Approximating Satisfiable Satisfiability Problems*

Luca Trevisan†

February 22, 1999

Abstract

We study the approximability of the Maximum Satisfiability Problem (Max SAT) and of the boolean k-ary Constraint Satisfaction Problem (Max kCSP) restricted to satisfiable instances. For both problems we improve on the performance ratios of known algorithms for the unrestricted case.

Our approximation for satisfiable Max 3CSP instances is better than any possible approximation for the unrestricted version of the problem (unless $\mathsf{P} = \mathsf{NP}$). This result implies that the requirement of perfect completeness weakens the acceptance power of non-adaptive PCP verifiers that read 3 bits.

We also present the first non-trivial results about PCP classes defined in terms of free bits that collapse to P.

1 Introduction

In the Max SAT problem we are given a boolean formula in conjunctive normal form (CNF) and we are asked to find an assignment of values to the variables that satisfies the maximum number of clauses. More generally, we can assume that each clause has a non-negative weight and that we want to maximize the total weight of satisfied clauses.

Max SAT is a standard NP-hard problem and a considerable research effort has been devoted in the last two decades to the development of approximation algorithms for it. An r-approximate algorithm for Max SAT (where $0 \leq r \leq 1$) is a polynomial-time algorithm that given a formula finds an assignment that satisfies clauses of total weight at least r times the optimum.

Max SAT is also the prototypical element of a large family of optimization problems in which we are given a set of weighted *constraints* over (not necessarily boolean) variables, and we want to find an assignment of values to such variables that maximizes the total weight of satisfied constraints. Problems of this kind, called constraint satisfaction problems, are of central interest in Artificial Intelligence. Their approximability properties are of interest in Theory of Computing since they can express the class $\mathsf{MAX\ SNP}$ [PY91, KMSV99] and the computation of PCP verifiers [ALM+98, Tre96]; complete classifications of their approximability properties, for the case of boolean variables, appear in [Cre95, KSW97]. We call Max kCSP the boolean constraint satisfaction problem where every constraint involves at most k variables.

In this paper we consider the following restriction of the problem of r-approximating Max SAT and Max kCSP: given a satisfiable instance of Max SAT (resp. Max kCSP), find in

*An extended abstract of this paper was presented at the 5th European Symposium on Algorithms (ESA'97), Graz, 1997. This work was done while the author was at University of Geneva.

†`luca@cs.columbia.edu`. Department of Computer Science, Columbia University.

Max SAT	Max 3SAT	Due to
.75	.75	[Yan94]
.75	.75	[GW94]
.758	.765*	[GW95] (using [GW94])
.762*	.77*	[FG95] (using [GW94, GW95])
.765	.769	[OHA96] (using [Yan94, GW94, GW95])
	.801	[TSSW96] (using [FG95])
.77		[Asa97] (using [Yan94, GW94, GW95, FG95, OHA96])
.8	**.826**	**This paper** for satisfiable instances

Table 1: Evolution of the approximation factors for Max SAT and Max 3SAT. The factors depicted with a $*$ do not appear explicitly in the referenced papers [GW95, FG95].

polynomial time an assignment that satisfies at least a fraction r of the total weight of clauses (resp. constraints). The issue of approximating constraint satisfaction problems restricted to satisfiable instances has been considered by Petrank [Pet94], and called *approximation problem at gap location one*. Petrank observed that Max SAT remains MAX SNP-complete when restricted to satisfiable instances, and proved that the same is true for other problems, such as Max 3-Colorable Subgraph and Max 3-Dimensional Matching. More recently, Khanna, Sudan and Williamson [KSW97] proved that for any MAX SNP-complete constraint satisfaction problem for which deciding satisfiability is NP-hard, the restriction to satisfiable instances remains MAX SNP-complete.

In partial contrast with the results of Petrank and of Khanna et al. we prove that restricting Max SAT and Max kCSP to satisfiable instances makes the problems somewhat easier, since we can exploit satisfiability to develop new algorithms with improved approximation guarantees. Our algorithms can also be used to show that certain classes defined in terms of probabilistically checkable proofs with bounded *query bit* or *free bit* complexity are contained in P.

We now discuss our results in more detail.

The Maximum Satisfiability Problem. The Max SAT problem appears in a paper of Johnson [Joh74] which is the first paper where the term "approximation algorithm" was introduced. Johnson proved that his algorithm was 1/2-approximate. It has been recently showed that Johnson's algorithm is indeed 2/3-approximate [CFZ97]. In the last five years, several improved approximation algorithms for Max SAT and its restricted versions Max 2SAT and Max 3SAT have been developed; we summarize such previous results in Table 1. There is a corresponding history of continuous improvements in the non-approximability; we do not mention it here (the interested reader can find it in [BGS98]), and we only recall that the best known hardness is $7/8 + \varepsilon$ due to Håstad [Hås97], and it still holds when restricting to satisfiable instances with exactly three literals per clause.

Our results. We present a polynomial-time algorithm that, given a satisfiable Max SAT instance, satisfies a fraction .8 of the total weight of clauses, and an algorithm that, given a satisfiable Max 3SAT instance, satisfies a fraction .826 of the total weight of clauses.

Satisfiable instances	Arbitrary instances	Due to
.125	.125	[PY91]
.299		[BGS98]
	.25	[Tre96]
.367	.367	[TSSW96]
.514		**This paper**

Table 2: Evolution of the approximation factors for Max 3CSP with and without the satisfiability promise.

Source of our improvement. In both cases, we show how to reduce the given instance to an instance without unit clauses. The reduction sequentially applies a series of substitutions of values to variables. The .826 approximation for Max 3SAT then follows by adapting the analysis of [TSSW96] to the case of no unit clauses. The .8 approximation for Max SAT involves the use of known algorithms, with a couple of small changes.

Maximum 3-ary Constraint Satisfaction Problem (and 3-query PCP). The PCP Theorem states that membership proofs for any NP language can be probabilistically checked by a verifier that uses logarithmic randomness, has *perfect completeness*, *soundness*[1] 1/2 and *non-adaptively* reads a constant number of bits from the proof. Since its appearance, there was interest in understanding the tightest possible formulation of the PCP Theorem, especially in terms of how low the number of query bits could be made.

It is easy to see that, with two queries, it is impossible to get perfect completeness, while with 3 it is possible (see e.g. [BGS98]). The challenging question arises of determining which is the best soundness achievable with three bits and perfect completeness. The state of the art for this question is that NP can be checked with soundness $.75 + \varepsilon$ [Hås97], while this is impossible with soundness .367 [TSSW96], unless $\mathsf{P} = \mathsf{NP}$. Furthermore, it is possible to check NP with three queries, soundness $.5 + \varepsilon$ and completeness $1 - \varepsilon$ for any $\varepsilon > 0$ [Hås97]. The latter result implies that Max 3SAT is hard to approximate within $7/8 + \varepsilon$, but not when restricted to satisfiable instances. A different and more complicated proof was needed to prove the $7/8 + \varepsilon$ hardness result also for satisfiable instances [Hås97]. It was an open question whether soundness $.5 + \varepsilon$ is achievable with three queries and perfect completeness.

Our result. We show that for PCP verifiers of NP languages with three non-adaptive queries and perfect completeness, the soundness is bounded away from .5, and has to be at least .514 (unless $\mathsf{P} = \mathsf{NP}$).

Source of our improvement. We give a .514-approximate algorithm for satisfiable instances of Max 3CSP. A preprocessing step reduces the instance to an instance where any constraint has at least 3 satisfying assignments and each satisfying assignment is consistent with the set of linear constraints. We then apply two algorithms and take the best solution. In one algorithm, we reduce all the constraints to 2SAT using gadgets, extending an idea of [TSSW96]. In the other algorithm we take a random solution for the set of linear constraints.

[1] Roughly speaking, a verifier has *perfect completeness* if it accepts a correct proof with probability 1, while the *soundness* is the probability of accepting a wrong proof (see Definition 7).

Maximum k-ary Constraint Satisfaction Problem. The approximability of the Max kCSP problem is an algorithmic rephrasing of the accepting power of PCP verifiers that non-adaptively read k bits of the proof. The restriction to satisfiable instances of Max kCSP corresponds to the restriction to non-adaptive PCP verifiers with *perfect completeness*. The requirement of perfect completeness and non-adaptiveness appeared in the first definitions of PCP and in several subsequent papers [AS98, ALM+98, BGLR93, BS94]. Recently, adaptiveness (with perfect completeness) was used in [BGS98], and a verifier without perfect completeness (but non-adaptive) appears in [Hås97]. The latter result was of particular interest, because it formerly appeared that "current techniques" could only yield PCP constructions with perfect completeness. The study of which PCP classes lie in P was initiated in [BGS98]. The best known approximation for Max kCSP, for general k, is 2^{1-k} [Tre96].

Our results. We improve the approximation to $(k+1)2^{-k}$ for satisfiable instances.

Source of our improvement. We use again substitutions (but of a more general kind) as a preprocessing step. The substitutions reduce the problem to an instance where any k-ary constraint has at least $k+1$ satisfying assignments, and any such assignment is consistent with the set of linear constraints. We then take a random feasible solution for the set of linear constraints, and this satisfies each constraint with probability at least $(k+1)2^{-k}$.

Free bits. Besides the number of query bits, there is another very important parameter of the verifier that is studied in the field of probabilistic proof-checking: the number of *free bits*. It is a *relaxation* of the notion of query bit: if a verifier queries q bits on the proof, than it uses at most f free bits, but a verifier using f free bits can read arbitrarily many bits. The interest in this parameter (implicit in [FK94] and explicitly introduced in [BS94]) lies in the fact that the "efficiency" of the reduction from PCP to Max Clique [FGL+96] depends only on the number of free bits of the verifier (indeed, it depends only on the *amortized* number of free bits [BGS98], but we will not exploit the latter notion here). Since the same reduction is used to derive the best known hardness result for Min Vertex Cover, further improvements in the hardness of approximating Min Vertex Cover could be obtained by improved PCP constructions with low free bits complexity. Roughly speaking, a verifier uses f free bits if, after making its queries to the proof, there at most 2^f possible answers that make him accept (this is why f cannot be larger than the number of query bits.) This definition has been used almost always, including in Håstad's papers on Max Clique (where he used the free bit-efficient *complete test*.) One exception is [BGS98], where an *adaptive* version of the definition of free bits is used. We also mention that the free bit parameter has almost always been used for verifiers with perfect completeness (Bellare et al. [BGS98] also show that one can always reduce the free bit complexity by reducing the completeness.) However, the currently best hardness result for Min Vertex Cover is due to Håstad [Hås97] and uses a verifier with low free bit complexity and completeness $1-\varepsilon$, for any $\varepsilon > 0$.

Even in the simple case of the *non-adaptive* definition and of *perfect completeness* there were basically no result about PCP classes with low free bit complexity collapsing to P. The only result was that, with perfect completeness, it is impossible to characterize NP with only 1 free bit, while $\log 3$ free bits are sufficient [BGS98]. It has been conjectured that with $\log 3$ free bits and perfect completeness it is possible to achieve any soundness.

Our result. Under the weak (non-adaptive) definition of free bits, we prove that a verifier with perfect completeness, that uses f free bits, and whose soundness is less than $2^f/2^{2^f-1}$ can only capture P.

Source of our improvement. We adapt the previously described reductions and algorithms.

Independent and Subsequent Results. In an independent and simultaneous research, Karloff and Zwick [KZ97] found a new semidefinite relaxation of the MAX 3SAT problem, and a new way of analysing the randomized rounding of solutions of the relaxation. As a consequence of their new technique, they were able to present a $(7/8 - \varepsilon)$-approximate algorithm for MAX 3SAT, for any $\varepsilon > 0$. Such an algorithm is the best possible, since we recall that $(7/8 + \varepsilon)$-approximating MAX 3SAT is NP-hard [Hås97]. More recently, Zwick [Zwi98] applied the techniques of [KZ97] to the study of MAX 3CSP, and came up with a 1/2-approximate algorihtm, which is again the best possible. Using ideas from the present paper, Zwick [Zwi98] also improved our approximation of satisfiable instances of MAX 3CSP, developing a 5/8-approximate algorithm for this restricted problem. Guruswami et al. [GLST98] applied Zwick's ideas to MAX 4CSP, and obtained a .33-approximate algorithm for this problem (which improves and generalize our .3125-approximation for satisfiable instances). Our results for satisfiable instances of MAX kCSP are still (as of February 1999) the best known for $k \geq 5$.

Organization of the Paper. Basic definitions on constraint satisfaction problems, PCP, and gadgets are given in Section 2. We prove a simple combinatorial result in Section 3. We present the MAX SAT approximation algorithms in Section 4 and the MAX kCSP approximation algorithms (as well as the implications with PCP classes) in Sections 5 and 6. The free bit parameter is discussed in Section 7.

2 Preliminaries

For an integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$. We see elements of $\{0,1\}^n$ as vectors in the vector space $GF(2)^n$. We denote with the same symbol $\oplus$ both the boolean xor between elements of $\{0,1\}$ and the *bitwise* xor between elements of $\{0,1\}^n$. We use boldface letters to denote elements of $\{0,1\}^n$; we also denote by $\mathbf{0} = (0, \ldots, 0)$ (resp. $\mathbf{1} = (1, \ldots, 1)$) the element of $\{0,1\}^n$ all whose entries are 0 (resp. 1), where n will depend on the context. *Linear equations* will always be *linear equations on* $GF(2)$.

2.1 Constraint Satisfaction

We begin with a definition of constraint satisfaction problem, that unifies the definitions of all the problems we are interested in.

Definition 1 *A (k-ary) constraint function is a boolean function* $f : \{0,1\}^k \to \{0,1\}$.

When it is applied to variables $x_1, \ldots, x_k$ (see the following definitions) the function f is thought of as imposing the constraint $f(x_1, \ldots, x_k) = 1$.

Definition 2 *A* constraint family $\mathcal{F}$ *is a finite collection of constraint functions. The* arity *of* $\mathcal{F}$ *is the maximum number of arguments of the functions in* $\mathcal{F}$. *A* constraint C *over a variable set* $x_1, \ldots, x_n$ *is a pair* $C = (f, (i_1, \ldots, i_k))$ *where* $f : \{0,1\}^k \to \{0,1\}$ *is a constraint function and* $i_j \in [n]$ *for* $j \in [k]$. *The constraint* C *is said to be* satisfied *by an assignment* $\mathbf{a} = a_1, \ldots, a_n$ *to* $x_1, \ldots, x_n$ *if* $C(a_1, \ldots, a_n) \stackrel{\text{def}}{=} f(a_{i_1}, \ldots, a_{i_k}) = 1$. *We say that constraint* C *is* from $\mathcal{F}$ *if* $f \in \mathcal{F}$.

We will also write a constraint $(f, (i_1, \ldots, i_k))$ as $(f(x_{i_1}, \ldots, x_{i_k}) = 1)$.

Definition 3 (Satisfying table) *A satisfying table for a constraint function* $f : \{0,1\}^k \to \{0,1\}$ *with* s *satisfying assignments is a* $s \times k$ *boolean matrix whose rows are the satisfying assignments of* f.

Sometimes, we will blur the important distinction between a boolean function and a constraint, e.g. we will talk about the satisfying table of a constraint.

The satisfying table is not unique since the matrix representation imposes an order to the assignments. Even if it would be more natural to represent the satisfying assignments as a set of vectors rather than a matrix, the latter representation is more suitable for combinatorial arguments, especially because we can *see it as a set of* k *vectors of length* s (see Section 3).

Definition 4 (Constraint families) *A* literal *is either a variable or the negation of a variable. We define the following constraint families:*

kCSP: *the set of all* h*-ary functions,* $h \le k$.

kCSP^i: *the set of all* k*-ary functions with* i *satisfying assignments.*

kSAT: *the set of all functions expressible as the* `or` *of at most* k *literals.*

SAT: *the set of all functions expressible as the* `or` *of literals.*

We also sometimes see the constants 0 and 1 as *zero-ary functions*.

We say that a constraint function $f(x_1, \ldots, x_k)$ is *linear* if either $f(x_1, \ldots, x_k) = x_1 \oplus \ldots \oplus x_k$ or $f(x_1, \ldots, x_k) = 1 \oplus x_1 \oplus \ldots \oplus x_k$. (Note that linear functions are also, more appropriately, called *affine* in other papers, e.g. in [KSW97].)

Definition 5 (Constraint satisfaction problems) *For a function family* $\mathcal{F}$, Max $\mathcal{F}$ *is the optimization problem whose instances consist of* m *weighted constraints from* $\mathcal{F}$, *on* n *variables, and whose objective is to find an assignment to the variables which maximizes the total weight of satisfied constraints.*

Note that Definitions 4 and 5 give rise to the problems Max SAT, Max 3SAT, and Max kCSP, that are defined in the standard way. A constraint from SAT is also called a *clause*.

Observe that Max $\mathcal{F}$ is equivalent to Max $\mathcal{F} \cup \{0,1\}$, since adding constraints that are always false does not change a problem, while adding constraints that are always true can only make the problem easier to approximate. For this reason, we will always assume that 0-constraints and 1-constraints can occur in any Max $\mathcal{F}$ problem.

Given an instance φ of a constraint satisfaction problem, we denote by $LIN(\varphi)$ the set of linear constraints of φ.

gl1-Max $\mathcal{F}$[2] is the restriction of Max $\mathcal{F}$ to instances where all the constraints are simultaneously satisfiable.

We say that a maximization problem is r-approximable $r < 1$ if there exists a polynomial-time algorithm that, for any instance, finds a solution whose cost is at least r times the optimum (such a solution is said to be r-approximate).

[2] gl1 stands for "Gap Location 1", which is the terminology of Petrank [Pet94].

2.2 Gadgets

We also need the definition of *gadgets*.

Definition 6 (Gadget [BGS98]) *For $\alpha \in \mathcal{R}$, a function $f : \{0,1\}^k \to \{0,1\}$, and a constraint family $\mathcal{F}$: an α-*gadget *reducing f to $\mathcal{F}$ is a finite collection of constraints C_j from $\mathcal{F}$ over* primary variables *$x_1, \ldots, x_k$ and* auxiliary variables *$y_1, \ldots, y_n$, and associated real weights $w_j \geq 0$, with the property that, for boolean assignments $\mathbf{a}$ to $x_1, \ldots, x_k$ and $\mathbf{b}$ to $y_1, \ldots, y_n$, the following are satisfied:*

$$(\forall \mathbf{a} : f(\mathbf{a}) = 1)\ (\forall \mathbf{b}) : \sum_j w_j C_j(\mathbf{a}, \mathbf{b}) \quad \leq \quad \alpha, \tag{1}$$

$$(\forall \mathbf{a} : f(\mathbf{a}) = 1)\ (\exists \mathbf{b}) : \sum_j w_j C_j(\mathbf{a}, \mathbf{b}) \quad = \quad \alpha, \tag{2}$$

$$(\forall \mathbf{a} : f(\mathbf{a}) = 0)\ (\forall \mathbf{b}) : \sum_j w_j C_j(\mathbf{a}, \mathbf{b}) \quad \leq \quad \alpha - 1. \tag{3}$$

Gadgets can be used in approximation algorithms in the following way [TSSW96]. Assume we have a satisfiable instance of a constraint satisfaction problem, with constraints of total weight m, and there is α-gadget reducing each such constraint to 2SAT. Then we can build a 2SAT instance ψ whose optimum is αm and such that any solution of cost c for ψ has cost at least $c - (\alpha - 1)m$ for the old instance.

In a more general setting, assume that, for $i = 1, \ldots, k$, we have type-i constraints of total weight w_i, and that there exists an α_i-gadget reducing type-i constraints to 2SAT. Assume also that the whole CSP instance be satisfiable. Then the optimum of the instance is $\sum_i w_i$; applying all the gadgets we have a 2SAT instance ψ whose optimum is $\sum_i \alpha_i w_i$.

Applying a β-approximate algorithm to ψ, we obtain a solution for the original instance whose cost is at least

$$\sum_i \beta \alpha_i w_i - \sum_i (\alpha_i - 1) w_i = \sum_i (\beta - (1 - \beta)(\alpha_i - 1)) w_i \ .$$

In the following, we will refer to such kind of reductions as the *TSSW technique* (see Section 2.5.2 below). The FGW [GW95, FG95] algorithm for Max 2SAT is .931-approximate.

2.3 Probabilistically Checkable Proofs

We now talk about PCP classes and their relation with the approximability of Max kCSP.

Definition 7 (Restricted verifier) *A* verifier *V for a language L is a probabilistic polynomial-time Turing machine that during its computations has oracle access to a string called* proof. *We denote by $\mathbf{ACC}[V^\pi(x)]$ the probability over its random tosses that V accepts x when accessing proof π. We also denote by $\mathbf{ACC}[V(x)]$ the maximum of $\mathbf{ACC}[V^\pi(x)]$ over all proofs π. We say that*

- V has query complexity q *(where q is an integer) if for any input x, any proof π, and any outcome of its random bits, V reads at most q bits from π;*
- V has soundness s *if, for any $x \notin L$, $\mathbf{ACC}[V(x)] \leq s$;*
- V has completeness c *if, for any $x \in L$, $\mathbf{ACC}[V(x)] \geq c$. V has* perfect completeness *if it has completeness* 1.

Definition 8 (PCP classes) *$L \in \mathsf{PCP}_{c,s}[\log, q]$ if L admits a verifier V with completeness c, soundness s, query complexity q, and that uses $O(\log n)$ random bits, where n is the size of the input. We say that $L \in \mathbf{na}\mathsf{PCP}_{c,s}[\log, q]$ if V, in addition, queries the q bits* non-adaptively.

Theorem 9 ([ALM$^+$98]) *If* GL1-MAX kCSP *is r-approximable, then $\mathbf{na}\mathsf{PCP}_{1,s}[\log, k] \subseteq \mathsf{P}$ for any $s < r$.*

Note that the relation between PCP classes and MAX kCSP problems holds also in the case of non-perfect completeness and for adaptive verifiers.

Theorem 10 ([Tre96]) *If* MAX kCSP *is r-approximable, then $\mathsf{PCP}_{c,s}[\log, k] \subseteq \mathsf{P}$ for any $c/s < r$.*

We define free bits as a property of boolean functions. There are two possible definitions.

Definition 11 *A function $f : \{0,1\}^q \to \{0,1\}$ uses f* non-adaptive free bits *if it has at most 2^f satisfying assignments. It uses f* adaptive free bits *if it can be expressed by a DNF with at most 2^f terms such that any two terms are inconsistent.*

It is easy to see that if a function uses f non-adaptive free bits that it also uses at most f adaptive free bits. On the other hand, there are functions using 1 adaptive free bit and requiring arbitrarily many non-adaptive free bits.

Definition 12 *A PCP verifier uses f adaptive (resp. non-adaptive) free bits if for any input, and any fixed random string, its acceptance or rejectance (which is a bollean function of the proof) can be expressed as a boolean function that uses f adaptive (resp. non-adaptive) free bits. $\mathsf{FPCP}_{c,s}[\log, f]$ is the class of languages admitting a PCP verifier with logarithmic randomness, completeness c, soundness s, that uses f adaptive free bits. The class $\mathbf{na}\mathsf{FPCP}_{c,s}[\log, f]$ is defines analogously by using the non-adaptive free bit parameter.*

Regarding recent constructions of verifiers optimized for the free bit parameter, the verifiers that use the *Complete Test* [Hås96b] are non-adaptive, while the verifier that uses the *Extended Monomial Basis Test* [BGS98] is adaptive.

The notion of free bit was originally introduced to prove hardness results for MAX CLIQUE. An amortized version of the free bit parameter has been defined in [BS94]. We will not deal with this amortized version in this paper, since an essentially tight result has been established: Håstad [Hås96a] has shown that it is possible to characterize NP using $\varepsilon > 0$ amortized free bits, for any fixed $\varepsilon > 0$.

For the non-amortized version of this parameter, it is still an open question to find the best possible characterizations of NP. Improved PCP constructions with low free bit complexity are also motivated by the following application to the MIN VERTEX COVER problem. (Recall that an r-approximate algorithm for vertex Cover, for $r \geq 1$, is an algorithm that computes a cover whose number of nodes is at most r times the size of the optimum cover.)

Theorem 13 ([BGS98]) *If $\mathsf{NP} \subseteq \mathsf{FPCP}_{c,s}[\log, f]$, then, for any $\varepsilon > 0$, it is NP-hard to approximate* MIN VERTEX COVER *within*

$$1 + \frac{c - s}{2^f - c} - \varepsilon$$

The best result in this respect, due to Håstad, is that $\mathsf{NP} = \mathsf{FPCP}_{1-\epsilon, .5+\epsilon}[\log, 2]$ for any $\varepsilon > 0$. This implies that Vertex Cover is hard to approximate within $7/6 - \varepsilon$ for any $\varepsilon > 0$.

2.4 Substitutions

Let φ be a set of weighted constraints from $\mathcal{F}$ over variables $x_1, \ldots, x_n$. Let $x_i = a_0 \oplus \bigoplus_{j\in[n]-\{i\}} a_j x_j$ with $a_j \in \{0,1\}$ for $j \in \{0,\ldots,n\} - \{i\}$ be a linear equation. Let $C \equiv (f(x_{i_1},\ldots,x_{i_k}) = 1)$ be a constraint of φ. Then the *application* of the *substitution* $\sigma = [x_i \leftarrow a_0 \oplus \bigoplus_{j\in[n]-\{i\}} a_j x_j]$ to C (denoted by $C\sigma$) is defined as follows.

1. If x_i does not occur in C, then C is left unchanged by the substitution. More formally, if $i \notin \{i_1,\ldots,i_k\}$ then $C[x_i \leftarrow a_0 \oplus \bigoplus_{j\in[n]-\{i\}} a_j x_j] \equiv C$.
2. If x_i occurs in C, then the occurrence of x_i is substituted by the expression $a_0 \oplus \bigoplus_{j\in[n]-\{i\}} a_j x_j$. More formally, if $i = i_h$ for some $h \in [k]$, then

$$C[x_i \leftarrow a_0 \oplus \bigoplus_{j\in[n]-\{i\}} a_j x_j] \equiv (f(x_{i_1},\ldots,x_{i_{h-1}}, a_0 \oplus \bigoplus_{j\in[n]-\{i\}} a_j x_j, x_{i_{h+1}},\ldots,x_{i_k}) = 1)$$

We note that in the second case, the set of variables occurring in the constraint becomes $\{i_1,\ldots,i_k\} - \{i\} \cup \{j \in [n] - \{i\} : a_j = 1\}$.

The *width* of a substitution is the number of non-zero coefficients a_j for $j \geq 1$. Thus, a width-zero substitution is of the form $x \leftarrow 0$ or $x \leftarrow 1$, and always decreases the arity of the constraint it is applied to. A width-one substitution is of the form $x \leftarrow y$ or $x \leftarrow \neg y$, and it either leaves the arity of the constraint unchanged, or it decreases the arity.

We note that if we apply a widht-one or a widht-two substitution to a SAT (resp. 3SAT) constraint, then we obtain another SAT (resp. 3SAT) constraint[3]. Additionally, if we apply to a kCSP constraint C a substitution σ such that all the variables occurring in the left-hand side of the equation of σ already occur in C, then $C\sigma$ is a $(k-1)$CSP constraint.

For an instance φ of a constraint satisfaction problem and substitution σ, we denote by $\varphi\sigma$ the instance obtained by applying the substitution σ to all the constraints of φ.

2.5 Approximation Algorithms and Techniques for Max SAT

For the rest of this section we fix a satisfiable instance φ of Max SAT, that has clauses of total weight m. For any $i \geq 1$, m_i is the total weight of clauses with exactly i literals. We examine different algorithms, and different ways to extend/mix them. Under the assumption that the formula be satisfiable, the cost of the solutions provided by all the algorithms below will always be lower bounded by some linear combination of the m_i's.

2.5.1 Algorithms

Johnson's algorithm [Joh74]. It finds a solution that satisfies clauses of total weight at least

$$\sum_i \left(1 - \frac{1}{2^i}\right) m_i \ .$$

FGW Algorithm [GW95, FG95]. Given an instance of Max 2SAT, it satisfies at least a fraction $\beta = .931$ of the cost of an optimum solution.

[3] We can indeed obtain the 0-constraint or the 1-constraint, but we noted that we can assume 3SAT and SAT contain such constraint without loss in generality.

2SAT algorithm. If a 2SAT instance is satisfiable, it is possible to find a satisfying assignment in polynomial time.

Linear systems mod 2. If a system of linear equations over variables $x_1, \ldots, x_n$ is satisfiable, we can find in polynomial time an explicit description for the set of its solution, that is, a vector $\mathbf{u} = (u_1, \ldots, u_n) \in \{0,1\}^n$ that is a feasible solution, and vectors $\mathbf{y}^1, \ldots, \mathbf{y}^k$ such that the set of feasible solutions is precisely

$$\{\mathbf{u} \oplus a_1 \mathbf{y}^1 \oplus \cdots \oplus a_k \mathbf{y}^k : a_1, \ldots, a_k \in \{0,1\}\} \ .$$

2.5.2 Techniques

GW Technique [GW95]. It allows to extend a β-approximate algorithm for Max 2SAT (e.g. the FGW algorithm) to the Max SAT problem. Here we present a simplified analysis of the GW technique that is sufficient to deal with satisfiable instances of Max SAT.

Any clause of length $k \geq 3$ and weight w is substituted by the $\binom{k}{2}$ clauses of length 2 obtained by taking in all the possible ways two literals out of k. Each new clause receives weight $w/\binom{k}{2}$. We then apply a β-approximate algorithm to the resulting instance of Max 2SAT.

The number of clauses in the original formula that are satisfied in this way is at least

$$\beta(m_1 + m_2) + \sum_k \frac{2}{k} \beta m_k \ .$$

TSSW Technique [TSSW96]. This technique has been already described in Section 2.2. As an application to Max SAT, we note that if, for some k, we have an α_k-gadget reducing kSAT to 2SAT, then we can substitute $(\beta - (1-\beta)(\alpha_k - 1))$ in place of $\frac{2}{k}\beta$ in the analysis of the GW technique.

CT Technique [CT97]. This technique is parameterized with an integer k and a real $0 \leq p \leq 1$. If we have an algorithm for gl1-Max kSAT which satisfies clauses of length i of total weight $\rho_i m_i$, $1 \leq i \leq k$, then the CT technique allows to design an algorithm for gl1-Max SAT that satisfies clauses of total weight

$$\sum_{i=1}^{k} (1 - p(1-p)^{i-1}) \rho_i m_i + \sum_{i \geq k+1} (1 - (1-p)^i) m_i \ .$$

3 An Application of the Linear Algebra Method

The linear algebra method in combinatorics [BF92] is a collection of techniques that allow to prove combinatorial results by making use of the following well-known fact: if we have a set of n-dimensional vectors that are linearly independent, then the size of the set is at most n. In this section we will provide some definitions and prove easy bounds using linear algebra. Despite the triviality of the results, they will have powerful applications in Sections 5 and 6.

Definition 14 *For a vector $\mathbf{u} \neq \mathbf{0}$, we say that a collection $\mathbf{x}_1, \ldots, \mathbf{x}_m$ of elements of $\{0,1\}^n$ is* $(k, \mathbf{u})$-dependent *if there are values $a_0, \ldots, a_m \in \{0,1\}$ such that $1 \leq |\{i = 1, \ldots, m : a_i = 1\}| \leq k$ and $a_1 \mathbf{x}_1 \oplus \ldots \oplus a_m \mathbf{x}_m = a_0 \mathbf{u}$. A collection is* $\mathbf{u}$-dependent *if it is $(k, \mathbf{u})$-dependent for some k. A collection is* $((k, \mathbf{u})$-$)$independent *if it is not $((k, \mathbf{u})$-$)$dependent.*

More intuitively, the vectors $\mathbf{x}_1, \dots, \mathbf{x}_m$ are $(k, \mathbf{u})$-independent if any xor of at most k of them is different from $\mathbf{0}$ and from $\mathbf{u}$.

Lemma 15 *For any $\mathbf{u} \neq \mathbf{0}$, if $\mathbf{x}_1, \dots, \mathbf{x}_m \in \{0,1\}^n$ are $(2, \mathbf{u})$-independent, then $m \leq 2^{n-1} - 1$. The bound is tight.*

PROOF: All the $2m+2$ vectors $\mathbf{0}, \mathbf{x}_1, \dots, \mathbf{x}_m, \mathbf{u}, (\mathbf{u} \oplus \mathbf{x}_1), \dots, (\mathbf{u} \oplus \mathbf{x}_m)$ are distinct. Therefore $2m + 2 \leq 2^n$. To prove the tightness of the bound, let $\mathbf{u} = \mathbf{1}$ and consider the set of $2^{n-1} - 1$ vectors of $\{0,1\}^n - \{\mathbf{0}\}$ whose first entry is zero. Clearly, these vectors form a $(2, \mathbf{1})$-independent collection. $\square$

Lemma 16 *For any $\mathbf{u} \neq \mathbf{0}$, if $\mathbf{x}_1, \dots, \mathbf{x}_m \in \{0,1\}^n$ are $\mathbf{u}$-independent, then $m \leq n-1$. The bound is tight.*

PROOF: The $m+1$ vectors $\mathbf{u}, \mathbf{x}_1, \dots, \mathbf{x}_m$ are distinct and linearly independent in the ordinary sense. Therefore $m + 1 \leq n$. To prove the tightness, let $\mathbf{u} = \mathbf{1}$ and consider a linearly independent set of $n-1$ vectors of $\{0,1\}^n$ whose first entries are zero. Such a set must exist, since the set of elements of $\{0,1\}^n$ whose first entry is zero form a linear subspace of $\{0,1\}^n$ of dimension $n-1$. It is easy to see that such a set is a $\mathbf{1}$-independent collection. $\square$

In the following sections we will use the special case where $\mathbf{u} = \mathbf{1}$. Let f be a k-ary constraint function with s satisfying assignments, and M be a satisfying table for f. If the columns of M are $(2, \mathbf{1})$-independent, then $k \leq 2^{s-1} - 1$, that is $s \geq 1 + \lceil \log(k+1) \rceil$, which implies $s = 2$ if $k = 1$ and $s \geq 3$ if $k \geq 2$. If the columns of M are $\mathbf{1}$-independent, then we can draw the stronger statement $s \geq k + 1$.

4 The MAX SAT Algorithms

Lemma 17 *If* GL1-MAX SAT *(resp.* GL1-MAX 3SAT*) restricted to instances without unit clauses is r-approximable, then it is r-approximable for arbitrary instances.*

PROOF: We describe an algorithm that given a satisfiable instance φ of MAX SAT over variables $x_1, \dots, x_n$, finds a new satisfiable instance φ' over a subset of variables $X' \subseteq \{x_1, \dots, x_n\} = X$ and a set S of width-1 linear equations, such that any assignment to X' can be extended to an assignment to X using the equations of S. The important properties of φ' are that φ' has no unit clause, and an r-approximate solution for φ', when extended with S, becomes an r-approximate solution for φ.

The algorithm is described in Figure 1.

Claim 18 `NoUnary` *runs in polynomial time.*

PROOF: Each substitution requires at most linear time. Furthermore, each substitution reduces the number of variables of φ. It follows that there can be at most a linear number of substitutions and thus the algorithm runs at most in quadratic time. (A more careful analysis would show that in fact a linear time implementation is possible. We choose not to do so since this would not improve the performance of the overall algorithm, that is dominated by the running time of the FGW algorithm.) $\square$

Let now φ be some satisfiable instance of MAX SAT over variable set X, and let $\texttt{NoUnary}(\varphi) = (\varphi', S)$. The following two claims are easily proved by induction on the number of substitutions performed by the algorithm.

```
algorithm NoUnary
input: φ;
begin
  S := ∅;
  while φ has unit clauses do begin
    for each positive unit clause (x) ∈ φ do begin
      S := S ∪ {(x = 1)};
      φ := φ[x ← 1]
    end;
    for each negative unit clause (¬x) ∈ φ do begin
      S := S ∪ {(x = 0)};
      φ := φ[x ← 0]
    end;
  end;
  return (φ, S);
end.
```

Figure 1: The MAX SAT reduction.

Claim 19 *φ' is satisfiable. Indeed, any satisfying assignment for φ is a satisfying assignment for φ' and is consistent with S.*

Claim 20 *Let $\mathbf{a}'$ be an assignment to X', and $\mathbf{a}$ be the extension of $\mathbf{a}'$ to X using S. Then the total weight of clauses of φ' that are satisfied by $\mathbf{a}'$ equals the total weight of clauses of φ that are satisfied by $\mathbf{a}$.*

Claim 21 *If φ is an instance of* MAX 3SAT *then φ' is an instance of* MAX 3SAT*.*

PROOF: A substitution can never make a clause longer. □

The lemma now follows. □

Lemma 22 *There exists a polynomial-time .826-approximate algorithm for* GL1-MAX 3SAT *without unit clauses.*

PROOF: We use the TSSW technique applied to the FGW algorithm and Johnson's algorithm. We assume we have an α-gadget reducing 3SAT to 2SAT, and we use a β-approximate algorithm for MAX 2SAT. Let φ be an instance of MAX 3SAT with no unit cluase. Let m be the total weight of clauses of φ, m_2 be total weight of binary clauses and m_3 be the total weight of ternary clauses. If we denote by $A(\varphi)$ the total weight of clauses satisfied by the above outlined algorithm we obtain the following relation:

$$\begin{aligned} A(\varphi) &\geq \max\left\{\frac{3}{4}m_2 + \frac{7}{8}m_3, \beta m_2 + (\beta - (1-\beta)(\alpha-1))m_3\right\} \\ &= \max\left\{\frac{3}{4}m + \frac{1}{8}m_3, \beta m - (1-\beta)(\alpha-1))m_3\right\} \end{aligned}$$

$$\begin{aligned}
&\geq \left(\frac{8(1-\beta)(\alpha-1)}{1+8(1-\beta)(\alpha-1)}\right)\left(\frac{3}{4}m+\frac{1}{8}m_3\right)+\left(\frac{1}{1+8(1-\beta)(\alpha-1)}\right)(\beta m-(1-\beta)(\alpha-1)m_3) \\
&= \frac{1}{1+8(1-\beta)(\alpha-1)}(6m(1-\beta)(\alpha-1)+m3(1-\beta)(\alpha-1)+\beta m-m_3(1-\beta)(\alpha-1)) \\
&= m\frac{\beta+6(1-\beta)(\alpha-1)}{1+8(\alpha-1)(1-\beta)}
\end{aligned}$$

where first equality follows from the substitution $m_2 = m - m_3$, and the next step uses a convex combination in place of max. For $\beta = .931$ and $\alpha = 3.5$ the last term evaluates to .82605, □

For the MAX SAT analysis we resort to a methodology of analysis introduced in [AE98]. We have to study the performance of an approximation algorithm that consists of running several approximation algorithms and then take the best solution. If the performance of each individual approximation algorithm is a linear function of some parameters of the instance (e.g. the total weight of i-ary clauses), then finding the *worst case for the best algorithm* can be formulated as a linear program. We could have applied the same methodology in the proof of Lemma 22, but we liked it better to present a traditional analysis because it gives a closed formula that depends only on the quality of the approximation and of the gadget.

Lemma 23 *There exists a polynomial-time* .8*-approximate algorithm for* GL1-MAX SAT *without unit clauses.*

PROOF: We use

J: Johnson's algorithm;

FGW: the FGW algorithm, extended to length-3 and lenght-4 clauses with the TSSW method, and to longer clauses with the GW method. We use a 3.5-gadget for length-3 clauses and a new 6-gadget for length-4 clauses (see Lemma 52 below);

CT_p: the 2SAT algorithm extended to longer clauses with the CT methodology. We use all the possible values of p, $0 \leq p \leq .5$ with a sufficiently small increment (e.g. $\delta = .001$.)

For an instance φ of MAX SAT with no unit cluase, let m_i be the total weight of the clauses with exactly i literals, and let $A(\varphi)$ be the total weight of the clauses satisfied by our algorithm. Then,

$$\begin{aligned}
A(\varphi) \geq \max \quad \{ \quad & \sum_i \left(1-\frac{1}{2^i}\right)m_i, \\
& .931m_2 + .7585m_3 + .586m_4 + \sum_i \frac{2}{i}m_i, \\
& \max_{p\in\{0,\delta,2\delta,\ldots,.5\}} \left\{p(1-p)m_2 + \sum_i (1-(1-p)^i)m_i\right\} \\
\}
\end{aligned}$$

It is easy to cast the search for a worst case as a linear program [AE98]. The only seeming difficulty is that we would need infinitely many variables $m_2, m_3, \ldots$ Indeed, we can choose a large enough upper bound N (e.g. $N = 100$), and then we use variables $m_2, \ldots, m_N$ to represent the total weight of clauses with $2, \ldots, N$ literals, and a variable m_∞ to represent the total weight of clauses with more than N literals. The contribution of clauses with more than N literals to the FGW algorithm is ignored; the contribution to the other algorithms is lower bounded with the contribution that there would be if they were all with $N+1$ literals. An optimal solution to the following linear program is thus a lower bound to the performance ration of the combined algorithm.

$$
\begin{array}{ll}
\min & ratio \\
\text{Subject to} & \\
& ratio \geq \sum_{i=1}^{N} \left(1 - \frac{1}{2^i}\right) m_i + \left(1 - \frac{1}{2^{N+1}}\right) m_\infty \\
& ratio \geq .931 m_2 + .7585 m_3 + .586 m_4 + \sum_{i=1}^{N} \frac{2}{i} m_i, \\
& ratio \geq p(1-p)m_2 + \sum_{i=1}^{N}(1-(1-p)^i)m_i + (1-(1-p)^{N+1})m_\infty \quad p = i\delta \text{ for } i = 0, \ldots, \frac{1}{2\delta} \\
& \sum_{i=1}^{N} m_i + m_\infty = 1 \\
& m_i, m_\infty \geq 0
\end{array}
$$

For $\delta = .001$ and $N = 100$ the linear program above has an optimal solution $ratio = .8000939$. The same solution can be obtained using only one value of p, namely $p = .082$ (the value .082 has been found with numerical experiments.) □

Theorem 24 *There exists a .8-approximate algorithm for* GL1-MAX SAT *and a .826-approximate algorithm for* GL1-MAX 3SAT.

5 The MAX 3CSP Algorithm

Definition 25 *We say that an instance φ of* MAX kCSP *is* simplified *if the following conditions hold for any constraint $C = (f, (i_1, \ldots, i_k))$ of φ:*

1. *the columns of a satisfying table of C are $(2, \mathbf{1})$-independent;*
2. *either C is linear or all its satisfying assignments are consistent with $LIN(\varphi)$. More formally, if f is not linear, then for any values $a_1, \ldots, a_k \in \{0,1\}$ such that $f(a_1 \ldots, a_k) = 1$, the system $LIN(\varphi) \cup \{(x_{i_1} = a_1), \ldots, (x_{i_k} = a_k)\}$ is satisfiable.*

Lemma 26 *If* GL1-MAX 3CSP *is r-approximable when restricted to simplified instances, then* GL1-MAX 3CSP *is r-approximable.*

PROOF: Given a general instance φ of GL1-MAX 3CSP, we reduce it to a simplified instance φ'. The reduction is described in Figure 2.

The proof now continues with the same structure of the proof of Lemma 17.

Claim 27 *For any instance φ,* `Simplify`(φ) *terminates in polynomial time.*

PROOF: Each step either reduces the size of the satisfying table of a constraint or reduces the number of variables occurring in φ. Thus, the number of possible steps is bounded by the lenght of the description of φ. Each step can be implemented in polynomial time. □

Let now φ be a satisfiable instance of MAX 3CSP over variable set X, and let `Simplify`$(\varphi) = (\varphi', S)$. Let X' be the variable set of φ'.

Claim 28 *φ' is satisfiable. Indeed, any satisfying assignment for φ is a satisfying assignment for φ' and is consistent with S.*

algorithm `Simplify`;
begin
  $S := \emptyset$;
  while φ is not simplified **do begin**
    for each constraint $C \equiv (f(x_{i_1}, \ldots, x_{i_k}) = 1)$ with a $(2, \mathbf{1})$-dependent satisfying table
    do begin
      Comment: C enforces a linear relation $(x_{i_j} = a_0 \oplus a_h x_{i_h})$;
      $\varphi := \varphi[x_{i_j} \leftarrow a_0 \oplus a_h x_{i_h}]$;
      $S := S \cup \{(x_{i_j} = a_0 \oplus a_h x_{i_h})\}$;
    end;
    for each constraint C one whose satisfying assignment is inconsistent with $LIN(\varphi)$
      remove such assignment from the satisfying table of C;
  end;
  return (φ, S);
end

Figure 2: Algorithm reducing an instance of GL1-MAX 3CSP to a simplified instance.

PROOF: By induction on the number of substitutions performed by the algorithm. □

Claim 29 *Let* $\mathbf{a}'$ *be an assignment to* X'*, and* $\mathbf{a}$ *be the extension of* $\mathbf{a}'$ *to* X *using* S*. Then the total weight of clauses of* φ' *that are satisfied by* $\mathbf{a}'$ *equals the total weight of clauses of* φ *that are satisfied by* $\mathbf{a}$.

□

Lemma 30 GL1-MAX 3CSP *restricted to simplified instances is* .5145*-approximable.*

PROOF: From Lemma 15, φ has no unit constraint (besides the always true constraint), the 2-ary constraints can only be from 2SAT, the 3-ary constraints must have at least three satisfying assignments. Let m_2 be the total weight of 2SAT constraints, $m^{(3)}$, $m^{(4)}$, $m^{(5)}$, $m^{(6)}$, $m^{(7)}$ be the total weight of 3-ary constraints that have, respectively, 3, 4, 5, 6, and 7 satisfying assignments. We also let $m^{(4L)}$ be the total weight of 3-ary linear constraints and $m^{(4O)} = m^{(4)} - m^{(4L)}$.

We use two algorithms and take the best solution.

In the first algorithm, we simply consider a random feasible solution for $LIN(\varphi)$. On the average, the total weight A_{RAND} of satisfied constraints is at least

$$A_{RAND} \geq \frac{3}{4}\, m_2 + \frac{3}{8}\, m^{(3)} + \frac{4}{8}\, m^{(4O)} + \; m^{(4L)} + \frac{5}{8}\, m^{(5)} + \frac{6}{8}\, m^{(6)} + \frac{7}{8}\, m^{(7)}$$

Derandomization is possible using the method of conditional expectation.

The other algorithm uses the TSSW method and the FGW algorithm. We have to find gadgets reducing the various possible 3-ary constraints to 2SAT constraints. The new constructions (and the old ones that we use) are listed in Table 3. More details are available in Section 8. Using the FGW algorithm with the TSSW method and the gadgets of Table 3, we have an algorithm that satisfies constraints of total weight

Source Constraint	Target Constraint	α	Due to
3SAT	2SAT	3.5	[TSSW96]
4SAT	2SAT	6	**This paper**
3CSP^3	2SAT	5.5	**This paper**
3CSP^4 not linear	2SAT	5.5	**This paper**
3CSP^4 linear	2SAT	11	[BGS98]
3CSP^5	2SAT	8.25	**This paper**
3CSP^6	2SAT	5.5	**This paper**

Table 3: Gadgets used.

$$A_{GAD} \geq .931\ m_2 + .6205\ m^{(3)} + .241\ m^{(4L)} + .6205\ m^{(4O)} + .43075\ m^{(5)} + .6205\ m^{(6)} + .7585\ m^{(7)}$$

If we take the best solution, then we satisfy constraints of total weight

$$\max\{A_{RAND}, A_{GAD}\} \geq .4315 A_{RAND} + 0.5685 A_{GAD} \geq .5145m$$

where $m = m_2 + m^{(3)} + m^{(4L)} + m^{(4O)} + m^{(5)} + m^{(6)} + m^{(7)}$. □

Theorem 31 *There exists a polynomial-time* .5145*-approximate algorithm for* GL1-MAX 3CSP.

Theorem 32 $\mathbf{na}\mathsf{PCP}_{1,.514}[\log, 3] \subseteq \mathsf{P}$.

Recall that Håstad [Hås97] proved that $\mathbf{na}\mathsf{PCP}_{1-\varepsilon,\frac{1}{2}+\varepsilon}[\log, 3] = \mathsf{NP}$ and that $\mathbf{na}\mathsf{PCP}_{1,\frac{3}{4}+\varepsilon}[\log, 3] = \mathsf{NP}$ for any $\varepsilon > 0$

6 The MAX kCSP Algorithm

The general pattern of the results of this paper is that we take an instance of a constraint satisfaction problem and, using approximation preserving reductions, we transform it into an instance where each constraint has a "reasonably large" set of satisfying assignment. For the new instance, the approach of taking a random solution will then work well enough.

In order to bound from below the number of satisfying assignments of the constraints in the new instance, we use algebraic properties of the satisfying table. For the MAX SAT algorithm we just use the fact that the new instance has no unit clause. For MAX 3CSP we go one step further, and we use the fact that the constraints of the new instance have a $(2, \mathbf{1})$-independent satisfying table. For MAX kCSP we will enforce full $\mathbf{1}$-independency among the row of the satisfying table.

It turns out that it is easier to define an approximation algorithm for a generalization of MAX kCSP.

Definition 33 (Max kCSP+LIN) *For any positive integer k, the* Max kCSP+LIN *problem is defined as follows: an* instance *is a pair (φ, S), where φ is an instance of* Max kCSP *and S is a system of linear equations. A* feasible solution *is an assignment of $0/1$ values to the variables occurring in φ and S, such that all the equations of S are satisfied. The* measure *of a solution is the total weight of satisfied constraints in φ.*

Thus, the difference between Max kCSP+LIN and Max kCSP is that in the former problem we also have a set of *mandatory* linear constraints that must be satisfied by any feasible solution. Max kCSP can be seen as the special case of Max kCSP+LIN where the set S is empty.

Definition 34 *We say that an instance (φ, S) of* Max kCSP+LIN *is* over-simplified *if, for any constraint C of φ, the following properties hold:*

1. *the columns of the satisfying table of C are $\mathbf{1}$-independent;*
2. *any assignment to the variables occurring in C that satisfies C is consistent with S.*

Note that the first property implies that in an over-simplified instance (φ, S) no constraint in φ can be linear.

Lemma 35 *If* GL1-Max kCSP+LIN *is r-approximable when restricted to over-simplified instances, then* GL1-Max kCSP+LIN *is r-approximable.*

Proof: Let (φ, S) be an instance of GL1-Max kCSP+LIN. Let $(\varphi', S') = \texttt{OverSimp}(\varphi, S)$. It is easy to see that `OverSimp` has a polynomial-time implementation. The following claims prove the lemma.

Claim 36 *(φ', S') is an instance of* GL1-Max kCSP.

Proof: Algorithm `OverSimp` never replaces a constraint with a constraint of bigger arity. Thus φ' is an instance of Max kCSP. By induction on the number of substitutions performed by the algorithm, we can prove that any satisfying assignment for (φ, S) is also a satisfying assignment for (φ', S') □

Claim 37 *If an assignment $\mathbf{a}'$ is feasible and r-approximate for (φ', S') then it is feasible and r-approximate for (φ, S).*

Proof: By induction on the number of steps. □

□

Lemma 38 GL1-Max kCSP+LIN *is $(k+1)/2^k$-approximable when restricted to simplified instances.*

Proof: Let (φ, S) be a simplified instance of Max kCSP+LIN. If $h \leq k$ is the arity of a constraint C in a φ, then, by Lemma 16 C has at least $h+1$ satisfying assignments, and a random assignment satisfies it with probability at least $(h+1)/2^h \geq (k+1)/2^k$. If we take a random feasible solution for S, it satisfies all constraints of S and, on the average, a fraction at least $(k+1)/2^k$ of the total weight of the constraints of φ. Derandomization is possible with the method of conditional expectation. □

algorithm `OverSimp`
input: φ, S;
begin
  while (φ, S) is not over-simplified **do begin**
    for each $C \in \varphi$ that is linear **do begin**
      $\varphi := \varphi - \{C\}$
      $S := S \cup \{C\}$;
    end;
    for each $C \equiv (f(x_{i_1}, \ldots, x_{i_k}) = 1) \in \varphi$ with a **1**-dependent satisfying table
    do begin
      Comment: C enforces a linear relation $x_{i_j} = a_0 \oplus \bigoplus_{h \neq j} a_h x_{i_h}$;
      $C := C[x_{i_j} \leftarrow a_0 \oplus \bigoplus_{h \neq j} a_h x_{i_h}]$;
      $S := S \cup \{x_{i_j} = a_0 \oplus \bigoplus_{h \neq j} a_h x_{i_h}\}$;
    end;
    for each $C \in \varphi$ one whose satisfying assignments is inconsistent with S **do**
      remove such assignment from the satisfying table of C;
  end;
  return (φ, S);
end

Figure 3: The algorithm that reduces a pair (φ, S) to an over-simplified pair (φ', S').

Theorem 39 *There exists a polynomial-time $(k+1)/2^k$-approximate algorithm for* GL1-MAX kCSP.

Theorem 40 *For any $q \geq 3$, for any $s < (q+1)/2^q$,* $\mathbf{na}\mathsf{PCP}_{1,s}[\log, q] \subseteq \mathsf{P}$.

Observe that the bound of Lemma 39 above is 1/2 for MAX 3CSP.

7 Free Bits

We now state some results (the first ones with $f > 1$) about **na**FPCP classes that collapse to P.

Theorem 41 *The following statements hold:*

1. $\mathbf{na}\mathsf{FPCP}_{1,s}[\log, f] \subseteq \mathbf{na}\mathsf{PCP}_{1,s}[\log, 2^{2^f-1} - 1]$.
2. $\mathbf{na}\mathsf{FPCP}_{1,s}[\log, f] \subseteq \mathsf{P}$ *for all $s > (2^f)/2^{2^f-1}$ and $f \geq \log 3$.*

PROOF: Let $L \in \mathbf{na}\mathsf{FPCP}_{1,s}[\log, f]$. Let V be the verifier witnessing this fact. Given x, we can find in polynomial time an instance φ_x of a constraint satisfaction problem with the following property:

- Each constraint uses f non-adaptive free bits, that is, has at most 2^f satisfying assignments.
- If $x \in L$ then φ_x is satisfiable, otherwise at most a fraction s of the constraints is satisfiable.

Indeed, φ_x has a constraint C_R for any random string R of V with input x. The weight of C_R is the probability of the random string being selected. A boolean variable is associated to each position of the proof, and the constraint C_R is satisfied precisely by the assignments corresponding with proofs that would be accepted by V with input x and random string R.

We run algorithm `Simplify` with input φ_x. We obtain an instance ψ_x and a set of equations S that enjoy the following properties:

1. No constraint C of ψ_x has more than 2^f satisfying assignments (`Simplify` never makes the satisfying table of a constraint larger);

2. Any constraint in ψ_x has a $(2, \mathbf{1})$-independent satisfying table;

3. If φ_x is satisfiable then ψ_x is satisfiable;

4. An assignment satisfying a fraction s of the constraints of ψ_x can be extended, using S, to an assignment satisfying at least a fraction s of the constraints of φ_x.

From the first two properties, it follows that no constraint in ψ_x has arity larger than $2^{2^f-1} - 1$. From the other properties we have:

- If $x \in L$ then ψ_x is satisfiable.

- If $x \notin L$ then at most a fraction s of the constraints of ψ_x is satisfiable.

We now describe a verifier for L that uses logarithmic randomness, has perfect completeness, soundness s and queries $2^{2^f-1} - 1$ bits. Given x, the verifier computes ψ_x, and assumes that the proof contains a satisfying assignment for ψ_x. The verifier picks a random constraint of ψ_x, reads from the proof the value of the variables occurring in the constraint, and accepts if and only if the constraint is satisfied. Clearly, the verifier reads at most $2^{2^f-1} - 1$ bits. If $x \in L$ then ψ_x is satisfiable and so there exists a proof that makes the verifier accept with probability one. Observe that the probability of acceptance of the verifier is equal to the fraction of constraints that are satisfied by the proof; if $x \notin L$, then no proof can make the verifier accept with probability larger than s.

This establish the first part of the theorem.

To establish the second part, we use algorithm `OverSimp`. Let `OverSimp`$(\varphi_x) = (\psi'_x, S')$. We have the following properties:

1. No constraint C of ψ_x has more than 2^f satisfying assignments (`OverSimp` never makes the satisfying table of a constraint larger);

2. Any constraint in ψ_x has a $\mathbf{1}$-independent satisfying table;

3. If φ_x is satisfiable then ψ_x is satisfiable;

4. An assignment satisfying a fraction s of the constraints of ψ'_x and all equations of S' also satisfies at least a fraction s of the constraints of φ_x.

From the first two properties, it follows that no constraint in ψ_x has arity larger than $2^f - 1$. If $s < 2^f / 2^{2^f-1}$ then we are able to satisfy more than a fraction s of the constraints of φ_x whenever $x \in L$. This implies that we have a polynomial-time algorithm for L. □

Observe that, in particular, we have

$$\mathbf{na}\mathsf{FPCP}_{1,3/4-\epsilon}[\log, \log 3] \subseteq \mathsf{P}\ \forall \varepsilon > 0$$

and

$$\mathbf{na}\mathsf{FPCP}_{1,1/2-\epsilon}[\log, 2] \subseteq \mathsf{P}\ \forall \varepsilon > 0\ .$$

The first inclusion means that using a Proof System with perfect completeness, non-adaptiveness, and $\log 3$ free bits, the stronger hardness result that can be shown for Vertex Cover is $13/12 - \varepsilon$ for any ε (weaker than Håstad's result that uses almost-perfect completeness [Hås97]). Using 2 free bits, the stronger result is $7/6 - \varepsilon$, which equals Håstad's result. With $\log 5$ free bits or more, we can only do worse. The moral of this result is that the only way to improve the $7/6 - \varepsilon$ hardness result for Vertex Cover is to use non-perfect completeness and/or adaptiveness.

An interesting open question is to find impossibility result for non-adaptive verifiers. Indeed, we suspect that, with a bounded number of adaptive free bits and perfect completeness, it is impossible to achieve an arbitrary good soundness. In particular, it may be the case that $\mathsf{FPCP}_{1,s}[\log, f] \subseteq \mathsf{P}$ when $s < 2^{2^{2^f}}$.

8 Gadget Construction

8.1 Methodology

All the new gadgets used in this paper are computer-constructed using the methodology of [TSSW96]. We refer the reader to the paper for details about the method (a full version is available from the authors. See also the presentation in [Tre97].) In short, [TSSW96] show that the problem of finding the best possible gadget reducing a function to 2SAT (in general, the target can be any *hereditary* family, that is, any family that is closed under substitutions) can be reduced to a linear program.

This method has been implemented by Greg Sorkin. His implementation consists in an APL2 program that given the description of the source family and of the target family generates the appropriate linear program and then solves it using OSL (the IBM Optimization Subroutine Library, a commercial package for mathematical programming.) Almost all the gadgets reported in [TSSW96] have been found with Sorkin's program. In order to deal with the computing environment of the University of Geneva we had to develop a different implementation. Our implementation is much more rudimentary but only requires public domain resources.

In our implementation, the LP is generated by a C program. The program is specialized to the case where the target family is 2SAT and the source function is 3-ary. The function that we want to reduce to 2SAT is specified in a header file. To find gadgets reducing different functions to 2SAT it is thus necessary to edit the header and then recompile the program. To find the gadget reducing 4SAt to 2SAT we also had to slightly alter the program itself. Another special case arose for gadgets reducing functions in $3CSP^4$ to 2SAT. There were 34 cases to be considered (see Lemma 48), so we modified the program in order to have it generate itself all the cases, solve all of them, and then report the 34 solutions.

Finally, we remark that a straightforward application of the technique of [TSSW96] would lead to exceedingly too big linear programs in some cases. For example, in order to find the best gadget reducing 4SAT to 2SAT we would have to solve a linear program with 2^{63} constraints. To deal with such cases, we used a method already employed in [TSSW96] that reduces the size of the linear program at the cost of possibly producing non-optimal gadgets. Alternatively, it is possible to reduce the size of the linear programs at the cost of possibly producing unfeasible gadgets of super-optimal cost. When the two approaches produce gadgets of the same cost, then we have a guarantee of optimality. The reader is again referred to [TSSW96] for more details.

The generated LP is solved using a public domain LP solver, lp_solve version 2, written by Michel Berkelaar (with some additions by Jeroen Dirks), available at the URL ftp://ftp.zib.de/pub/mathprog/lp-berkelaar/lp-solve/ Once the LP is solved, the solution is reported in LaTeX. The descriptions of the gadgets appearing in Section 8.2, except the gadget reducing 4SAT to 2SAT, are unedited outputs of our program.

We used a SUN SPARCstation4 running Solaris, with 64Mb of memory. Our program and the lp_solve libraries have been compiled with GNU's gcc. The larger linear programs that we solved had ~ 200 variables and ~ 1000 constraints. It took roughly two minutes to solve them. All other LPs had only 100 to 200 constraints and less than 100 variables, and they were solved in a few seconds.

8.2 Constructions

Lemma 42 *For any $f \in$ 2CSP that is not identically 0 nor identically 1, there is a 2-gadget reducing f to 2SAT.*

Lemma 43 *If $f \in$ 3CSP has a $(2, \mathbf{1})$-dependent satisfying table, then there is a 2-gadget reducing f to 2SAT.*

Definition 44 *For two functions $f, g : \{0,1\}^k \to \{0,1\}$, we say that f is derivable from g if $f(x_1, \ldots, x_k) = g(l_{\pi[1]}, \ldots, l_{\pi[k]})$ where π is a permutation of $\{1, \ldots, k\}$ and l_i either x_i or $\bar{x}_i$.*

For example, if $f(x_1, x_2, x_3) = g(x_3, 1 - x_1, x_2)$, then f is derivable from g.

Lemma 45 *If f is derivable from g and there is an α-gadget reducing g to 2SAT, then there is also an α-gadget reducing f to 2SAT.*

Lemma 46 *For any $f \in$ 3CSP^3, there is a 5.5-gadget reducing f to 2SAT, and it is optimal.*

Proof: By Lemma 43 it is sufficient to only consider functions with a $(2, \mathbf{1})$-independent satisfying table. By Lemma 45, we can consider only functions such that each column of the satisfying table has more zeros than ones (all other functions can be then obtained using complementation). We can also assume that columns are in lexicographic oreder. It is thus sufficient to only consider the function 1-in-3 with the following satisfying table:

x_1	x_2	x_2
0	0	1
0	1	0
1	0	0

There is a 5.5-gadget reducing 1-in-3 to 2SAT. The gadget has the following clauses:

$$\begin{array}{llll} \text{weight } 0.5: & (x_2 \vee x_3), & (x_1 \vee x_2), & (x_1 \vee x_3) \\ \text{weight } 1.5: & (\neg x_1 \vee \neg x_3), & (\neg x_1 \vee \neg x_2), & (\neg x_2 \vee \neg x_3) \end{array}$$

$\square$

Columns chosen	cost of the gadget
$\{1,4,6\},\{1,4,7\},\{1,5,6\},$ $\{1,5,7\},\{2,4,5\},\{2,4,7\},$ $\{2,5,6\},\{2,6,7\},\{3,4,5\},$ $\{3,4,6\}\{3,5,7\},\{3,6,7\}$	2
$\{4,5,6\},\{4,5,7\},\{4,6,7\},$ $\{5,6,7\}$	3
$\{1,2,4\},\{1,2,5\},\{1,2,6\},$ $\{1,2,7\},\{1,3,4\},\{1,3,5\},$ $\{1,3,6\},\{1,3,7\},\{2,3,4\},$ {2,3,5} ,{2,3,6}, {2,3,7}	4.5
$\{1,4,5\},\{1,6,7\},\{2,4,6\},$ $\{2,5,7\},\{3,4,7\},\{3,5,6\}$	5.5

Table 4: Gadgets reducing non-linear functions in $3\mathrm{CSP}^4$ to 2SAT.

Lemma 47 ([BGS98],[TSSW96]) *For any 3-ary linear function f, there is an 11-gadget reducing f to* 2SAT*, and it is optimal.*

Lemma 48 *For any $f \in 3\mathrm{CSP}^4$ that is not linear, there is a 5.5-gadget reducing f to* 2SAT*. It is optimal for some of these functions.*

PROOF: As usual, we restrict ourselves to functions with a $(2,\mathbf{1})$-independent satisfying table. It is easy to see that we can also restrict to functions f such that $f(0,0,0) = 1$. (Any other function is derivable from a function that is satisfied by $(0,0,0)$.) To sum up, we have to consider all the functions whose satisfying table is obtained by taking three out of the seven columns of the following matrix, except the function obtained by taking the first three columns (it is linear!).

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{matrix}$$

There are $\binom{7}{3} - 1 = 34$ cases to be considered. For space reasons we do not report all the gadgets. In Table 4 we report the costs of the optimal gadgets for all the cases. The description of the 34 gadgets are available on request from the author.

$\square$

Lemma 49 *For any $f \in 3\mathrm{CSP}^5$, there is a* 8.25*-gadget reducing f to* 2SAT*. This is optimal for some of these functions.*

PROOF: Up to permutation of variables and complementation, there are only three possible constraints. We show below their satisfying tables

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
1	0	0
1	1	1

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
1	0	0
0	1	1

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
1	0	1
1	1	0

We have a 8.25-gadget reducing the first one to 2SAT:

$$\begin{array}{llll}
\text{weight } 0.25: & (\neg x_3 \vee \neg x_5), & (x_1 \vee \neg x_5), & (\neg x_1 \vee x_5), \\
& (\neg x_1 \vee x_4), & (x_3 \vee x_5) & \\
\text{weight } 0.5: & (x_3 \vee \neg x_7), & (\neg x_2 \vee x_4), & (\neg x_2 \vee x_6), \\
& (\neg x_2 \vee \neg x_5), & (\neg x_1 \vee \neg x_6), & (x_2 \vee x_7), \\
& (\neg x_3 \vee \neg x_6), & (\neg x_1 \vee \neg x_7) & \\
\text{weight } 0.75: & (x_3 \vee \neg x_4), & (\neg x_3 \vee x_4) & \\
\text{weight } 1: & (x_2 \vee \neg x_4) & & \\
\text{weight } 1.25: & (x_1 \vee \neg x_4) & &
\end{array}$$

a 2-gadget for the second:

$$\text{weight } 1: \quad (\neg x_1 \vee \neg x_2), \quad (\neg x_1 \vee \neg x_3)$$

and a 3.5-gadget for the third:

$$\begin{array}{llll}
\text{weight } 0.5: & (x_2 \vee x_3), & (x_1 \vee \neg x_3), & (x_1 \vee \neg x_2), \\
& (\neg x_1 \vee x_2), & (\neg x_1 \vee x_3) & \\
\text{weight } 1.5: & (\neg x_2 \vee \neg x_3) & &
\end{array}$$

The 8.25-gadget is optimal [SS97]. □

Lemma 50 *For any $f \in$ 3CSP^6 with a 2-independent satisfying table, there is a 5.5-gadget reducing f to 2SAT. This is optimal for some of the functions.*

Proof: In this case there are again three basic cases. Indeed, we can assume with no loss of generality that the function is false in 000, then the other non-satisfying assignment can have three ones, and be 111, or two ones, and be 011 without loss of generality, or one one and be 001 without loss of generality.

x_1	x_2	x_3
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0

x_1	x_2	x_3
0	0	1
0	1	0
1	0	0
1	0	1
1	1	0
1	1	1

x_1	x_2	x_3
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

The first constraint is also known as 3-set-splitting. There is a 2.5-gadget reducing it to 2SAT.

$$\begin{array}{lllll} \text{weight } 0.5: & (\neg x_1 \vee \neg x_2), & (\neg x_1 \vee \neg x_3), & (x_1 \vee x_3), \\ & (x_1 \vee x_2), & (x_2 \vee x_3), & (\neg x_2 \vee \neg x_3) \end{array}$$

The second constraint gives rise to a constraint satisfaction problem such that, given a satisfiable instance, it is NP-hard to satisfy more than a fraction $3/4 + \varepsilon$ of the constraint [Hås97]. We have a 5.5-gadget for this constraint.

$$\begin{array}{llll} \text{weight } 0.5: & (x_1 \vee x_8), & (\neg x_1 \vee \neg x_8), & (\neg x_3 \vee \neg x_9), \\ & (\neg x_1 \vee x_9), & (\neg x_3 \vee \neg x_8), & (x_3 \vee x_9), \\ & (x_3 \vee x_8), & (x_1 \vee \neg x_9) & \\ \text{weight } 1: & (\neg x_2 \vee x_9), & (x_2 \vee \neg x_8) & \end{array}$$

Our gadget, combined with the non-approximability result of [Hås97] gives an alternative proof of the fact that MAX 2SAT is hard to approximate within $21/22 + \varepsilon$. The third constraint is just $x_1 \vee x_2$. This 5.5-gadget is optimal. □

Lemma 51 ([TSSW96]) *For any $f \in$ 3CSP^7 = 3SAT, there is a 3.5-gadget reducing f to 2SAT, and it is optimal.*

Lemma 52 *For any $f \in$ 4SAT, there is a 6-gadget reducing f to 2SAT.*

PROOF: The 6-gadget reducing $(x_1 \vee x_2 \vee x_3 \vee x_4)$ to 2SAT is

$$(x_3 \vee y_2)\ ,\ (\neg x_3 \vee \neg x_4)\ ,\ (\neg x_3 \vee \neg y_1)\ ,$$

$$(x_2 \vee \neg y_2)\ ,\ (y_5 \vee \neg y_2)\ ,\ (x_1 \vee \neg y_1)\ ,\ (x_4 \vee y_2)$$

Where y_1 and y_2 are auxiliary variables. All the constraints in the gadget have weight one. □

Acknowledgements

I thank Greg Sorkin and Madhu Sudan for having checked some of the gadget constructions of this paper. I am grateful to Pierluigi Crescenzi, Oded Goldreich, Shafi Goldwasser, and Madhu Sudan for valuable discussions.

References

[AE98] G. Andersson and L. Engebretsen. Better approximation algorithms for Set Splitting and Not-All-Equal Sat. *Information Processing Letters*, 65(6):305–311, 1998.

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in *Proc. of FOCS'92.*

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in *Proc. of FOCS'92.*

[Asa97] T. Asano. Approximation algorithms for MAX SAT: Yannakakis vs. Goemans-Williamson. In *Proceedings of the 5th IEEE Israel Symposium on Theory of Computing and Systems*, 1997.

[BF92] L. Babai and P. Frankl. *Linear Algebraic Methods in Combinatorics (2nd Preliminary version)*. Monograph in preparation, 1992.

[BGLR93] M Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 294–304, 1993. See also the errata sheet in *Proc of STOC'94*.

[BGS98] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCP's and non-approximability – towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998. Preliminary version in *Proc. of FOCS'95*.

[BS94] M. Bellare and M. Sudan. Improved non-approximability results. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 184–193, 1994.

[CFZ97] J. Chen, D. Friesen, and H. Zheng. Tight bound on Johnson's algorithm for MaxSAT. In *Proceedings of the 12th IEEE Conference on Computational Complexity*, pages 274–281, 1997.

[Cre95] N. Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *Journal of Computer and System Sciences*, 51(3):511–522, 1995.

[CT97] P. Crescenzi and L. Trevisan. MAX NP-completeness made easy. Technical Report TR97-039, Electronic Colloquium on Computational Complexity, 1997.

[FG95] U. Feige and M.X. Goemans. Approximating the value of two provers proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the 3rd IEEE Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995.

[FGL$^+$96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. Preliminary version in *Proc. of FOCS91*.

[FK94] U. Feige and J. Kilian. Two prover protocols - low error at affordable rates. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 172–183, 1994.

[GLST98] V. Guruswami, D. Lewin, M. Sudan, and L. Trevisan. A tight characterization of NP with 3 query PCPs. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, 1998.

[GW94] M.X. Goemans and D.P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994. Preliminary version in *Proc. of IPCO'93*.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995. Preliminary version in *Proc. of STOC'94*.

[Hås96a] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 627–636, 1996.

[Hås96b] J. Håstad. Testing of the long code and hardness for clique. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 11–19, 1996.

[Hås97] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[KMSV99] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1):164–191, 1999. Preliminary version in *Proc. of FOCS'94*.

[KSW97] S. Khanna, M. Sudan, and D.P. Williamson. A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 11–20, 1997.

[KZ97] B. Karloff and U. Zwick. A $(7/8 - \varepsilon)$-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, 1997.

[OHA96] T. Ono, T. Hirata, and T. Asano. Approximation algorithms for the maximum satisfiability problem. In *Proceedings of Scandinavian Workshop on Algorithmic Theory.* LNCS, Springer-Verlag, 1996.

[Pet94] E. Petrank. The hardness of approximations : Gap location. *Computational Complexity*, 4:133–157, 1994. Preliminary version in *Proc. of ISTCS'93.*

[PY91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991. Preliminary version in *Proc. of STOC'88.*

[SS97] G.B. Sorkin and M. Sudan. Personal communication, 1997.

[Tre96] L. Trevisan. Positive linear programming, parallel approximation, and PCP's. In *Proceedings of the 4th European Symposium on Algorithms*, pages 62–75. LNCS 1136, Springer-Verlag, 1996.

[Tre97] L. Trevisan. *Reductions and (Non-)Approximability.* PhD thesis, University of Rome "La Sapienza", 1997. Also available at ECCC.

[TSSW96] L. Trevisan, G.B. Sorkin, M. Sudan, and D.P. Williamson. Gadgets, approximation, and linear programming. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 617–626, 1996.

[Yan94] M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17:475–502, 1994. Preliminary version in *Proc. of SODA'92.*

[Zwi98] U. Zwick. Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.