Parallel Approximation Algorithms by Positive Linear Programming[*]

LUCA TREVISAN
Università di Roma *La Sapienza*
Dipartimento di Scienze dell'Informazione
Via Salaria 113, I-00198 Roma, Italy. Email: trevisan@dsi.uniroma1.it.
Url: http://www.dsi.uniroma1.it/~trevisan

September 30, 1996

Abstract

Several sequential approximation algorithms for combinatorial optimization problems are based on the following paradigm: solve a linear or semidefinite programming relaxation, then use randomized rounding to convert fractional solutions of the relaxation into integer solutions for the original combinatorial problem. We demonstrate that such a paradigm can also yield *parallel* approximation algorithms by showing how to convert certain linear programming relaxations into essentially equivalent *positive linear programming* [LN93] relaxations that can be near-optimally solved in NC. Building on this technique, and finding some new linear programming relaxations, we develop improved parallel approximation algorithms for MAX SAT, MAX DIRECTED CUT, and MAX kCSP. The MAX SAT algorithm essentially matches the best approximation obtainable with sequential algorithms and has a fast sequential version. The MAX kCSP algorithm improves even over previous sequential algorithms. We also show a connection between probabilistic proof checking and a restricted version of MAX kCSP. This implies that our approximation algorithm for MAX kCSP can be used to prove inclusion in P for certain PCP classes.

1 Introduction

Several approximation algorithms for combinatorial optimization problems are based on the following paradigm: find a mathematical programming (usually, linear or semidefinite programming) relaxation of the problem, that can be solved in polynomial time, and then prove that any feasible "fractional" solution for the relaxation can be *rounded* to yield a feasible solution for the combinatorial problem whose measure is within a multiplicative factor r from the measure of the original fractional solution. Thus, if we round an optimum solution for the relaxation we will get an r-approximate[1] solution for the combinatorial problem. A well known early example of this technique is Hochbaum's approximation algorithm for MIN WEIGHTED VERTEX COVER [Hoc82],

[*]Research partially supported by the HCM SCOOP project of the European Union. Part of this work was done while the author was visiting the Departament de Llenguatges i Sistemes Informàtics of the Universitat Politècnica de Catalunya

[1]We say that a solution is r-approximate ($r < 1$) for an instance of a combinatorial problem if its measure is within a multiplicative factor of r from the optimum. See e.g. [BC93] for formal definitions about optimization problems and approximation algorithms.

where a simple *deterministic* rounding scheme is used. However, *randomized* rounding schemes (first introduced by Raghavan and Thompson [RT87]) are in general more efficient and are usually derandomizable. A very nice application of randomly rounding a linear programming (LP) relaxation is Goemans and Williamson's 3/4-approximate algorithm for MAX SAT [GW94] that achieves the same performance of a previous algorithm by Yannakakis [Yan94] but that is much easier to describe and analyse. Outstanding approximation results have been obtained in the last two years by randomly rounding *semidefinite relaxations* of combinatorial problems. Starting with the celebrated results by Goemans and Williamson [GW95], who showed that MAX CUT and MAX 2SAT are .878-approximable with this technique, an increasing number of results have been obtained using semidefinite programming, including better results for MAX 2SAT [FG95] and new results for graph coloring [KMS94], and the "betweeness" problem [CS95]. Shmoys' recent survey on approximation algorithms [Shm95] contains other applications of linear and semidefinite programming.

Unfortunately, such powerful techniques do not seem to be useful to develop *efficient parallel* approximation algorithms, the main reason being that both linear and semidefinite programming not only are P-hard problems, but it is even P-hard to approximate them [Ser91].

However, there exists a restricted version of linear programming (called Positive Linear Programming, PLP for short) that can be near-optimally solved using an NC algorithm[2] by Luby and Nisan [LN93]. Luby and Nisan observed that their algorithm could be used to approximate MIN SET COVER in NC within a factor $(1 + o(1)) \ln \Delta$, were Δ is the maximum cardinality of any set in the family, but, to the best of our knowledge, PLP has never been used to give relaxations of combinatorial problems in combination with random rounding schemes. Indeed, PLP is seemingly a very restricted version of linear programming, capturing *packing* and *covering* problems but nearly nothing else. Contrary to this intuition, we show that some good linear programming relaxations can be "translated" in a PLP form, thus yielding NC approximation algorithms.

Reductions to PLP. As a preparatory technical step, we prove that given a linear program that satisfies a certain set of properties, we can find near optimal solutions for it in NC by transforming it into an essentially equivalent positive linear program and then using Luby and Nisan's algorithm.

The MAX SAT problem. We then consider the MAX SAT problem, and its linear programming relaxation due to Goemans and Williamson [GW94]. In the MAX SAT problem we are given a set $\{C_1, \ldots, C_m\}$ of *disjunctive clauses* over variables $\{x_1, \ldots, x_n\}$ and non-negative weights $w_1, \ldots, w_m$ for the clauses. We seek for an assigment of truth value to the variables $\{x_1, \ldots, x_n\}$ that maximizes the sum of the weights of the satisfied clauses. We prove that Goemans and Williamson's LP relaxation can be near-optimally solved in NC using our reduction to PLP. We also show how to introduce a minor change in Goemans and Williamson's arguments and thus prove the 3/4-approximation guarantee assuming only 5-wise independence. As a consequence, we have that the MAX SAT problem is $(3/4 - o(1))$-approximable in NC. Since PLP can be solved sequentially in quasi-linear time, our translation also implies a $(3/4 - o(1))$-approximate sequential algorithm that runs in $\tilde{O}(m)$ time, where m is the number of clauses. Recall that the best approximation that is currently achievable for MAX SAT using sequential algorithms [GW94, GW95, FG95] is roughly .76, and to obtain such approximation it is necessary to solve semidefinite programs; Yannakakis' 3/4-approximate algorithm [Yan94] requires to solve max flow problems. Our algorithm achieves similar approximation with a remarkably faster running time. The best previous NC approximation for this problem was 1/2, due to Bongiovanni et al. [BCA91] and, independentely, to Hunt et al. [HMR$^+$93]

[2]Recall that, informally, an NC algorithm is an algorithm that runs on a parallel computer in poly-logarithmic time using a polynomial number of processors (see e.g. [BC93] for formal definitions).

using techniques of Luby [Lub86]. More generally, [BCA91, HMR$^+$93] developed NC approximation algorithms for all the problems in the Max SNP [PY91] class. In particular, their algorithm for Max Sat requires a quadratic number of processors. More recently, Haglin [Hag92] presented an NC 1/2-approximate algorithm for Max 2Sat that uses a linear number of processors, and Serna and Xhafa [SX95] showed that a linear number of processors is sufficient to 1/2-approximate the general Max Sat problem.

The Max kCSP **problem.** For any $k \geq 1$, the Max kCSP problem is the variation of the Max Sat problem where any clause (also called *constraint*) is allowed to be an arbitrary boolean function over k variables. This problem is somehow implicit in [PY91] and has been defined in [KMSV94] (it has also been called "Max k Function Sat" in [AKK95] and "Max k-GSAT" in [Pap94]). The interest in this problem has been mainly related to the fact that it can express any Max SNP problem. Variations of this problem have also been studied due to their applications to the field of Artificial Intelligence (see [LW96] and the references therein). We show that in order to r-approximate this problem it is sufficient to r-approximate its restricted version Max k Conj Sat, where each clause is a *conjunction* of literals. For both problems, only 2^{-k}-approximate (see e.g. [KMSV94]) sequential algorithms are known. The same approximation factor is easily achievable in NC using ideas from [Lub86, BCA91, HMR$^+$93]. We define a linear programming relaxation of the Max k Conj Sat problem and we show that a proper random rounding scheme can be used to yield a 2^{1-k} approximation. The LP relaxation can be near-optimally solved in NC, and so we obtain an NC $(2^{1-k} - o(1))$-approximate algorithm. Using our reduction, both algorithms extend to the general Max kCSP problem.

The Max Directed Cut **problem.** In the Max Directed Cut problem, we are given a directed graph $G = (V, E)$ and non-negative weights $\{w_{(u,v)}\}_{(u,v)\in E}$ and we search for a partition (V_1, V_2) of the vertices that maximizes the sum of the weights of the edges whose first endpoint is in V_1 and whose second endpoint is in V_2. Since it is well known that Max Directed Cut can be seen as a special case of Max 2 Conj Sat, our result for Max kCSP implies that we have an NC $(1/2 - o(1))$-approximate algorithm for Max Directed Cut. Previous results [Lub86] implied that this problem was 1/4-approximable in NC. Sequential approximation algorithms for this problem are, however, far better: Feige and Goemans [FG95] (improving a previous .796-approximate algorithm by Goemans and Williamson [GW95]) recently gave a .855-approximate algorithm using semidefinite programming.

Relation to PCP's. Finally, we show that an approximation algorithm for Max k Conj Sat can be used to *approximate the probability of acceptance of probabilistic verifiers* that *adaptively* read k bits. On the one hand this reduction yields the NP-hardness of approximating Max k Conj Sat (and thus Max kCSP) within a factor $2^{-0.09k}$, on the other hand, together with our improved approximation algorithm, it can be used to show that certain classes of languages defined in terms of probabilistic proof checking are contained in P. Such results strengthen previous ones by Bellare, Goldreich and Sudan [BGS95].

Related and independent results After completing this research, we learnt that Cristina Bazgan independentely used linear programming and random rounding to approximate Max k Conj Sat within a factor $e/(e^{1/k} + 1)^k$ [Baz96]. Such approximation is better than 2^{-k}, but is worse than 2^{1-k}. Lau and Watanabe [LW96] used linear programming and random rounding to approximate the Max 2CSP problem over non-boolean domains: both their relaxations and their rounding schemes are different from ours. Motivated by the results of the present paper, a .3674-approximate algorithm for MAX 3 CONJ SAT has been recently developed in [TSSW96]. Seemingly, such an algorithm does not extend to the general MAX k CONJ SAT problem, and, since it involves semidefinite programming, it cannot be easily parallelized.

Organization of the paper

In Section 2 we review some known results; in Section 3 we introduce a general technique to reduce a certain class of linear programming problems to PLP. Section 4 is devoted to the $(3/4 - o(1))$-approximate algorithm for MAX SAT. The algorithms for MAX kCSP and the applications to the MAX DIRECTED CUT problem are discussed in Sections 5, while the applications to probabilistically checkable proofs are presented in Section 6. Some open questions are discussed in Section 7.

2 Preliminaries

In what follows we will denote by $[n]$ the set $\{1, \ldots, n\}$. Boldface letters will be used to denote vectors, e.g. $\mathbf{u} = (u_1, \ldots, u_m)$. Sometimes we will use $\mathbf{1}$ to denote a vector all whose entries are equal to 1. We also use the notations $\tilde{O}(f) \stackrel{\text{def}}{=} O(f(\log f)^{O(1)})$ and $\text{poly}(f) \stackrel{\text{def}}{=} O(f^{O(1)})$. Given an instance I of MAX SAT we let $\mathsf{opt}_{MS}(I)$ be the measure of an optimum solution for I.

Definition 1 (Positive Linear Programming [LN93]) *A maximization linear program is said to be an instance of* positive linear programming *(PLP for short) if it is written as*

$$\begin{array}{ll} \max & \mathbf{c}^T\mathbf{x} \\ s.t. & \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

where all the entries of A, $\mathbf{b}$ and $\mathbf{c}$ are non-negative.

Maximization positive linear programs are also called *fractional packing* problems. Luby and Nisan developed a very efficient algorithm for approximating positive linear programming problems.

Theorem 2 ([LN93]) *There exists a parallel algorithm that given in input a maximization instance P of PLP and a rational $\epsilon > 0$ returns a feasible solution for P whose cost is at least $(1 - \epsilon)$ times the optimum. Furthermore, the algorithm runs in time polynomial in $1/\epsilon$ and $\log N$ using $O(N)$ processors, where N is the number of non-zero entries in P.*

The following result is useful to derandomize parallel algorithms where randomization is only needed to generate random variables with limited independence.

Theorem 3 (see e.g. [LW95, Section 16]) *A pairwise independent distribution of n random variables of size $O(n)$ is explicitely constructable in* NC*. For any $k > 2$, a k-wise independent distribution of n random variables of size $O(n^k)$ is explicitely constructable in* NC.

3 On LP Problems that Are Reducible to PLP

In this section we will show that a certain class of linear programs can be approximately solved in NC by reducing them to instances of PLP and then using Luby and Nisan's algorithm.

Definition 4 ((γ, k)-form) *Let $\gamma > 0$, $k \in \mathbf{Z}^+$ be constants; we say that a linear program is in* (γ, k)-form *if it is written as*

$$\begin{array}{ll} \max & \mathbf{c}^T\mathbf{z} \\ s.t. & \\ & A^{(1)}\mathbf{z} + A^{(2)}\mathbf{x} - A^{(3)}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \\ & \mathbf{0} \leq \mathbf{z} \leq \mathbf{1} \end{array}$$

where $\mathbf{z} = (z_1, \ldots, z_m)$, $\mathbf{x} = (x_1, \ldots, x_n)$ and the following properties hold:

1. *All the entries of* $\mathbf{c}$ *and* $\mathbf{b}$ *are non negative.*
2. *All the entries of $A^{(1)}, A^{(2)}, A^{(3)}$ are in $\{0, 1\}$.*
3. *Any row of $A^{(1)}$ has at most one non-zero entry.*
4. *For any $j \in [m]$, $\sum_{i=1}^{n} \sum_{h:a^{(1)}_{h,j} \neq 0} (a^{(2)}_{h,i} + a^{(3)}_{h,i}) \leq k$.*
5. *The solution $x_1 = \ldots = x_n = z_1 = \ldots = z_m = \gamma$ is feasible.*

The aboved definition is admittedly quite artificial. It tries to capture the kind of nice properties that are shared by LP relaxation of certain combinatorial problems. In this paper we deal with satisfiability problems where there are clauses defined over boolean variables, and we want to find a setting of the variables that satisfies the maximum number of clauses. A natural way of formulating such problems is to use one LP variable for any clause and one for any boolean variable; this motivates the fact that we distinguish between $\mathbf{z}$ variables (for clauses) and $\mathbf{x}$ variables (for boolean variables) and we let the objective function depend only on the $\mathbf{z}$ variables. It is also natural to ask such variables to be between zero and one (one obtains the original combinatorial problem enforcing the integrality constraints $\mathbf{x} \in \{0, 1\}^n$, $\mathbf{z} \in \{0, 1\}^m$). The constraints of the first kind express the fact that, for any j, z_j can be one only if the variables occurring in the j-th clause satisfy a given property. This explains Property 3 in the definition: any constraint involves at most one $\mathbf{z}$ variable. Property 4 says that, for any j, the value of z_j is constrained by at most k of the $\mathbf{x}$ variables. This is not true for the general MAX SAT problem, but it will be easy to overcome this shortcoming.

The main result of this section is that we can efficiently find in NC near optimal solutions for LP in (γ, k) form.

Theorem 5 *For any fixed constants $\gamma > 0$ and $k \in \mathbf{Z}^+$, there exists a parallel algorithm that, given in input a linear program in (γ, k) form and a rational $\epsilon > 0$, finds a feasible $(1 - \epsilon)$-approximate solution in time* $\text{poly}(1/\epsilon, \log N)$ *using $O(N)$ processors, where N is the size of the input.*

PROOF: We will essentially describe an L-reduction [PY91] from LP in (γ, k) form to PLP. Let (LP1) be the following program in (γ, k) form.

$$
\begin{array}{lll}
\max & \mathbf{c}^T\mathbf{z} & \\
\text{s.t.} & & \\
& A_1\mathbf{z} + A_2\mathbf{x} - A_3\mathbf{x} \leq \mathbf{b} & \text{(LP1)} \\
& \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} & \\
& \mathbf{0} \leq \mathbf{z} \leq \mathbf{1} &
\end{array}
$$

Let $\mathbf{z} = (z_1, \ldots, z_m)$, $\mathbf{x} = (x_1, \ldots, x_n)$. We introduce new variables $\mathbf{y} = (y_1, \ldots, y_n)$ with the property that $y_i = 1 - x_i$. Using proper substitutions, we get the linear program (LP2), that is clearly equivalent to (LP1).

$$
\begin{array}{lll}
\max & \mathbf{c}^T\mathbf{z} & \\
\text{s.t.} & & \\
& A_1\mathbf{z} + A_2\mathbf{x} + A_3\mathbf{y} \leq \mathbf{b} + A_3\mathbf{1} & \\
& \mathbf{x} + \mathbf{y} = \mathbf{1} & \text{(LP2)} \\
& \mathbf{x} \geq \mathbf{0} & \\
& \mathbf{y} \geq \mathbf{0} & \\
& \mathbf{0} \leq \mathbf{z} \leq \mathbf{1} &
\end{array}
$$

By using such substitutions we have obtained a formulation where all the entries are non-negative. Yet, we have introduced equality constraints that are not admitted in PLP. We deal with this new problem by relaxing the equality constraints to the inequalities $\mathbf{x} + \mathbf{y} \leq \mathbf{1}$ and then modifying the objective function in such a way that it is never "convenient" to satisfy strictly such inequalities. More formally, we introduce a third (and last) formulation, that we call (LP3). For any $i \in [n]$ we let

$$
occ_i = \sum_{j=1}^{m} c_j \sum_{h : a^{(1)}_{h,j} \neq 0} a^{(3)}_{(h,i)}
$$

be the weighted number of constraints where variable x_i occurs (with a negative sign) together with variable z_j, let also $\mathbf{occ} = (occ_1, \ldots, occ_n)$. (LP3) is the following positive linear program.

$$
\begin{array}{lll}
\max & \mathbf{c}^T\mathbf{z} + \mathbf{occ}^T(\mathbf{x} + \mathbf{y}) & \\
\text{s.t.} & & \\
& A_1\mathbf{z} + A_2\mathbf{x} + A_3\mathbf{y} \leq \mathbf{b} + A_3\mathbf{1} & \\
& \mathbf{x} + \mathbf{y} \leq \mathbf{1} & \text{(LP3)} \\
& \mathbf{x} \geq \mathbf{0} & \\
& \mathbf{y} \geq \mathbf{0} & \\
& \mathbf{0} \leq \mathbf{z} \leq \mathbf{1} &
\end{array}
$$

In the following we will prove that a good approximate solution for (LP3) is (possibly, after small changes) also a good approximate solution for (LP1). Let Z^*_{LP1} be the optimum value of (LP1), Z^*_{LP3} the optimum value of (LP3).

We first show that the optimum of (LP3) is not much larger than the optimum of (LP1).

Claim 1 $Z^*_{\text{LP3}} \leq Z^*_{\text{LP1}}(1 + k)/\gamma$.

Proof: [Of Claim 1] Property 5 in the definition of (γ, k) form ensures that $Z^*_{\text{LP1}} \geq \sum_j \gamma c_j$. On the other hand, the constraints on $\mathbf{z}$, $\mathbf{x}$ and $\mathbf{y}$ imply that $Z^*_{\text{LP3}} \leq \sum_j c_j + \sum_i occ_i$. Let us now extimate $\sum_i occ_i$; from the definition of **occ** and from Property 4 in Definition 4 we have that

$$\sum_i occ_i = \sum_i \sum_j c_j \sum_{h: a^{(1)}_{h,j} \neq 0} a^{(3)}_{h,i} = \sum_j c_j \sum_i \sum_{h: a^{(1)}_{h,j} \neq 0} a^{(3)}_{h,i} \leq k \sum_j c_j \ .$$

Thus, $Z^*_{\text{LP3}} \leq (k+1) \sum_j c_j \leq (k+1)/\gamma Z^*_{\text{LP1}}$. □

The following two claims will imply that a feasible solution for (LP3) can be converted into a feasible solution for (LP1) without increasing the *additive error.*

Claim 2 *If* $(\mathbf{z}, \mathbf{x})$ *is a feasible solution for* (LP1) *of cost* C*, then* $(\mathbf{z}, \mathbf{x}, \mathbf{1} - \mathbf{x})$ *is a feasible solution for* (LP3) *of cost* $C + \sum_i occ_i$*.*

Claim 3 *Given a feasible solution* $(\mathbf{z}, \mathbf{x}, \mathbf{y})$ *for* (LP3) *of cost* $C + \sum_i occ_i$ *a feasible solution* $(\mathbf{z}', \mathbf{x}')$ *for* (LP1) *of cost at least* C *exists and can be computed in NC using* $O(N)$ *processors in* $\text{poly}(\log N)$ *time.*

Proof: [Of Claim 3] Let $(\mathbf{z}', \mathbf{x}')$ be the defined as follows

$$\mathbf{x}' = \mathbf{x} \qquad z'_j = \min\{z_j, \min_{h: a^{(1)}_{h,j} \neq 0} \{b_h - \sum_i a^{(2)}_{h,i} x_i + \sum_i a^{(3)}_{h,i} x_i\}\} \ .$$

It is easy to see that $(\mathbf{z}', \mathbf{x}')$ is feasible for (LP1) and that it can be computed with the time and processor bound required in the claim. It will take some more efforts to bound its cost. Let J be the set of indices j such that $z'_j \neq z_j$. For any such $j \in J$, let h_j be one of the indices (say, the smallest one) such that

$$a^{(1)}_{h_j,j} \neq 0 \qquad z'_j = b_{h_j} - \sum_i a^{(2)}_{h_j,i} x_i + \sum_i a^{(3)}_{h_j,i} x_i \ .$$

Since we have that

$$z_j \leq b_{h_j} - \sum_i a^{(2)}_{h_j,i} x_i + \sum_i a^{(3)}_{h_j,i} (1 - y_i) \ ,$$

we can derive

$$z_j - z'_j \leq \sum_i a^{(3)}_{h_j,i} (1 - (x_i + y_i)) \ .$$

Thus

$$\sum_j c_j z_j - \sum_j c_j z'_j = \sum_{j \in J} c_j (z_j - z'_j) \leq \sum_{j \in J} c_j \sum_i a^{(3)}_{h_j,i} (1 - (x_i + y_i)) \leq \sum_i occ_i (1 - (x_i + y_i))$$

Since we assumed that

$$\sum_j c_j z_j + \sum_i occ_i (x_i + y_i) = C + \sum_i occ_i$$

we have that

$$\sum_j c_j z'_j \geq C \ .$$

□

Claim 4 *Let* $(\mathbf{z}, \mathbf{x}, \mathbf{y})$ *be a* $(1 - \epsilon\gamma/(k+1))$ *approximate solution for* (LP3)*, and let* $(\mathbf{z}', \mathbf{x}')$ *be the feasible solution for* (LP1) *computed as in Claim 3. Then* $(\mathbf{z}', \mathbf{x}')$ *is a* $(1-\epsilon)$*-approximate solution for* (LP1)*.*

PROOF: [Of Claim 4] Let cost_3 (respectively, cost_1) be the cost of $(\mathbf{z}, \mathbf{x}, \mathbf{y})$ (respectively, of $(\mathbf{z}', \mathbf{x}')$). From Claim 3 it follows that $\mathsf{cost}_1 \geq \mathsf{cost}_3 - \sum_i occ_i$. From Claim 2 we have that $Z^*_{\mathrm{LP1}} \leq Z^*_{\mathrm{LP3}} - \sum_i occ_i$, and thus $Z^*_{\mathrm{LP1}} - \mathsf{cost}_1 \leq Z^*_{\mathrm{LP3}} - \mathsf{cost}_3$. It follows that.

$$\frac{\mathsf{cost}_1}{Z^*_{\mathrm{LP1}}} = 1 - \frac{Z^*_{\mathrm{LP1}} - \mathsf{cost}_1}{Z^*_{\mathrm{LP1}}} \geq 1 - \frac{Z^*_{\mathrm{LP3}} - \mathsf{cost}_3}{Z^*_{\mathrm{LP1}}} \geq 1 - \gamma/(k+1)\frac{Z^*_{\mathrm{LP3}} - \mathsf{cost}_3}{Z^*_{\mathrm{LP3}}} \leq 1 - \epsilon$$

□

The Theorem follows from Claim 4 and from Theorem 2. □

4 The Max Sat problem

Let $\{C_1, \ldots, C_m\}$ be a collection of disjunctive boolean clauses over variable set $X = \{x_1, \ldots, x_n\}$ and let $w_1, \ldots, w_m$ be the weights of such clauses. For any clause C_j let us denote by C_j^+ the set of indices of variables occuring positively in C_j and with C_j^- the set of indices of variables occuring negated, so that $C_j = \bigvee_{i \in C_j^+} x_i \vee \bigvee_{i \in C_j^-} \neg x_i$. Goemans and Williamson [GW94] consider the following linear programming relaxation of the Max Sat problem.

$$\begin{array}{lll}
\max & \sum_{j=1}^m w_j z_j & \\
\text{s.t.} & & \\
& z_j \leq \sum_{i \in C_j^+} t_i + \sum_{i \in C_j^-}(1 - t_i) & \text{for all } j \in [m] \qquad \text{(SAT)} \\
& 0 \leq z_j \leq 1 & \text{for all } j \in [m] \\
& 0 \leq t_i \leq 1 & \text{for all } i \in [n]
\end{array}$$

To see that (SAT) is indeed a relaxation of Max Sat it suffices to observe that any assigment $\tau : X \to \{\mathsf{true}, \mathsf{false}\}$ can be converted into a feasible solution for (SAT) such that, for any i, $t_i = 1$ if $\tau(x_1) = \mathsf{true}$, and $t_i = 0$ otherwise, while $z_j = 1$ if C_j is satisfied by τ and $z_j = 0$ otherwise. Note that the measure of such a solution is equal to the sum of the weights of the clauses satisfied by τ.

Remark 6 *We note that the solution* $\mathbf{x} = \mathbf{z} = 0.5 \cdot \mathbf{1}$ *is feasible: thus, if we start from a formula with at most k literals per clause, it is easy to verify that* (SAT) *is in* $(1/2, k)$ *form.*

Theorem 7 ([GW94, Theorem 5.3]) *Let* $(\mathbf{t}, \mathbf{z})$ *be a feasible solution for* (SAT)*. Consider the random assigment such that, for any i, independently,* $\mathbf{Pr}[x_i = \mathsf{true}] = \frac{1}{4} + \frac{1}{2}t_i$*. Then, for any* $j \in [m]$*,* $\mathbf{Pr}[C_j \textit{ is satisfied}] \geq \frac{3}{4}z_j$*.*

Starting with an optimum solution for (SAT), one gets a random assigment that, on the average, has a cost that is at least 3/4 of the optimum. An explicit 3/4-approximate assignment can be found deterministically using the method of conditional expectation ([AS92], see also [Yan94]).

We are now ready to prove the main result of this section.

Theorem 8 (Approximation for Max Sat)

1. *An* RNC *algorithm exists that given an instance of the weighted* Max Sat *problem and a rational* $\epsilon > 0$*, returns an assigment whose expected measure is at least* $(3/4 - \epsilon)$ *times the optimum. The algorithm runs in* $\mathrm{poly}(1/\epsilon, \log m)$ *time and uses* $O(m+n)$ *processors.*

2. *For any $\epsilon > 0$, an* NC *$(3/4-\epsilon)$-approximate algorithm for the weighted* MAX SAT *problem exists that runs in* $\mathrm{poly}(1/\epsilon, \log m)$ *time and uses* $O((m+n)n^5)$ *processors.*

3. *A sequential $(3/4 - o(1))$-approximate algorithm for the weighted* MAX SAT *problem exists that runs in* $\tilde{O}(m)$ *time.*

PROOF: Let $\phi = (C_1, \ldots, C_m)$ be any instance of MAX SAT, and let $w_1, \ldots, w_m$ be the weights of the clauses. We use the following notation: $J_4 \stackrel{\mathrm{def}}{=} \{j : C_j$ contains at most four literals $\}$, $J_5 \stackrel{\mathrm{def}}{=} [m] - J_4$, $\phi_4 \stackrel{\mathrm{def}}{=} \{C_j : j \in J_4\}$, $\phi_5 \stackrel{\mathrm{def}}{=} \{C_j : j \in J_5\}$, $w_4^{\mathrm{tot}} \stackrel{\mathrm{def}}{=} \sum_{j \in J_4} w_j$, and $w_5^{\mathrm{tot}} \stackrel{\mathrm{def}}{=} \sum_{j \in J_5} w_j$. Clearly, we have that $\mathsf{opt}_{MS}(\phi) \leq \mathsf{opt}_{MS}(\phi_4) + w_5^{\mathrm{tot}}$. Let us consider the linear programming relaxations (SAT), relative to ϕ_4 and let Z^*_{SAT}be the optimum of such a linear program. Since (SAT) is in $(1/2, 4)$ form, it follows from Theorem 5 that we can find in NC a solution $(\mathbf{z}, \mathbf{t})$ for (SAT) whose measure is at least $(1 - 4\epsilon/3) Z^*_{\mathrm{SAT}}$. Consider now the random assigment such that x_i is true with probabilty $1/4 + t_1/2$. Note that in such assigment each literal is true with probability at least $1/4$, and thus a clause with five or more literals is true with probability at least $1 - (3/4)^5 = 0.76269\ldots > 3/4$. From Theorem 7 and from the above considerations we have that the average measure of such assigment is

$$\begin{aligned}
\sum_{j=1}^{m} w_j \mathbf{Pr}[C_j \text{ is satisfied }] &\geq \frac{3}{4} \sum_{j \in J_4}^{m} w_j z_j + \sum_{j \in J_5} 0.76269 w_j \\
&\geq \frac{3}{4} \left(1 - \frac{4}{3}\epsilon\right) Z^*_{\mathrm{SAT}} + \frac{3}{4} w_5^{\mathrm{tot}} \\
&\geq \left(\frac{3}{4} - \epsilon\right) \mathsf{opt}_{MS}(\phi)\ .
\end{aligned}$$

The time bound follows from the fact that the instance of positive linear programming to be solved has size $O(m)$.

To prove Part (2) just note that the above analysis only assumed 5-wise independence. From Theorem 3 we have that a 5-wise independent probability distribution over n random variables exists of size $O(n^5)$. We can thus run in parallel $O(n^5)$ copies of the above algorithm (one for each element of the distribution) and then take the best outcome.

Finally, regarding Part (3), one can use a sequential version of Luby and Nisan's algorithm to approximate the relaxation. Since the size of the relaxation is bounded by m, it will take $O(m(\log m)^{O(1)})$ time to find a $(1-\epsilon)$-approximate solution, provided that $\epsilon = 1/(\log m)^{(O(1))}$. After applying random rounding, derandomization can be done in linear time using conditional expectation (see e.g. [Yan94]). Observe that, while doing the derandomization, we can ignore all literals occuring in a clause but the first five (this is compatible with our approximation analysis). Thus, derandomization can be done in $O(m)$ time, independent of n. □

5 The MAX kCSP problem

In this section we deal with the MAX kCSP problem. We begin by showing that, without loss of generality, we can restrict ourselves to the simpler MAX k CONJ SAT problem.

Theorem 9 (MAX kCSP **vs.** MAX k CONJ SAT) *For any $k > 1$ and for any r, $0 < r \leq 1$, if* MAX k CONJ SAT *is r-approximable, then* MAX kCSP *is r-approximable.*

PROOF: Without loss of generality, we can assume that any constraint involves the application of a boolean function of arity exactly k (indeed, any h-ary function, with $h < k$ can be seen as a k-ary function whose value is independent of the value of $k-h$ parameters). Let $C_1, \ldots, C_m$ be an instance of MAX kCSP over boolean variables $x_1, \ldots, x_n$ and assume that all weights are equal to one. For any $j \in [m]$, $C_j = f_j(x_{j_1}, \ldots, x_{j_k})$ where $j_1, \ldots, j_k \in [n]$ and $f_j : \{\mathsf{true}, \mathsf{false}\}^k \to \{\mathsf{true}, \mathsf{false}\}$. Consider now the set $S_{f_j} \subseteq \{\mathsf{true}, \mathsf{false}\}^k$ of satisfying assigment for f_j. For any such assignment $b_1, \ldots, b_k$ we can write a k-ary conjunctive clause that is satisfied only by that assigment, namely, the clause $l_{j_1} \wedge \ldots \wedge l_{j_k}$ where l_{j_i} stands for x_{j_i} if $b_i = \mathsf{true}$, and for $\neg x_{j_i}$ otherwise. In this way, we can convert any constraint C_j into $|S_{f_j}|$ new conjunctive constraints. We give weight w_j to all such constraints. Let $D_1, \ldots, D_{m'}$ be the union of such sets of new constraints: they constitute an instance of MAX k CONJ SAT over variables $x_1, \ldots, x_n$, and it should be clear that any assigment to $x_1, \ldots, x_n$ has the same measure for $C_1, \ldots, C_m$ and for $D_1, \ldots, D_{m'}$, so the instances have the same optimum, and an r-approximate solution for $D_1, \ldots, D_{m'}$ is also an an r-approximate solution for $C_1, \ldots, C_m$. □

We shall now prove that, for any $k \geq 1$, MAX k CONJ SAT is 2^{1-k}-approximable. As in the preceding section, we shall give a linear programming relaxation of the problem and a proper randomized rounding scheme. Assume we have an instance of MAX k CONJ SAT given by constraints $C_1, \ldots, C_m$, whose weights are $w_1, \ldots, w_m$ over variables $x_1, \ldots, x_n$. We denote by C_j^+ (respectively, C_j^-) the set of indices of positive (respectively, negative) literals in C_j, so that

$$C_j = \bigwedge_{i \in C_j^+} x_i \ \wedge \ \bigwedge_{i \in C_j^-} \neg x_i \ .$$

The linear programming relaxation has a variable t_i for any variable x_i of the MAX k CONJ SAT problem, plus a variable z_j for any constraint C_j. The formulation is

$$\begin{array}{lll} \max & \sum_j w_j z_j & \\ \text{s.t.} & & \\ & z_j \leq t_i & \text{for all } j \in [m], i \in C_j^+ \\ & z_j \leq 1 - t_i & \text{for all } j \in [m], i \in C_j^- \\ & 0 \leq t_i \leq 1 & \text{for all } i \in [n] \\ & 0 \leq z_j \leq 1 & \text{for all } j \end{array} \qquad \text{(CSP)}$$

The proof that (CSP) is a relaxation of MAX k CONJ SAT, is identical to the proof that (SAT) is a relaxation of MAX SAT: given an assigment τ for MAX k CONJ SAT, set $t_i = 1$ if $\tau(x_i) = \mathsf{true}$, and $t_i = 0$ otherwise; set $z_j = 1$ if τ satisfies C_j, set $z_j = 0$ otherwise. It is immediate to verify that such solution is feasible for (CSP) and that its cost is equal to the total weight of clauses satisfied by the assigment.

Theorem 10 (Random rounding for (CSP)**)** *Let* $(\mathbf{z}, \mathbf{t})$ *be a feasible solution for* (CSP)*, consider the random assigment such that*

$$\mathbf{Pr}[x_i = \mathsf{true}] = \frac{k-1}{2k} + \frac{t_i}{k} \ .$$

Then, for any $j \in [m]$,

$$\mathbf{Pr}[C_j \textit{ is satisfied}] \geq \frac{z_j}{2^{k-1}} \ .$$

Proof: Note that, according to the random assigment,

$$\mathbf{Pr}[x_i = \mathsf{false}] = 1 - \left(\frac{k-1}{2k} + \frac{t_i}{k}\right) = \frac{k-1}{2k} + \frac{1-t_i}{k} .$$

Let us assume that C_j is a h-ary constraint for some $h \leq k$.

$$\begin{aligned}
\mathbf{Pr}[C_j \text{ is satisfied }] &= \left(\prod_{i\in C_j^+} \frac{k-1}{2k} + \frac{t_i}{k}\right) \cdot \left(\prod_{i\in C_j^-} \frac{k-1}{2k} + \frac{1-t_i}{k}\right) \\
&\geq \left(\frac{k-1}{2k} + \frac{z_j}{k}\right)^h \\
&\geq \left(\frac{k-1}{2k} + \frac{z_j}{k}\right)^k \\
&\geq \frac{z_j}{2^{k-1}} .
\end{aligned}$$

Where first inequality follows from the constraints on z_j, second inequality from the fact that $h \leq k$ and the last inequality can be proved by studying the first derivative of the function

$$f(z) = \frac{(\frac{k-1}{2k} + \frac{z}{k})^k}{z}$$

and showing that, in the interval $(0, 1)$, $f(z)$ reaches its minimum for $z = 1/2$: in that point we have that $f(1/2) = 2^{1-k}$. □

Remark 11 *The above analysis is tight, as can be shown by considering an instance $C_1, \ldots, C_{2^k}$ where the clauses are all the possible size-k conjunctions over $\{x_1, \ldots, x_k\}$. Any assigment to $\{x_1, \ldots, x_k\}$ will satisfy exactly one clause (that is, the optimum is equal to 1). On the other hand, the feasible solution for* (CSP) *where all variables are equal to 1/2 has measure 2^{k-1}.*

Theorem 12 (Approximation for Max k Conj Sat**)** *For any $k \geq 1$, the weighted* Max k Conj Sat *problem is 2^{1-k}-approximable in polynomial time, and is $(2^{1-k} - o(1))$-approximable in* NC.

Proof: Regarding the first claim, in order to compute a 2^{1-k}-approximate solution it is sufficient to optimally solve (CSP) using a polynomial time algorithm for linear programming [Kha79, Kar84], then use the random rounding scheme described in Theorem 10 and finally use conditional expectation (see [AS92]) to obtain an assigment whose measure is no smaller than the average measure of such random assigment. The approximation guarantee follows from Theorem 10.

Regarding the second claim, observe that $\mathbf{t} = \mathbf{z} = .5 \cdot \mathbf{1}$ is a feasible solution, and so it can be easily checked that (CSP) is in $(1/2, k)$ form. We can then apply Theorem 5 to show that (CSP) can be approximated within a factor $1 - o(1)$ in NC. Random rounding yields a solution that on the average is $2^{1-k} - o(1)$ approximate, and we can use k-wise independent distributions to do derandomization. □

Corollary 13 (Approximation for Max kCSP**)** *For any $k > 1$, the weighted* Max kCSP *problem is 2^{1-k}-approximable in polynomial time, and $(2^{1-k} - o(1))$-approximable in* NC.

5.1 The Max Directed Cut problem

The Max Directed Cut can be seen as the restriction of Max 2 Conj Sat to instances where in each clause there is exactly one positive literal and one negative literal. Indeed, there is a natural correspondence between nodes and boolean variables, edges and constraints; and partitions and assigments. It is a folklore result that has been mentioned a few times in the litterature (e.g. in [TSSW96]) and can be traced back to the idea of giving logical formulations of combinatorial optimization problems [PY91].

Such equivalence immediately implies the existence of a parallel approximation algorithm for Max Directed Cut.

Corollary 14 (Approximation for Max Directed Cut)

1. *An* RNC *algorithm exists that given an instance of the weighted* Max Directed Cut *problem and a rational $\epsilon > 0$, returns a cut whose expected measure is at least $(1/2 - \epsilon)$ times the optimum. The algorithm runs in* $\mathrm{poly}(1/\epsilon, \log m)$ *time and uses $O(m + n)$ processors.*
2. *For any $\epsilon > 0$, an* NC *$(1/2 - \epsilon)$-approximate algorithm for the weighted* Max Directed Cut *problem exists that runs in* $\mathrm{poly}(1/\epsilon, \log m)$ *time and uses $O((m + n)n)$ processors.*

In Remark 11 we showed the tightness of our analysis of random rounding for any k. An inspection of the proof for $k = 2$ shows that the instance that we consider is

$$\{(x_1 \wedge x_2), (\neg x_1 \wedge x_2), (x_1 \wedge \neg x_2), (\neg x_1 \wedge \neg x_2)\}$$

and it doesn't correspond to a Max Directed Cut problem. This fact may lead us to hope that a better random rounding scheme is possible for the special case of linear programs arising from Max Directed Cut instances. Indeed, this is not the case. Let us first see how the LP relaxations of Max Directed Cut problems look like. Given a graph $G = (V, E)$, where we assume for simplicity $V = [|V|]$, and given weights $\{w_{(i,j)}\}_{(i,j)\in E}$ for the edges, the LP relaxation is as follows.

$$\begin{array}{lll} \max & \sum_{(i,j)\in E} w_{(i,j)} z_{(i,j)} & \\ \text{s.t.} & & \\ & z_{(i,j)} \leq t_i & \text{for all } (i,j) \in E \\ & z_{(i,j)} \leq 1 - t_j & \text{for all } (i,j) \in E \\ & 0 \leq t_i \leq 1 & \text{for all } i \in V \\ & 0 \leq z_{(i,j)} \leq 1 & \text{for all } (i,j) \in E \end{array} \qquad \text{(DI)}$$

Consider now the directed complete graph with $2n$ vertices and $2n(2n-1) = 4n^2 - 2n$ edges (assume that all weights are one). Then the optimum of the Max Directed Cut problem is clearly n^2 (the balanced partition), while the solution with all variables equal to 1/2 is feasible for (DI) and has measure $2n^2 - n$. The ratio between the two values is arbitrarily close to 1/2.

6 Relations with Proof Checking

We start by giving some definitions about probabilistically checkable proofs (we follow the notation used in [BGS95]). A *verifier* is an oracle probabilistic polynomial-time Turing machine V. During its computation, V tosses random coins, reads its input and has oracle access to a string π called *proof.* Let x be an input and π be a proof. We denote by $\mathbf{Acc}[V^\pi(x)]$ the probability over its random tosses that V accepts x using π as an oracle. We also denote by $\mathbf{Acc}[V(x)]$ the maximum of $\mathbf{Acc}[V^\pi(x)]$ over all proofs π. The efficiency of the verifier is determined by several parameters.

Definition 15 (PCP parameters) *Let L be a language, and let V be a verifier for L. Then we say that*

- V uses $r(n)$ random bits *(where $r : \mathbf{Z}^+ \to \mathbf{Z}^+$ is an integer function) if for any input x and for any proof π, V tosses at most $r(|x|)$ random coins;*
- V has query complexity q *(where q is an integer) if for any input x, any random string R, and any proof π, V reads at most q bits from π;*
- V has soundness s *(where $s \in [0,1]$ is a real) if, for any $x \notin L$, $\mathbf{Acc}[V(x)] \leq s$;*
- V has completeness c *(where $c \in [0,1]$ is a real) if, for any $x \in L$, $\mathbf{Acc}[V(x)] \geq c$.*

Remark 16 *Note that a verifier that has query complexity q can read its q bits* adaptively, *that is, the i-th access to the proof may depend on the outcomes of the previous $i-1$ accesses.*

Definition 17 (PCP classes) *Let L be a language, let $0 < s < c \leq 1$ be any constants, q be a positive integer and $r : \mathbf{Z}^+ \to \mathbf{Z}^+$, then we say that $L \in \mathrm{PCP}_{c,s}[r,q]$ if a verifier V exists for L that uses $O(r(n))$ random bits, has query complexity q, soundness s and completeness c.*

Several recent results about the hardness of approximation of combinatorial optimization problems (including Max Sat [BGS95] and Max Directed Cut [BGS95, TSSW96]) have been proved using the fact, proved in [BGS95], that NP = $\mathrm{PCP}_{1,s}[\log, 3]$ for any $s > 0.85$. The verifier developed to prove such result is adaptive. Using less than 3 queries or having a soundness smaller than 0.85 would immediately imply improved non-approximability results. Due to such consideration, it seems interesting to consider what kind of combinations of parameters may be sufficient to characterize NP, and which one are too weak (unless P = NP). The next result implies that one can prove inclusion of PCP classes into P by simply developing approximation algoritms for Max k Conj Sat.

Theorem 18 (Max k Conj Sat vs PCP) *If* Max k Conj Sat *is r-approximable for some $r \leq 1$, then $\mathrm{PCP}_{c,s}[\log, k] \subseteq \mathrm{P}$ for any $c/s > 1/r$.*

Proof: Let $L \in \mathrm{PCP}_{c,s}[\log, k]$ and let V be a verifier witnessing it. Let x be any string of size n. Let l be the maximum lenght of a proof for x (note that l is polynomial in n). On input x, V tosses $\alpha \log n$ random bits (for some constant α) and adaptively reads q bits of the proof. Let $\mathsf{r} = 2^{\alpha \log n}$ be the total (polynomial) number of possible sequences of coin tosses of V. For any of such strings $R \in \{0,1\}^{\alpha \log n}$, let us consider the behaviour of V with input x and random tosses R: clearly it is entirely determined by the outcomes of its queries, and can be represented as a binary tree of height k. Indeed, any internal vertex corresponds to a query, branching corresponds to the two possible outcomes of the query and leaves correspond to final accepting or rejecting states. Any root-leaf path corresponds to a set of queries (positions of the proof) $q_1, \ldots, q_k$ and to answers $a_1, \ldots, a_k$, and it is followed by the verifier if and only if $\pi[q_i] = a_i$ for any $i \in [k]$. Let us now associate a boolean variable x_i to any position $\pi[i]$ of the proof, for any $i \in [l]$. There is an immediate correspondence between proofs (i.e. strings in $\{0,1\}^l$) and assigments for such variables (i.e. strings in $\{\mathsf{false}, \mathsf{true}\}^l$), say by identifying 1 with true, and 0 with false. According to this intuition, we encode root-leaf paths leading to acceptance as conjunctive clauses in the expected way: a path involving queries $q_1, \ldots, q_k$ and answers $a_1, \ldots, a_k$ is converted into the conjunctive clause $l_{q_1} \wedge \ldots, l_{q_k}$ where l_{q_i} stands for x_{q_i} if $a_i = 1$, and stands for $\neg x_{q_i}$ otherwise. It should be clear that those clauses are inconsistent one with the other, and that an assigments satisfies one of those clauses iff the corresponding proof makes the verifier accept when it uses random string R. We can repeat such construction for any R, and it leads to an instance of Max k Conj Sat such that i clauses are simultaneously satisfiable iff a proof exists that makes V accept x with probability i/r. It follows that the optimum of the resulting Max k Conj Sat instance is at least $c\mathsf{r}$ if $x \in L$, and is at most $s\mathsf{r}$ otherwise. Any approximation better than s/c is sufficient to distinguish between the two cases. □

A first consequence of Theorem 18 is that Max k Conj Sat is hard to approximate even within very small factors.

Theorem 19 (Hardness of Max k Conj Sat**)** *For any* $k \geq 11$, *if* Max k Conj Sat *is* $2^{-\lfloor k/11 \rfloor}$*-approximable, then* P = NP.

Proof: Bellare Goldreich and Sudan [BGS95] prove that an $s < .5$ exists such that NP = $\mathrm{PCP}_{1,s}[\log, 11]$. Then, $\lfloor k/11 \rfloor$ independent repetitions of their protocol yield NP = $\mathrm{PCP}_{1,s'}[\log, k]$, where $s' < 2^{-\lfloor k/11 \rfloor}$: applying Theorem 18, the claim follows. □

The following result can be obtained by combining Theorems 12 and 18

Theorem 20 (Weak PCP classes) $\mathrm{PCP}_{c,s}[\log, q] \subseteq \mathrm{P}$ *for any* $c/s > 2^{q-1}$. *In particular,* $\mathrm{PCP}_{1,0.249}[\log, 3] \subseteq \mathrm{P}$.

The above theorem improves over previous results by Bellare, Goldreich and Sudan [BGS95], stating that $\mathrm{PCP}_{c,s}[\log, q] \subseteq \mathrm{P}$ for any $c/s > 2^q$ and $\mathrm{PCP}_{1,0.18}[\log, 3] \subseteq \mathrm{P}$, respectively.

7 Conclusions

Following the work of [Rag88, GW94] we considered linear programming relaxations of combinatorial optimization problems and used *random rounding* to obtain feasible solutions from the fractional solutions of the linear programs. Since linear programming is hard to approximate in parallel, we converted such linear programs into instances of positive linear programming, that can be approximated in NC.

In the case of the Max Sat problem we presented an algorithm that almost matches the best known sequential positive approximability result. For the Max Directed Cut problem, instead, the gap between the performances of polynomial-time algorithms and NC ones is still large. An intriguing question is whether the known non-approximability results for sequential algorithms can be improved when we restrict to NC algorithms (under the assumption that P $\neq$ NC). A possible way may be to devise *probabilistic proof systems for* P more efficient than the currently known proof systems for NP. Such a result would have a great independent interest. However, it is not clear why proofs for P should be *easier to check* than proofs for NP (they only appear to be *easier to generate*).

Acknowledgments

I wish to thank Pierluigi Crescenzi for suggesting the problem, encouraging this research, and giving several useful suggestions. I am also grateful with Madhu Sudan and Fatos Xhafa for helpful comments on a preliminary version of this paper.

References

[AKK95] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 284–293, 1995.

[AS92] N. Alon and J. Spencer. *The Probabilistic Method.* Wiley Interscience, 1992.

[Baz96] C. Bazgan. Personal communication. 1996.

[BC93] D.P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, 1993.

[BCA91] G. Bongiovanni, P. Crescenzi, and S. De Agostino. Descriptive complexity and parallel approximation of optimization problems. Manuscript, 1991.

[BGS95] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCP's and non-approximability – towards tight results (3rd version). Technical Report TR95-24, Electronic Colloquium on Computational Complexity, 1995. Extended abstract in *Proc. of FOCS'95.*

[CS95] B. Chor and M. Sudan. A geometric approach to betweennes. In *Proceedings of the 3rd European Symposium on Algorithms*, 1995.

[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 Israel Symposium on Theory of Computing and Systems*, 1995.

[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.*

[Hag92] D.J. Haglin. Approximating maximum 2-CNF satisfiability. *Parallel Processing Letters*, 2:181–187, 1992.

[HMR$^+$93] H. B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz, and R.E. Stearns. Every problem in MAX SNP has a parallel approximation algorithm. Manuscript, 1993.

[Hoc82] D. Hochbaum. Approximation algorithms for set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556, 1982.

[Kar84] N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[Kha79] L. G. Khachian. A polynomial time algorithm for linear programming. *Doklady Akademia Nauk SSR*, 224:1093–1096, 1979.

[KMS94] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, 1994.

[KMSV94] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 819–830, 1994.

[LN93] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 448–457, 1993.

[Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[LW95] M. Luby and A. Wigderson. Pairwise independence and derandomization. Technical Report TR-95-035, International Computer Science Institute, 1995.

[LW96] H.C. Lau and O. Watanabe. Randomized approximation of the constraint satisfaction problem. In *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory*, 1996.

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[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.*

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[RT87] P. Raghavan and C.D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[Ser91] M. Serna. Approximating linear programming is log-space complete for P. *Information Processing Letters*, 37, 1991.

[Shm95] D. Shmoys. Computing near-optimal solutions to combinatorial optimization problems. In *Combinatorial Optimization.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 20, 1995.

[SX95] M. Serna and F. Xhafa. On parallel versus sequential approximation. In *Proceedings of the 3rd European Symposium on Algorithms*, pages 409–419, 1995.

[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*, 1996.

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