetilde{O}(r^{19m/100})$ algorithm is presented in Section 5.

## 2 Max $(r, 2)$-CSP

The problem Max Cut is to partition the vertices of a given graph into two classes so as to maximize the number of edges "cut" by the partition. Think of

each *edge* as being a *function* on the classes (or "colors") of its endpoints, with value 1 if the endpoints are of different colors, 0 if they are the same: Max Cut is equivalent to finding a 2-coloring of the vertices which maximizes the sum of these edge functions. This view naturally suggests a generalization.

An *instance* $(G, S)$ of Max $(r, 2)$-CSP is given by an "underlying" graph $G = (V, E)$ and a set $S$ of "score" functions. Writing $[r] = \{1, \ldots, r\}$ for the set of available colors, we have a "dyadic" score function $s_e : [r]^2 \to \mathbb{R}$ for each edge $e \in E$, a "monadic" score function $s_v : [r] \to \mathbb{R}$ for each vertex $v \in V$, and finally a single "niladic" score "function" $s_\emptyset : [r]^0 \to \mathbb{R}$ which takes no arguments and is just a constant convenient for bookkeeping.

A *candidate solution* is a function $\phi : V \to [r]$ assigning "colors" to the vertices (we call $\phi$ an "assignment" or "coloring"), and its *score* is

$$s(\phi) := s_\emptyset + \sum_{v \in V} s_v(\phi(v)) + \sum_{uv \in E} s_{uv}(\phi(u), \phi(v)). \tag{1}$$

An *optimal solution* $\phi$ is one which maximizes $s(\phi)$.

**Notation** We reserve the symbols $G$ for the underlying graph of a Max $(r, 2)$-CSP instance, $n$ and $m$ for its numbers of vertices and edges, $[r] = \{1, \ldots, r\}$ for the allowed colors of each vertex, and $L = 1 + nr + mr^2$ for the input length. Since a CSP instance with $r < 2$ is trivial, we will assume $r \geq 2$ as part of the definition. For brevity, we write "$d$-vertex" for "vertex of degree $d$". The notation $\widetilde{O}(\cdot)$ suppresses polynomial factors in any parameters, so for example $\widetilde{O}(r^{cn})$ may mean $O(r^3 n\, r^{cn})$.

**Remarks** The class Max $(r, 2)$-CSP is surprisingly flexible, and in addition to Max Cut and Max 2-Sat includes problems like MIS and minimum vertex cover that are not at first inspection structured around pairwise constraints. Readers familiar with the class $\mathcal{F}$-Sat will see that when the arity of $\mathcal{F}$ is limited to 2, Max $(r, 2)$-CSP also contains $\mathcal{F}$-Sat, $\mathcal{F}$-Max-Sat and $\mathcal{F}$-Min-Sat (e.g., Max 2-Sat and Max 2-Lin) and $\mathcal{F}$-Max-Ones.

## 3 Reductions

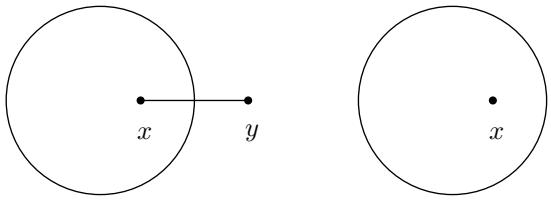
As with most of the works surveyed above, our algorithms are based on progressively reducing the instance to one with fewer vertices and edges until the instance becomes trivial. Because we work in the general class Max $(r, 2)$-CSP rather than trying to stay within a smaller class such as Max 2-Sat, our reductions are simpler and fewer than is typical. For example, [GHNR03] uses seven reduction rules; we have just three (plus a trivial "0-reduction" that other works may treat implicitly). The first two reductions each produce equivalent instances with one vertex fewer, while the third produces a set of $r$ instances, each with one vertex fewer, one of which is equivalent to the original instance. We expand the previous notation $(G, S)$ for an instance to $(V, E, S)$, where $G = (V, E)$.

**Reduction 0 (transformation)** This is a trivial "pseudo-reduction". If a vertex $y$ has degree 0 (so it has no dyadic constraints), then set $s_\emptyset = s_\emptyset + \max_{C \in [r]} s_y(C)$ and delete $y$ from the instance entirely.

**Reduction I** Let $y$ be a vertex of degree 1, with neighbor $x$. Reducing $(V, E, S)$ on $y$ results in a new problem $(V', E', S')$ with $V' = V \setminus y$ and $E' = E \setminus xy$. $S'$ is the restriction of $S$ to $V'$ and $E'$, except that for all colors $C \in [r]$ (and in total time $O(r^2)$) we set

$$s'_x(C) = s_x(C) + \max_{D \in [r]} \{s_{xy}(CD) + s_y(D)\}.$$

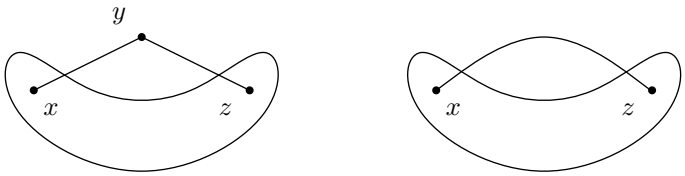
Note that any coloring $\phi'$ of $V'$ can be extended to a coloring $\phi$ of $V$ in $r$ ways, depending on the color assigned to $y$. Writing $(\phi', D)$ for the extension in which $\phi(y) = D$, the defining property of the reduction is that $S'(\phi') = \max_D S(\phi', D)$. In particular, $\max_{\phi'} S'(\phi') = \max_\phi S(\phi)$, and an optimal coloring $\phi'$ for the instance $(V', E', S')$ can be extended to an optimal coloring $\phi$ for $(V, E, S)$.




**Reduction II (transformation)** Let $y$ be a vertex of degree 2, with neighbors $x$ and $z$. Reducing $(V, E, S)$ on $y$ results in a new problem $(V', E', S')$ with $V' = V \setminus y$ and $E' = (E \setminus \{xy, yz\}) \cup \{xz\}$. $S'$ is the restriction of $S$ to $V'$ and $E'$, except that for $C, D \in [r]$ (and in total time $O(r^3)$) we set

$$s'_{xz}(CD) = s_{xz}(CD) + \max_{F \in [r]} \{s_{xy}(CF) + s_{yz}(FD) + s_y(F)\} \qquad (2)$$

if there was already an edge $xz$, discarding the first term $s_{xz}(CD)$ if there was not. As in Reduction I, any coloring $\phi'$ of $V'$ can be extended to $V$ in $r$ ways, according to the color $F$ assigned to $y$, and the defining property of the reduction is that $S'(\phi') = \max_F S(\phi', F)$. In particular, $\max_{\phi'} S'(\phi') = \max_\phi S(\phi)$, and an optimal coloring $\phi'$ for $(V', E', S')$ can be extended to an optimal coloring $\phi$ for $(V, E, S)$.




**Reduction III (splitting)** Let $y$ be a vertex of degree 3 *or higher*. Where reductions I and II each had a single reduction of $(V, E, S)$ to $(V', E', S')$, here we define $r$ different reductions: for each color $C$ there is a reduction

of $(V, E, S)$ to $(V', E', S^C)$ corresponding to assigning the color $C$ to $y$. We define $V' = V \setminus y$, and $E'$ as the restriction of $E$ to $V \setminus y$. $S^C$ is the restriction of $S$ to $V \setminus y$, except that we set

$$(s^C)_0 = s_\emptyset + s_y(C),$$

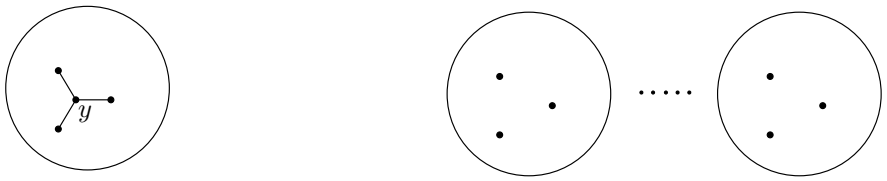
and, for every neighbor $x$ of $y$ and every $D \in [r]$,

$$(s^C)_x(D) = s_x(D) + s_{xy}(DC).$$

As in the previous reductions, any coloring $\phi'$ of $V \setminus y$ can be extended to $V$ in $r$ ways, $(\phi', C)$ where color $C$ is given to $y$, and now (this is different!) $S^C(\phi') = S(\phi', C)$. Furthermore,

$$\max_C \max_{\phi'} S^C(\phi') = \max_\phi S(\phi),$$

and an optimal coloring on the left *is* an optimal coloring on the right.




$(V, E, S)$ $(V', E', S^1)$ $(V', E', S^r)$

## 4 An $\widetilde{O}(r^{m/5})$ Algorithm

As a warm-up to our $\widetilde{O}(r^{19m/100})$ algorithm, in this section we will present Algorithm A, which will run in time $O(nr^{3+m/5})$ and space $O(L)$. (Recall that $L = 1 + nr + mr^2$ is the input length.) Roughly speaking, a simple recursive algorithm for solving an input instance could work as follows. Begin with the input problem instance.

Given an instance $\mathcal{M} = (G, S)$:

1. If any reduction of type 0, I or II is possible (in that order of preference), apply it to reduce $\mathcal{M}$ to $\mathcal{M}'$, recording certain information about the reduction. Solve $\mathcal{M}'$ recursively, and use the recorded information to reverse the reduction and extend the solution to one for $\mathcal{M}$.
2. If only a type III reduction is possible, reduce (in order of preference) on a vertex of degree 5 or more, 4, or 3. For $i \in [r]$, recursively solve each of the instances $\mathcal{M}^i$ in turn, select the solution with the largest score, and use the recorded information to reverse the reduction and extend the solution to one for $\mathcal{M}$.

3. If no reduction is possible then the graph has no vertices, there is a unique coloring (the empty coloring), and the score is $s_\emptyset$ (from the niladic score function).

If the recursion depth is $\ell$, the recursive algorithm's running time is $\widetilde{O}(r^\ell)$, and the preference order over type III reductions is needed to obtain the bound $\ell \leq m/5$ of Lemma 1; we prove this bound in Section 4. Numerous complications in optimizing the polynomial factors are addressed in the full paper.

**Phases** While a III-reduction produces $r$ different subinstances, all have the same underlying graph: the original graph with one vertex deleted. Type 0, I and II CSP reductions also change the underlying graph in a way independent of the score functions, so all the CSP reductions have graph-reduction counterparts depending only on the underlying graph and the reduction vertex. Our algorithm runs in three phases. The first, Algorithm A.1, finds a sequence of $n$ *graph* reductions; this can be done in linear time and space, and our focus is to show that it has $\ell \leq m/5$ III-reductions. Details of this phase will allow us to assume the graph is always simple. The second phase finds an optimal cost, and the third produces a corresponding coloring.

**Recursion depth** The crux of the analysis is the following lemma.

**Lemma 1.** *Algorithm A.1 reduces a graph $G$ with $n$ vertices and $m$ edges to an empty graph after $\ell \leq m/5$ III-reductions.*

*Proof.* While the graph has maximum degree 5 or more, Algorithm A.1 III-reduces only on such a vertex, destroying at least 5 edges; any I- or II-reductions included in the same step only increase the number of edges destroyed. Thus, it suffices to prove the lemma for graphs with maximum degree 4 or less. Since the reductions never increase the degree of any vertex, the maximum degree will always remain at most 4.

In this paragraph, we give some intuition for the rest of the argument. Algorithm A.1 III-reduces on 4-vertices as long as possible, before III-reducing on 3-vertices, whose neighbors must then all be of degree 3 (degrees 0, 1 or 2 would trigger a 0-, I- or II-reduction in preference to the III-reduction). Note that each III-reduction on a 3-vertex destroys 6 edges if we imagine immediately following up with II-reductions on its neighbors; similarly, reduction on a 4-vertex destroys at least 5 edges unless the 4-vertex has no degree-3 neighbor. The only problem comes from reductions on 4-vertices whose neighbors are also all of degree 4, as these destroy only 4 edges. Our LP analysis will show that because such reductions create degree-3 vertices, and the algorithm terminates with none, these bad reductions cannot occur too often.

We proceed by considering the various types of reductions and their effect on the number of edges and the number of 3-vertices. The reductions are catalogued in Table 1. The first row, for example, shows that III-reducing on a 4-vertex with 4 neighbors of degree 4 (and thus none of degree 3), destroys 4 edges, and (changing the neighbors from degree 4 to 3) destroys 5 4-vertices (including itself) and creates 4 3-vertices. The remaining rows up to the table's separating

| deg | #nbrs of deg | | | | destroys | | | | | steps |
|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 3 | 2 | 1 | $e$ | 4 | 3 | 2 | 1 | |
| 4 | 4 | 0 | 0 | 0 | 4 | 5 | −4 | 0 | 0 | 1 |
| 4 | 3 | 1 | 0 | 0 | 4 | 4 | −2 | −1 | 0 | 1 |
| 4 | 2 | 2 | 0 | 0 | 4 | 3 | 0 | −2 | 0 | 1 |
| 4 | 1 | 3 | 0 | 0 | 4 | 2 | 2 | −3 | 0 | 1 |
| 4 | 0 | 4 | 0 | 0 | 4 | 1 | 4 | −4 | 0 | 1 |
| 3 | 0 | 3 | 0 | 0 | 3 | 0 | 4 | −3 | 0 | 1 |
| 2 | | | | | 1 | 0 | 0 | 1 | 0 | 0 |
| $\frac{1}{2}\,e$ | 1 | 0 | 0 | 0 | $\frac{1}{2}$ | 1 | −1 | 0 | 0 | 0 |
| $\frac{1}{2}\,e$ | 0 | 1 | 0 | 0 | $\frac{1}{2}$ | 0 | 1 | −1 | 0 | 0 |
| $\frac{1}{2}\,e$ | 0 | 0 | 1 | 0 | $\frac{1}{2}$ | 0 | 0 | 1 | −1 | 0 |
| $\frac{1}{2}\,e$ | 0 | 0 | 0 | 1 | $\frac{1}{2}$ | 0 | 0 | 0 | 1 | 0 |

**Table 1.** Tabulation of the effects of various reductions in Algorithm A.1.

line similarly illustrate the other III-reductions. Below the line, II-reductions and I-reductions are decomposed into parts. As shown just below the line, a II-reduction, regardless of the degrees of the neighbors, first destroys 1 edge and 1 2-vertex, and counts as 0 steps (steps count only III-reductions). In the process, the II-reduction may create a parallel edge, which may at some stage be deleted by Algorithm A.1. Since the exact effect of an edge deletion depends on the degrees of its neighbors, to minimize the number of cases we treat an edge deletion as two half-edge deletions, each destroying $\frac{1}{2}$ an edge, and whose effect depends on the degree of the half-edge's incident vertex. For example the table's next line shows deletion of a half-edge incident to a 4-vertex, changing it to a 3-vertex and destroying half an edge. The last four rows of the table also capture I-reductions. 0-reductions are irrelevant to the table, which does not consider vertices of degree 0.

The sequence of reductions reducing $G$ to an empty graph can be parametrized by an 11-vector $\boldsymbol{n}$ giving the number of reductions (and partial reductions) indexed by the rows of the table, so for example its first element is the number of III-reductions on 4-vertices whose neighbors are also all 4-vertices. Since the reductions destroy all $m$ edges, the dot product of $\boldsymbol{n}$ with the table's column "destroys e" (call it $\boldsymbol{e}$) must be precisely $m$. Since all 4-vertices are destroyed, the dot product of $\boldsymbol{n}$ with the column "destroys 4" (call it $\boldsymbol{d_4}$) must be $\geq 0$, and the same goes for the "destroy" columns 3, 2 and 1. The number of III-reductions is the dot product of $\boldsymbol{n}$ with the "steps" column, $\boldsymbol{n} \cdot \boldsymbol{s}$. How large can the number of III-reductions $\boldsymbol{n} \cdot \boldsymbol{s}$ possibly be?

To find out, let us maximize $\boldsymbol{n} \cdot \boldsymbol{s}$ subject to the constraints that $\boldsymbol{n} \cdot \boldsymbol{e} = m$ and that $\boldsymbol{n} \cdot \boldsymbol{d_4}$, $\boldsymbol{n} \cdot \boldsymbol{d_3}$, $\boldsymbol{n} \cdot \boldsymbol{d_2}$ and $\boldsymbol{n} \cdot \boldsymbol{d_1}$ are all $\geq 0$. Instead of maximizing over proper reduction collections $\boldsymbol{n}$, which seem hard to characterize, we maximize over the larger class of non-negative real vectors $\boldsymbol{n}$, giving an upper bound on the proper maximum. Maximizing the linear function $\boldsymbol{n} \cdot \boldsymbol{s}$ of $\boldsymbol{n}$ subject to a set of linear constraints (such as $\boldsymbol{n} \cdot \boldsymbol{e} = m$ and $\boldsymbol{n} \cdot \boldsymbol{d_4} \geq 0$) is simply solving a

linear program (LP); the LP's constraint matrix and objective function are the part of Table 1 right of the double line. To avoid dealing with "$m$" in the LP, we set $\boldsymbol{n'} = \boldsymbol{n}/m$, and solve the LP with constraints $\boldsymbol{n'} \cdot \boldsymbol{e} = 1$, and as before $\boldsymbol{n'} \cdot \boldsymbol{d_4} \geq 0$, etc., to maximize $\boldsymbol{n'} \cdot \boldsymbol{s}$. The "$\boldsymbol{n'}$" LP is a small linear program (11 variables and 5 constraints) and its maximum is precisely 1/5, showing that the number of III-reduction steps — $\boldsymbol{n} \cdot \boldsymbol{s} = m\boldsymbol{n'} \cdot \boldsymbol{s}$ — is at most $m/5$.

This establishes that the number of type-III reductions can be at most 1/5th the number of edges $m$, concluding the proof.

**Theorem 2.** *A Max $(r, 2)$-CSP instance on $n$ variables with $m$ dyadic constraints and length $L$ can be solved in time $O(nr^{3+m/5})$ and space $O(L)$.*

*Proof.* The theorem is an immediate consequence of Lemma 1 and the polynomial-factor considerations ignored in this version of the paper.

The LP's *dual* solution gives a "potential function" proof of Lemma 1. The dual assigns "potentials" to the graph's edges and to vertices according to their degrees, such that the number of steps counted for a reduction is at most its change to the potential. Since the potential is initially at most $0.20m$ and finally 0, the number of steps is at most $m/5$. The *primal* solution of the LP uses (proportionally) 1 III-reduction on a 4-vertex with all 4-neighbors, 1 III-reduction on a 3-vertex, and 3 II-reductions; reducing a $K_5$ realizes these values.

## 5 An $\widetilde{O}(r^{19m/100})$ algorithm

The analysis of Algorithm A contains the seeds of its improvement. First, since reductions on vertices 5-vertices may destroy only 5 edges, we cannot ignore them and improve on $m/5$. This simply means including them in the LP.

Second, were this the only change we made, we would find that the LP solution is the same as before, with support on a reduction on a 4-vertex with all 4-neighbors (a "bad" reduction destroying only 4 edges), and harmless reductions (III-reduction on a 3-vertex and the I- and II-reductions it enables). This suggests that we should focus on eliminating the bad reduction. Indeed, if we exclude it from the LP, the LP cost decreases to 23/120 (about 0.192), and the new solution shows support on a reduction on a degree-5 vertex with all degree-5 neighbors and a degree-4 vertex with one degree-3 neighbor (each resulting in the destruction of 5 edges). If the first of these cases could also be eliminated, the LP would have cost 19/100, precisely what our algorithm will achieve. Improving beyond this would require addressing the remaining bad cases of a 5-vertex with neighbors of degree 5 except for one of degree 4, and a 4-vertex with neighbors of degree 4 except for one of degree 3.

Finally, a collection of many disjoint $K_5$s requires $m/5$ III-reductions in total. To beat $\widetilde{O}(r^{m/5})$ we will have to use the fact that an optimum solution to a disconnected CSP is a union of solutions of its components, and thus the $m/5$ reductions can in some sense be done in parallel, rather than sequentially. Correspondingly, where Algorithm A.1 built a *sequence* of reductions of length at most $m/5$, Algorithm B.1 will build a reduction *tree* whose *depth* is at most

$2 + 19m/100$. The depth bound is proved by showing that in any sequence of reductions in a component on a fixed vertex, all but at most two "bad" reductions can be paired with other reductions, and for the good reductions (including the paired ones), the LP has maximum 19/100.

**Algorithm B: First phase** As with Algorithm A, a first phase Algorithm B.1 of Algorithm B performs only graph reductions. Like Algorithm A, Algorithm B preferentially performs type 0, I or II reductions, but it is more particular about the vertices on which it III-reduces. When forced to perform a type III reduction, Algorithm B selects a vertex in the following decreasing order of preference:

- a vertex of degree $\geq 6$;
- a vertex of degree 5 with at least 1 neighbor of degree 3 or 4;
- a vertex of degree 5 whose neighbors all have degree 5;
- a vertex of degree 4 with at least 1 neighbor of degree 3;
- a vertex of degree 4 whose neighbors all have degree 4;
- a vertex of degree 3.

When Algorithm B makes any such reduction with any degree-3 neighbor, it immediately follows up with II-reductions on all those neighbors.

Because Algorithm B treats graph components individually, the sequence of reductions must be organized into a *reduction tree*. The defining property of the reduction tree is that if reduction on a vertex $v$ divides the graph into $k$ components, then a corresponding tree node $v$ has $k$ children, one for each component, the child node corresponding to the first vertex reduced upon in that component (the first vertex in the reduction sequence restricted to the set of vertices in the component). If the graph is initially disconnected, the reduction "tree" is really a forest, but since this case presents no additional issues we will speak in terms of a tree. We remark that the number of children $k$ is necessarily 1 for I- and II-reductions, can be 1 or more for a III-reduction, and is 0 for a 0-reduction.

Call the maximum number of III-reduction nodes in any root-to-leaf path in the reduction tree its "III-reduction" depth. Lemma 3 characterizes an efficient construction of the tree, but it is clear that it can be done in polynomial time and space. The crux of the matter is Lemma 4, which relies on the reduction preference order set forth above, but not on the algorithmic details of Algorithm B.1.

**Lemma 3.** *A reduction tree on $n$ vertices which has III-reduction depth $d$ can be constructed in time $O(dn + n)$ and space $O(m + n)$.*

**Splitting-tree depth** Analogous to Lemma 1 characterizing Algorithm A, the next lemma is the heart of the analysis of Algorithm B.

**Lemma 4.** *For a graph $G$ with $m$ edges, the reduction tree's III-reduction depth is $d \leq 2 + 19m/100$.*

*Proof.* It suffices to prove the lemma for graphs with maximum degree $\leq 5$. As in the proof of Lemma 1, for each type of reduction we will count the number of edges and vertices of various degrees it destroys, and its depth: the depth is normally 1 for a III-reduction and 0 otherwise, but we will now introduce "paired" pseudo-reductions counting for depth 2. Recall that in Algorithm B we immediately follow each III-reduction with a II-reduction on each 2-vertex it produces.

Define a "bad" reduction to be one on a 5-vertex all of whose neighbors are also of degree 5, or on a 4-vertex all of whose neighbors are of degree 4. (These two reductions destroy 5 and 4 edges respectively, while, except for reducing on a 4-vertex with three 4-neighbors and one 3-neighbor, every other reduction, coupled with the II-reductions it enables, destroys at least 6 edges.) The analysis is aimed at controlling the number of these reductions.

For shorthand, we write reductions in terms of the degree of the vertex on which we are reducing followed by the numbers of neighbors of degrees 5, 4, and 3, so for example the "bad" reduction on a 5-vertex is written (5|500).

Within a component, a (5|500) reduction is performed only if there is no 5-vertex adjacent to a 3- or 4-vertex; this means the component *has* no 3- or 4-vertices, since otherwise a path from such a vertex to the 5-vertex would include an edge incident on a 5-vertex and a 3- or 4-vertex. We track the component containing one vertex, say vertex 1, as it is reduced. If the component necessitates a bad 5-reduction, one of four things must be true:

1. *This is the first degree-5 reduction in this branch of the splitting tree.* This case can occur only once. Weakening this constraint, we will allow it to occur any number of times, but we will count its depth contribution as 0, and add 1 to the depth at the end. For this reason, the first bold row in Table 2 has depth 0 not 1.
2. *The previous III-reduction (which because of our preference order must also have been a degree-5 reduction) was on a* (5|005) *vertex, and left no vertices of degree 3 or 4.* In this case we pair the bad (5|500) reduction with its preceding (5|005) reduction. This defines a new "pair" pseudo-reduction shown as the second bold row of the table: it counts for 2 steps, destroys 15 edges, etc. (Other, non-paired (5|005) reductions are still allowed as before.)
3. *The previous III-reduction was on a 5-vertex and produced vertices of degree 3 or 4 in this component, but they were destroyed by I- and II-reductions.* In this case we similarly pair the (5|500) reduction with a I- or II-reduction, but we cannot say specifically with which sort. The "forces" column of Table 2 will constrain each (5|500) reduction for this case to be accompanied by at least one I- or II-reduction (or two "half-edge" reductions) of any sort.
4. *The previous III-reduction was on a 5-vertex and produced vertices of degree 3 or 4, but split them all off into other components.* In this case, the (5|500) reduction produces a non-empty side component destroyed with the usual reductions but adding depth 0 for the component of interest. These reductions can be expressed as a nonnegative combination of half-edge reductions, so we can pair the (5|500) reduction with any two of these, much as in case (3).

| deg | #nbrs of deg | | | | | destroys | | | | | forces | depth |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 5 | 4 | 3 | 2 | 1 | $e$ | 4 | 3 | 2 | 1 | | |
| 5 | 0 | 0 | 5 | 0 | 0 | 10 | 0 | 5 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 4 | 0 | 0 | 9 | 1 | 3 | 0 | 0 | 0 | 1 |
| . | . | . | . | . | . | . | . | . | . | . | . | . |
| 5 | 4 | 1 | 0 | 0 | 0 | 5 | −3 | −1 | 0 | 0 | 0 | 1 |
| **5** | **5** | **0** | **0** | **0** | **0** | **5** | **−5** | **0** | **0** | **0** | **0** | **0** |
| **5 + 5** | **5** | **0** | **5** | **0** | **0** | **15** | **−5** | **5** | **0** | **0** | **0** | **2** |
| **5** | **5** | **0** | **0** | **0** | **0** | **5** | **−5** | **0** | **0** | **0** | **−1** | **1** |
| 4 | 0 | 0 | 4 | 0 | 0 | 8 | 1 | 4 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 3 | 0 | 0 | 7 | 2 | 2 | 0 | 0 | 0 | 1 |
| 4 | 0 | 2 | 2 | 0 | 0 | 6 | 3 | 0 | 0 | 0 | 0 | 1 |
| 4 | 0 | 3 | 1 | 0 | 0 | 5 | 4 | −2 | 0 | 0 | 0 | 1 |
| **4** | **0** | **4** | **0** | **0** | **0** | **4** | **5** | **−4** | **0** | **0** | **0** | **0** |
| **4 + 4** | **0** | **4** | **4** | **0** | **0** | **12** | **6** | **0** | **0** | **0** | **0** | **2** |
| **4** | **0** | **4** | **0** | **0** | **0** | **4** | **5** | **−4** | **0** | **0** | **−1** | **1** |
| 3 | 0 | 0 | 3 | 0 | 0 | 6 | 0 | 4 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| $\frac{1}{2}\,e$ | 1 | 0 | 0 | 0 | 0 | $\frac{1}{2}$ | −1 | 0 | 0 | 0 | $\frac{1}{2}$ | 0 |
| $\frac{1}{2}\,e$ | 0 | 1 | 0 | 0 | 0 | $\frac{1}{2}$ | 1 | −1 | 0 | 0 | $\frac{1}{2}$ | 0 |
| $\frac{1}{2}\,e$ | 0 | 0 | 1 | 0 | 0 | $\frac{1}{2}$ | 0 | 1 | −1 | 0 | $\frac{1}{2}$ | 0 |
| $\frac{1}{2}\,e$ | 0 | 0 | 0 | 1 | 0 | $\frac{1}{2}$ | 0 | 0 | 1 | −1 | $\frac{1}{2}$ | 0 |
| $\frac{1}{2}\,e$ | 0 | 0 | 0 | 0 | 1 | $\frac{1}{2}$ | 0 | 0 | 0 | 1 | $\frac{1}{2}$ | 0 |

**Table 2.** Tabulation of the effects of various reductions in Algorithm B.

Table 2 summarizes the reductions. Together, the four cases above let us exclude (5|500) reductions, replacing them with less harmful possibilities represented by the first three bold rows in the table. Reasoning identically for bad reductions on 4-vertices contributes the other three bold rows. In analyzing a branch of the splitting tree, let vector $\boldsymbol{n}'$ count the (normalized) number of reductions of each type, as in the proof of Lemma 1. Constraining the "forces" column's dot product with $\boldsymbol{n}'$ forces the pairing of a I- or II-reduction with each bad reduction. Everything else goes as before. The LP maximum is 19/100, which (accounting for case (1) occurrences for a 4- and a 5-vertex) proves the reduction depth to be $\leq 2 + 19m/100$.

**Corollary 5.** *Algorithm B solves a Max $(r, 2)$-CSP instance $(G, S)$, where $G$ has $n$ vertices and $m$ edges, in time $O(nr^{5+19m/100})$ and in linear space.*

Motivated by a tree-decomposition CSP approach in a recent report by Kneis and Rossmanith [KR05], we note the following corollary of Lemma 4.

**Corollary 6.** *A graph $G$ with $m$ edges has treewidth at most $3 + 19m/100$.*

## 6 Conclusions

The LP is key to our algorithm design as well as the analysis. We begin with a collection of reductions, and a preference order on them, guided by intuition. The

preference order both excludes some cases (e.g., reducing on high-degree vertices first, we do not need to worry about a reduction vertex having a neighbor of larger degree) and determines an LP. Solving the LP pinpoints the "bad" reductions that determine the bound. We then try to ameliorate these cases: in the present paper we showed that each could be paired with another reduction to give a less bad combined reduction, but we might also have taken some other course such as changing the preference order to eliminate bad reductions. Using the LP as a black box is a convenient way to engage in this cycle of algorithm analysis and improvement, an approach that should be applicable to other problems.

Our methods seem not to extend to 3-variable CSPs, since a II-reduction would combine two 3-variable clauses into a 4-variable clause.

The improvement from $m/5$ to $19m/100$ is significant in that $m/6$ appears to be a natural barrier: In a random cubic graph, a III-reduction results in the deletion of 6 edges and a new cubic graph, and to beat $m/6$ requires either distinguishing the new graph from random cubic, or targeting many III-reductions so as to divide the graph into components. Such an approach would require new ideas outside the scope of the local properties we consider.

## References


[BE05] Richard Beigel and David Eppstein, *3-coloring in time $O(1.3289^n)$*, J. Algorithms **54** (2005), no. 2, 168–204.

[GHNR03] Jens Gramm, Edward A. Hirsch, Rolf Niedermeier, and Peter Rossmanith, *Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT*, Discrete Appl. Math. **130** (2003), no. 2, 139–155.

[KF02] A. S. Kulikov and S. S. Fedin, *Solution of the maximum cut problem in time $2^{|E|/4}$*, Zap. Nauchn. Sem. S.-Peterburg. Otdel. Mat. Inst. Steklov. (POMI) **293** (2002), no. Teor. Slozhn. Vychisl. 7, 129–138, 183.

[KR05] Joachim Kneis and Peter Rossmanith, *A new satisfiability algorithm with applications to Max-Cut*, Tech. Report AIB-2005-08, Department of Computer Science, RWTH Aachen, 2005.

[SS03] Alexander D. Scott and Gregory B. Sorkin, *Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances*, Proc. 7th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2003, Lecture Notes in Comput. Sci., vol. 2764, Springer, August 2003, pp. 382–395.

[SS04] ———, *A faster exponential-time algorithm for Max 2-Sat, Max Cut, and Max k-Cut*, Tech. Report RC23456 (W0412-001), IBM Research Report, December 2004, See http://domino.research.ibm.com/library/cyberdig.nsf.

[SS06] ———, *Generalized constraint satisfaction problems*, Tech. Report RC23935, IBM Research Report, April 2006, See http://domino.research.ibm.com/library/cyberdig.nsf.

[TSSW00] Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson, *Gadgets, approximation, and linear programming*, SIAM J. Comput. **29** (2000), no. 6, 2074–2097.

[Wil04] Ryan Williams, *A new algorithm for optimal constraint satisfaction and its implications*, Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP), 2004.