On Parallel versus Sequential Approximation [*]

Maria Serna Fatos Xhafa

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Campus Nord, C6
Jordi Girona Salgado, 1-3
08034-Barcelona
E-mail: {mjserna,fatos}@goliat.upc.es

Abstract

In this paper we deal with the class NCX of NP Optimization problems that are approximable within constant ratio in NC. This class is the parallel counterpart of the class APX. Our main motivation here is to reduce the study of sequential and parallel approximability to the same framework. To this aim, we first introduce a new kind of NC-reduction that preserves the relative error of the approximate solutions and show that the class NCX has *complete* problems under this reducibility.

An important subset of NCX is the class MAXSNP, we show that MAXSNP-complete problems have a threshold on the parallel approximation ratio that is, there are positive constants ε_1, ε_2 such that although the problem can be approximated in P within ε_1 it cannot be approximated in NC within ε_2, unless P=NC. This result is attained by showing that the problem of approximating the value obtained through a non-oblivious local search algorithm is P-complete, for some values of the approximation ratio. Finally, we show that approximating through non-oblivious local search is in average NC.

1 Introduction

It is already well known that there are no polynomial time algorithms for NP-hard problems, unless P=NP, therefore for such problems the attention have been focused in finding (in polynomial time) approximate solutions. In this paper we consider NP Optimization (NPO) problems with polynomially bounded, in the input's length, objective function. The class APX consists of those NPO problems whose solutions can be approximated in polynomial time, with relative error bounded by a constant. This class is computationally defined, in that, to prove membership of a problem to this class we should give a polynomial time algorithm that finds a feasible solution to the problem whose measure is within a constant factor of the optimum or reduce it to an APX problem under a certain approximation preserving reduction. A natural question in this direction was whether there is a subclass of problems in APX which could be proved constant approximable in a generic way, or, alternatively, is there any complexity class (included in APX) whose members do not accept Polynomial Time Approximation Schemes? In order to give a precise characterization of such (possible) complexity classes, Papadimitriou and Yannakakis [PY91] used Fagin's

[*]This research was supported by the ESPRIT BRA Program of the European Community under contract No. 7141, Project ALCOM II.

syntactic definition of the class NP and introduced the classes MAXNP and MAXSNP. They proved that any problem in MAXSNP/MAXNP class can be approximated in polynomial time with constant ratio and many problems were shown to be MAXSNP-complete under L-reductions (for linear reductions). Later on, [KMSV94] proved that the class APX coincides with the closure under E-reductions of MAXNP and MAXSNP, thus reconciling both views (syntactic and computational) of approximability.

In the parallel setting we have an analogous situation. We consider the class of problems that are approximable within a constant ratio in NC that we denote NCX. Many properties are common for the classes NCX and APX. For example, in [DST93] it was shown that MAXSNP is contained in NCX, to do so they introduced L-reductions under the logspace criterion and proved that all known MAXSNP-complete problems proved in [PY91] are also complete under this reducibility. For the inclusion MAXSNP $\subseteq$ NCX [DST93] show that the proof of [PY91] can be achieved also in NC.

We first consider the possibility, for the class NCX, of having complete problems. To this aim we define some kind of NC-reduction, called NCAS-*reduction*, that preserves the "relative error" of approximations. This reduction generalizes the logspace L-reduction of [PY91] in the following sense: in order to preserve approximability L-reductions relate (linearly) the optima of both problems, while NCAS-reduction relate only the errors of the approximate solutions; in particular, NCAS-reduction has the property that constant approximations to one problem translates into constant approximations to the other. Our definition comes from the definition of PTAS-reduction [CP91]. We show that Max Bounded Weighted SAT is complete for NCX under NCAS-reductions, notice that this problem is also complete for the class APX under PTAS-reductions [CP91].

One general approach when dealing with hard combinatorial problems is to use a *local search* algorithm. Starting from an initial solution, the algorithm moves to a better one among its neighbors, until a *locally optimal* solution is found. This approach was used in [KMSV94] where they provided a characterization of MAXSNP in terms of a class of problems called Max kCSP (for Constraint Satisfaction Problem), and show that a simple non-oblivious local search provides a polynomial time approximation algorithm for the problems of the class. Thus every MAXSNP problem can be approximated within a constant factor by such algorithms, a fact that is in concordance with the (previously known) constant-approximability of MAXSNP, and furthermore the ratios achieved using this algorithms are comparable to the best-known ones. We analyze the parallel complexity of such approach. Notice that for NPO problems that are polynomially bounded, a simple observation shows that local search algorithms run in polynomial time. We first define what we call a *local problem* in which we are given an instance of the problem, a starting feasible solution and we ask for the value of the local optimal solution attained accordingly to a pre-specified local search algorithm. We show that the problem corresponding to non-oblivious local search is P-complete, furthermore it cannot be approximated in NC for some ratios, unless P=NC. Then, we use this result to show that there exists a threshold on the parallel approximation ratio of MAXSNP-complete problems, that is, there are constants ε_0, ε_1 such that the problem can be approximated in NC within ε_0, but not within ε_1, unless P=NC. In particular we show that the problem Max CUT can be approximated in NC within 1 but not within $1 - \varepsilon$, for some ε.

Although this results means that we cannot achieve in NC the best ratios for MAXSNP-complete problems, we analyze the expected behavior of a general non-oblivious local search algorithm. We show that the expected number of iterations is polylogarithmic in the instance size, when the search starts from a random initial solution and using a quite general improvement model.

2 Preliminaries, Basic Definitions and Problems

An NP Optimization problem is given by: (a) the set of instances, (b) the set of feasible solutions associated to any instance, (c) an objective function that maps feasible solutions of a given instance to (non-negative) rationals, referred to as the cost of the solution, and (d) we seek for a feasible solution that optimizes (maximizes, minimizes) the objective function.
Let Π be an NP Optimization problem, whose objective function is polynomially bounded with respect to the input length. Let I_Π denote the set of instances and let $\mathsf{Sol}_\Pi(x)$ denote the solution set to instance x. For any solution S, $S \in \mathsf{Sol}_\Pi(x)$, let $V(x,S)$ be the value of the objective function on S and let $\mathsf{Opt}(x)$ be the optimum value to instance x.

Definition 1 *An algorithm $\mathcal{A}$ approximates the problem Π with error $\varepsilon > 0$ if it finds a feasible solution S to instances x of Π such that*

$$\frac{1}{1+\varepsilon} \leq \frac{V(x,S)}{\mathsf{Opt}(x)} \leq 1+\varepsilon. \tag{1}$$

In this case we say that S has relative error within ε from the optimum solution and use the notation $E_\Pi(x,S) \leq \varepsilon$. The performance ratio *of $\mathcal{A}$ is r ($r \geq 1$), if it always finds a solution with error at most $r-1$.*

Let Π be a given problem such that for any instance x there is a unique solution to x, and let $\Pi(x)$ denote the (unique) value corresponding to the solution, Π is called a function problem.

Definition 2 *Given an instance x of any function problem Π and an $\varepsilon > 0$, the ε-Π problem is: compute a value $V(x)$ such that $\varepsilon\Pi(x) \leq V(x) \leq \Pi(x)$.*

Based on the definition of PTAS-reductions [CP91] we define the error preserving reductions in NC.

Definition 3 *Let A and B be two* NPO *problems. We say that the problem A is* NCAS*-reducible to B ($A \leq_{\mathsf{NCAS}} B$) if three functions f, g, c exist such that the following conditions hold:*

(a) For any $x \in I_A$, $f(x) \in I_B$ and the function f is computable in NC *with respect to $|x|$.*

(b) For any $x \in I_A$ and for any $y \in \mathsf{Sol}_B(f(x))$, $g(x,y) \in \mathsf{Sol}_A(x)$ and the function g is computable in NC *with respect to both $|x|$ and $|y|$.*

(c) $c : (0,1) \to (0,1)$ is an invertible rational valued function.

(d) For any $x \in I_A$ and for any $y \in \mathsf{Sol}_B(f(x))$ and for any rational ε, $\varepsilon \in (0,1)$, if $E_B(f(x),y) \leq c(\varepsilon)$ then $E_A(x, g(x,y)) \leq \varepsilon$.

From this definition we have that NCAS-reduction preserves the relative error of approximation that is, whenever $A \leq_{\mathsf{NCAS}} B$ then if we can find in NC approximate solutions for B implies that we can find in NC also approximate solutions for A. On the other hand this kind of reduction also "transmits" the non-approximability from A to B.

We recall also the L-reduction as defined in [PY91]. Given two NPO problems Π and Π', it is said that Π L-reduces to Π', if there exist a pair of functions (f,g) computable in polynomial time and two constants $\alpha, \beta > 0$ such that (a) function f transforms a given instance x of Π into an instance x' of Π' satisfying $\mathsf{Opt}(x') \leq \alpha \mathsf{Opt}(x)$ and (b) function g maps solutions of instance x' of cost C' into solutions of x of cost C in such way that $|C - \mathsf{Opt}(x)| \leq \beta |C' - \mathsf{Opt}(x')|$.

If we put the additional condition for an L-reduction to be achievable in logspace then clearly, we have that NCAS-reduction is a generalization of L-reduction (we choose $c(\varepsilon) = \varepsilon/\alpha\beta$, where α and β are the constants of L-reduction).
The following properties are immediate:

Proposition 1 *If $A \leq_{\mathsf{NCAS}} B$ and $B \in \mathsf{NCX}$ then $A \in \mathsf{NCX}$.*

Proposition 2 *The reduction $\leq_{\mathsf{NCAS}}$ is reflexive and transitive.*

Proposition 3 *If a problem A NCAS-reduces to problem B and A cannot be approximated in NC within some ε, $\varepsilon \leq \varepsilon_0$ then B cannot be approximated in NC within some ε', $\varepsilon' \leq c(\varepsilon_0)$ where c is the function given by the NCAS-reduction.*

Definition 4 *Let A be a problem in NCX. We say that A is complete for the class NCX under NCAS-reduction iff for any $B \in \mathsf{NCX}$, $B \leq_{\mathsf{NCAS}} A$.*

Through the paper we will consider the following problems:

Weighted Max CUT
Given a graph $G = (V, E)$ with positive weights on the edges, find a partition of V into two sets V_1 and V_2, such that the sum of the weights of the edges between V_1 and V_2 is maximized. When all the weights are unitary we have the unweighted Max CUT.
Weighted Max kSAT
Given a Boolean formula in CNF where each clause contains at most k literals and has a positive integer weight, find an assignment of $0/1$ to all variables that maximizes the sum of the weights of the satisfied clauses. When $k = 3$ we have the problem of Max 3SAT.
Weighted Not All Equal kSAT
We are given a set of weighted clauses with at most k literals of the form $\mathrm{NAE}(x_1, \ldots, x_k)$ *where each x_i is a literal or a constant $0/1$. Such a clause is satisfied if its constitutes do not have all the same value. We want to find an assignment to the variables such that the sum of the weights of the satisfied clauses is maximized. When the clauses do not contain negated literals the problem is called* POS NAE kSAT.
Max Bounded Weighted SAT
Given a set of clauses C over a set of variables $\{x_1, x_2, \ldots, x_n\}$ and weights $\{w_1, w_2, \ldots, w_n\}$ to the variables such that

$$W \leq \sum_{i=1}^{n} w_i \leq 2W, \tag{2}$$

find a truth assignment to the variables that maximizes the following measure function:

$$V(C, \tau) = \begin{cases} \max(W, \sum_{i=1}^{n} w_i \tau(x_i)), & \text{if } \tau \text{ satisfies all the clauses of } C, \\ \\ W, & \text{otherwise.} \end{cases} \tag{3}$$

Circuit True Gates (CTG) *Given an encoding of a Boolean circuit C together with an input assignment, compute the number of true gates to the given assignment, denoted by TG(C).*

3 NCX-Completeness

In order to prove the completeness result for NCX we will consider the Max Bounded Weighted SAT problem. Firstly, we observe that this problem is in NCX. For that, we note that the assignment $x_i = 1$, $1 \leq i \leq n$, has measure either W or $\sum_{i=1}^{n} w_i$ and therefore from (2) it gives an approximation with factor 1/2, that is a 1-approximation according to our definition.

Theorem 4 Max Bounded Weighted SAT *is* NCX*-complete under* NCAS*-reductions.*

Proof: Let Π be an optimization problem in NCX (suppose first that Π is a maximization one). In order to reduce Π to Max Bounded Weighted SAT we first reduce it, using a NCAS-reduction, to another problem Δ and then reduce Δ to Max Bounded Weighted SAT. Our reduction is based in that given in [CP91] but extended to the parallel setting.

Let T be the NC δ-approximation algorithm for Π. The problem Δ is as follows. The instances of Δ are those of Π and its measure function, for instance x and solution y, $V_\Delta(x, y)$ is:

$$V_\Delta(x, y) = a(x, \delta) + \max\{V_\Pi(x, y), t(x)\},$$

where

$$a(x, \delta) = \begin{cases} \frac{2\delta-1}{1-\delta} t(x), & \text{if} \quad \delta > \frac{1}{2}, \\ \\ 0, & \text{otherwise,} \end{cases}$$

and $t(x) = V_\Pi(x, T(x))$. Notice that in the new problem Δ is included the value $T(x)$ delivered by the approximation algorithm T. The idea is to obtain a problem with bounded measure since we want to reduce it to a *weighted* problem of bounded measure. In fact, if we denote by $l(x) = a(x, \delta) + t(x)$ then we have $l(x) \leq V_\Delta(x, y) \leq 2l(x)$, which means that the measure function of Δ satisfies the same kind of constraint (2) as Max Bounded Weighted SAT.

Now, the reduction from Δ to Max Bounded Weighted SAT goes as follows. Given an instance x of Δ, we apply Cook's theorem (see, e.g., [GJ79]). Then we will have a transformation (in polynomial time) from the problem Δ to SAT. We observe that this transformation ca be achieved also in logspace. Indeed, all the information we need each step of computation (in the work tape) for the variables in order to write the formula is: the step of the computation i, the index k of the actual state q_k of the machine, the index j of the tape square where read-write head is scanning and the index l of the bit of the input x, $x = s_{k_1} s_{k_2} \ldots s_l \ldots s_{k_n}$ that is scanning the machine. Therefore, the amount of the space we need in the work tape is logarithmic in the size of the input x since all these indices are bounded by a polynomial in the size of x. In other terms, the main need for the memory in such construction is for counting up to a polynomial in the length of the input and this can be done in logarithmic space.

Let φ_x be the boolean formula obtained. Let us denote by $y_1, y_2, \ldots, y_r$ the variables that describe the solution y and by $m_1, m_2, \ldots, m_s$ the boolean variables that give the solution $V_\Delta(x, y)$. Now, we assign weights to the variables. The variables m_i's receive weights 2^{s-i} and all other variables are assigned the weight 0. So, we have an instance of Weighted Max SAT. Since the measure $V_\Delta(x, y)$ is bounded then we have an instance of Max Bounded Weighted SAT (the constant W depends on the bound of the problem Δ). Furthermore, for any truth assignment which satisfies the formula, to recover a solution y is straightforward (we look at the values of y_i's). On the other hand, this transformation guarantees the relative error because $V_\Delta(x, y)$ is equal to the sum of the weights of the *true* variables.

For the rest of the theorem we have to prove that Π is NCX-reduced to Δ. The transformation is the following:

- For any instances x we let $f(x) = x$.
- For any instance x and any solution y that corresponds to instance $f(x)$ we take

$$g(x,y) = \begin{cases} y, & \text{if } t(x) \leq V_\Delta(x,y), \\ T(x), & \text{otherwise.} \end{cases}$$

- For any rational ε, $\varepsilon \in (0,1)$,

$$c(\varepsilon) = \begin{cases} \frac{1-\delta}{\delta}\varepsilon, & \text{if } \delta > \frac{1}{2}, \\ \varepsilon, & \text{otherwise.} \end{cases}$$

This transformation preserves the relative error when passing from solutions of Δ to those of Π (see details in [CP91]). Since NCAS-reductions compose we have that $\Pi \leq_{\mathsf{NCAS}}$ Max Bounded Weighted SAT. The minimization case uses similar arguments. □

4 The Parallel Complexity of Local Search Problems

The sequential complexity of local search algorithms has been extensively treated in [JPY88, SY91]. Here we deal with this issue in the parallel setting. Let us start by the definition of such algorithms.

Definition 5 (Local Search Algorithm)
Given a solution S of to a maximization (resp. minimization) NPO *problem Π and a $\delta > o$, the δ-neighborhood of S, denoted by $N(S,\delta)$, is the set of all solutions S' that have distance at most δ from S,*

$$N(S,\delta) = \{S' \mid \mathcal{D}(S,S') \leq \delta\},$$

where $\mathcal{D}(S,S')$ is the Hamming distance between S and S'. A solution S is locally optimal iff

$$\forall S' \in N(S,\delta),\ V(x,S) \geq V(x,S'), \quad (\textit{resp. } V(x,S) \leq V(x,S')).$$

A local search algorithm starts from an initial solution S and each iteration moves in the neighborhood of S from the current solution S_i to some solution $S_{i+1} \in N(S,\delta)$ with better cost, until it arrives at a locally optimal solution.

The time needed by any local search algorithm to find a locally optimal solution depends on the neighborhood structure used. So, there are local search algorithms for which locally optimal solutions are not known to be computable in polynomial time. However, there is a subclass of problems for which local search algorithms run in polynomial time.

Definition 6 *An* NPO *problem Π is polynomially bounded if there is a polynomial p such that $\mathsf{Opt}(x) \leq p(|x|)$, for any instance x of Π, where $|x|$ is the instance size.*

Proposition 5 *Local Search algorithms run in polynomial time for* NPO *problems that are polynomially bounded.*

The main observation of this (folklore) result is that in the case of polynomially bounded problems the number of steps to achieve a local optimium is polynomially bounded since any step of local search improves the value of the solution by an integral amount.

The local search defined above is also called *Standard Local Search* or *Oblivious Local Search.* A more generalized (astute) method for local search, *Non-oblivious Local Search* is given in [KMSV94]. The Non-oblivious Local Search was shown to be more powerful than the oblivious one since it permits to explore in both directions: that of the objective function and also that of the distance function. Non-oblivious local search algorithms were used successfully to approximate within a constant factor all MAXSNP problems.

Definition 7 (Non-oblivious Local Search Algorithm) *A Non-oblivious Local Search algorithm is a local search algorithm whose weight function is defined to be*

$$\mathcal{F}(I,S) = \sum_{\vec{x}} \sum_{i=1}^{r} p_i \Phi(I,S,\vec{x}),$$

where r is a constant, Φ_i's are quantifier-free first-order formulas and the profits p_i are real constants. The distance function $\mathcal{D}$ is any arbitrary polynomial time computable function and both $S, \vec{x}$ are structures.

Given a polynomially bounded NPO problem, we can define a *local problem* in which we fix a starting solution and seek for the value of the local optimum achieved by a non-oblivious local search algorithm. First, we define such a problem for Max CUT.

Definition 8 (Local Max CUT) *Given an instance of* Max CUT *and an initial solution S, find a locally optimum solution, achieved through a local search, starting from S.*

We show that the Local Max CUT problem is **non-approximable** in NC, unless P=NC. Our results builts on the result of [SY91] where was shown that finding a locally optimal solution to the unweighted Max CUT is P-complete. Moreover, we do not refer to any particular method (standard local search, non-oblivious local search, etc.) used to find the locally optimal solution, i.e. the non-aproximability result is independent of the method used. Our proof uses a reduction from the Circuit True Gates, a problem that was shown non-approximable in NC [Ser91], to Local Max CUT.

Theorem 6 *There is an $\varepsilon_1 > 0$ such that the ε-*Local Max CUT *is* P*-complete for any $\varepsilon < \varepsilon_1$.*

Proof: [Sketch] Given an instance of Circuit True Gates, let us consider the instance of CVP corresponding to it, i.e., the encoding a of the circuit together with the input assignment. Then, we use the reduction given in [SY91] to reduce the CVP to Local Max CUT. This reduction goes through three stages. In the first stage, the instance of CVP is reduced, in NC, to an instance of POS NAE 3SAT, in the second one POS NAE 3SAT is reduced to Weighted Max CUT and, finally, the last instance is transformed into an instance of the (unweighted) Max CUT. Here we give the most relevant points of the reduction (the reduction is quite involved), the full details are found in [SY91]. Our main observation here is to relate the value of the CUT with the number of true gates of the circuit instance from which we deduce the non-approximability result.

Having the instance of CVP, the variables for POS NAE 3SAT are as follows. For each gate g_i there is introduced a variable (denoted with the same symbol) g_i with the property that in any locally optimal truth assignment, the value of gate variable g_i is consistent with the corresponding value of the gate in the circuit. Further, there are introduced two groups of variables. The first, *control variables* y_i, z_i (corresponding to the ith gate), where $z_i = \neg y_i$. The intended meaning of such variables is to force that in any truth assignment which is locally optimal the variable gates are consistent with the circuit. The second group are *local variables*, associated to gate variable g_i: $\alpha_i^1, \alpha_i^2, \delta_i^1, \delta_i^2, \beta_i^1, \beta_i^2, \beta_i^3, \gamma_i^1, \gamma_i^2, \gamma_i^3, \omega_i$. The clauses of the instance POS NAE 3SAT are constructed from gate variables, control variables and local variables. In order to assign positive weights to the clauses, there are assigned positive weights to the variables and from them are computed (adequately) the weights for the clauses. So, to each variable is associated a positive weight (the weight of the variable v is denoted by $|v|$, n the instance size), as given below.

$$\begin{aligned}
&|g_i| = 100(2n+1-i)+60 \\
&|z_i| = |g_i| - 60, \quad |y_i| = |g_i| + 10 \\
&|\alpha_i^k| = |g_i| + 10, \quad |\delta_i^k| = |g_i| + 10, \quad k = 1,2 \\
&|\beta_i^k| = |g_i|, \quad |\gamma_i^k| = |g_i|, \quad k = 1,2,3 \\
&|\omega_i| = |\delta_i^1| - |g_i| = 10.
\end{aligned} \tag{4}$$

The weights of the clauses for POS NAE 3SAT are computed from (4). The instance I of POS NAE 3SAT has the property that, if an assignment to variables is not consistent with the circuit, then the local search will *correct* the value of those gate variables that violate the consistency.

In the second stage, from the instance I of POS NAE 3SAT is constructed the instance of Weighted Max CUT as follows:

- There is one vertex for each variable and two additional vertices labeled by 0 and 1;
- For every clause NAE(x, y) with weight W in I, there is included an edge between the vertices corresponding to the variables of the clause, with the same weight and for each clause NAE(x, y, z), three edges (x, y), (x, z), (y, z) with weight $W/2$ each, are included.

Regarding the weights of the clauses in the instance I (defined as function of the variable's weights) and the weights of the edges of the graph, the following two properties hold:

(1) an edge connecting two variable vertices u, v has weight equal to the product $|u| \cdot |v|$;

(2) the weight of the edge connecting a variable vertex v to a constant vertex 0/1 is a multiple $c|v|$ of the weight $|v|$.

From these properties there is deduced that any locally optimal solution (locally optimal CUT) to Weighted Max CUT induces a truth assignment to the variables of POS NAE 3SAT that is locally optimal.

In the final stage is constructed the instance of (unweighted) Max CUT, obtained from the instance of Weighted Max CUT by replacing every variable vertex v by a set N_v of $|v|$ vertices, and any edge (u, v) connecting two variable vertices by a a complete bipartite graph between N_u and N_v, and an edge connecting variable vertex v to a constant vertex 0/1, by edges connecting any vertex of N_v to the constant vertex. This assures that the new graph is unweighted and verifies the above property for locally optimal solutions. Going back to CVP it means that the input variables and gate variables in such assignment are consistent with the circuit. In other words, the values of the input variables coincide with the given input of the circuit and the gate variables have the value that is computed by the corresponding gates on that input.

Now, let v be a vertex. From properties (1)-(2), the total weight of its incident edges is a multiple of $|v|$, denoted $d(v) \cdot |v|$. Therefore the total weight of the cut (V_1, V_0) will be

$$W(V_1, V_0) = \sum_{v \in V_1} d(v) \cdot |v|, \tag{5}$$

where V_1 is the set of vertices corresponding to the true variables. Now, we express the weights of the variables (i.e. the weights of control and local variables) in terms of $|g_i|$ as given in (4). Thus the weight of the cut given in (5) is written as

$$W(V_1, V_0) = \sum_{g \in TG} f(|g|), \tag{6}$$

where TG denotes the set of true gates (i.e. $V_1 = TG$) and f is a linear function. But we can always find constants m and M such that

$$m \cdot TG(C) \leq \sum_{g \in TG} f(|g|) \leq M \cdot TG(C). \tag{7}$$

Therefore, from (6) and (7) it results that we cannot approximate in NC the value of a locally optimal CUT for any $\varepsilon < \varepsilon_1$, for some $\varepsilon_1 > 0$ that is a function of m and M, because it would imply that we can approximate in NC the number of true gates $TG(C)$ of the circuit. □

We can define in the same way as Local Max CUT, the Non-oblivious Local Search Problem. Using arguments similar to those of Theorem 6 we can construct, instead of a Max CUT instance, an instance of Non-oblivious Local Search Problem. Therefore we have the following:

Corollary 7 *There exists $\varepsilon_1 > 0$, such that approximating a Non-oblivious Local Search problem is* P*-complete for values of $\varepsilon < \varepsilon_1$.*

Suppose we have a problem Π and an algorithm $\mathcal{A}$ that approximates it for some ε_0 in polynomial time (for example, Max CUT and the standard local search algorithm). Furthermore, suppose that **the value** given by this algorithm cannot be approximated in NC for any $\varepsilon < \varepsilon_1$. In this situation, we naturally may ask whether there is a threshold in the approximation value such that the problem Π itself cannot be approximated in NC, i.e. whether the NC non-approximability of the value given by the algorithm translates into an NC non-approximability result for the problem itself.

Theorem 8 *Let x be an instance of an* NPO *problem Π and suppose that the algorithm $\mathcal{A}$ approximates Π within ε_0. If the value $A(x) = V(x, S)$ computed by the algorithm cannot be approximated in* NC *for $\varepsilon < \varepsilon_1$, for some $\varepsilon_1 > \varepsilon_0$ then Π cannot be approximated in* NC *for $\varepsilon < \varepsilon_2$, for some ε_2 that depends on ε_0 and ε_1.*

Proof: Since $\mathcal{A}$ approximates Π within ε_0 we have that

$$\frac{1}{1 + \varepsilon_0} \leq \frac{A(x)}{\mathsf{Opt}_\Pi(x)} \leq 1 + \varepsilon_0. \tag{8}$$

Suppose the contrary, that there is an NC algorithm B that approximates Π within some ε, $0 \le \varepsilon < \varepsilon_2$, that is

$$\frac{1}{1+\varepsilon} \le \frac{B(x)}{\mathsf{Opt}_\Pi(x)} \le 1+\varepsilon. \tag{9}$$

Now, we can write

$$\frac{B(x)}{A(x)} = \frac{B(x)}{\mathsf{Opt}_\Pi(x)} \cdot \frac{\mathsf{Opt}_\Pi(x)}{A(x)}$$

and therefore from (8) and (9) we have

$$\frac{1}{(1+\varepsilon_0)(1+\varepsilon)} \le \frac{B(x)}{A(x)} \le (1+\varepsilon_0)(1+\varepsilon). \tag{10}$$

If we chose ε_2 such that $\varepsilon_0 + \varepsilon_0(1+\varepsilon_2) = \varepsilon_1$ then the inequalities (10) mean that we can approximate $A(x)$ within $\varepsilon < \varepsilon_1$ and this contradicts the supposition. □

The following is another interpretation of the above result. Given an optimization problem Π, if the values of its approximate solutions obtained through certain resources (e.g. polynomial ones) cannot be approximated for all values of error parameter ε using other resources (e.g. parallel ones), then there is a threshold in approximating the problem Π itself in the second setting.
The result of Theorem 8 has also the following two implications. First, since non-oblivious local search algorithm approximate MAX CUT then, under the supposition that P $\neq$ NP there exist a positive constant ε such that MAX CUT cannot approximated in NC for factors smaller than ε. Secondly, recall that MAX CUT is MAXSNP-complete under logspace L-reductions [PY91], therefore from Theorem 6 and Theorem 8 we obtain:

Theorem 9 *For every* MAXSNP*-complete problem* Π*, there exist* ε_0*,* ε_1*,* $\varepsilon_1 \le \varepsilon_0$*, such that* Π *can be approximated in* NC *for any* $\varepsilon \ge \varepsilon_0$ *but cannot be approximated for any* $\varepsilon < \varepsilon_1$*.*

Proving constant approximability in NC is an important issue. Many constant factor approximation results in sequential can be translated also into parallel approximation results of (almost) the same factor. For example, Luby [Lub88] shows that a simple 1-approximation algorithm for MAX CUT that puts a vertex in one side of the cut with probability 1/2 can be done also in parallel. In [ACP94] was given a sequential 1-approximation for MAX 3SAT, we give a different and simple algorithm that achieves the same ratio in NC for the general MAX SAT.

Proposition 10 *There exists an* NC *algorithm that given an instance of* MAX SAT *finds an assignment to the variables that satisfies at least* $1/2$ *of the total number of clauses. The algorithms runs in* $O(\log n)$ *time and uses* $O(n)$ *processors in an EREW parallel machine.*

PROOF: We consider the following algorithm:

- Let V_j be the set of clauses where the variable x_j, $1 \le j \le n$, or its negation appears,

$$V_j = \{C_i \mid x_j \in C_i \text{ or } \neg x_j \in C_i\} \quad \text{for} \quad j \ge 1,$$

  and let us denote by n_j its cardinality, $n_j = |V_j|$.

- Sort the sequence of the sets V_j, $1 \le j \le n$ in non-increasing order of their cardinalities.

- Do a partition of the sets V_j that is, take $V_j := V_j - \cup_{i<j} V_i$.

- For all V_j compute: n'_j-the number of appearances of x_j in clauses of V_j, and n''_j-the number of appearances of $\neg x_j$ in clauses of V_j.
- For all j, if $n'_j \geq n''_j$ then assign $x_j :=$ True else assign $x_j :=$ False.

We claim that the assignment found above satisfies at least $m/2$ clauses. Indeed, we note that x_j satisfies at least $n_j/2$ clauses, therefore at least $\sum_{j=1}^{n} n_j/2 = m/2$ clauses are satisfied. It is straightforward to see that this algorithm can be efficiently implemented in EREW parallel machine using $O(n)$ processors and in $O(\log n)$ time. □

5 Expected performance of local search algorithms

Recall from the definition of local search that, given an instance of the problem and a solution to it, we must be able to determine in polynomial time whether the solution is locally optimal and, if not, to generate a neighboring solution of improved cost. That, on turn, means that we are considering NPO problems whose domain of feasible solutions has cardinality polynomial in the input size. We are interested in the expected number of iterations of any local improvement algorithm for such problems under any reasonable probabilistic model.

Let us give first some notations and considerations. Given an NPO problem Π, we may consider the set of its feasible solutions as a $q \log n$-hypercube, where n is the input size and q a constant that depends only on the instance. We can also suppose that the values of the objective function f for the problem at hand are distinct. Therefore, the vertices of the hypercube can be ordered from high to lower functional values and this is called an *ordering*. Given a set S of vertices in the hypercube, $\mathcal{B}(S)$ denotes the boundary of S consisting of all vertices not in S that are adjacent to some vertex in S, that is

$$\mathcal{B}(S) = \{y \mid \exists x \in S,\ x \text{ and } y \text{ are adjacent}\}.$$

We recall again a local improvement algorithm in its standard form:

1. Start at some random vertex (i.e. feasible solution) x;
2. Choose a vertex y adjacent to x such that $f(y) > f(x)$. If no such y exist, then stop.
3. Set x equal y and iterate Step 2.

Given an optimization problem there are two possible cases. The first, the local and global optima coincides. In this case the problem is called local-global and the improvement algorithm can be seen as a walk to the root of a tree whose vertices represent feasible solutions and the root represents the local optima. Secondly, the problem has multiple local optimas. In this case the improvement algorithm generates a forest with as many trees as local optimas there are. When the problem is local-global, the height of the tree gives us the maximum number of iterations done by the algorithm in order to find the optima. In the later, the number of iterations is given by the forest's depth, i.e., the maximum depth of any tree in the forest.

In order to evaluate the expected number of iterations done by the algorithm, we must precise how do we **choose** at step 2. Many reasonable probability distributions exist for this choice [Tov86]. Here we will consider the *boundary distribution,* defined as follows. Let $\mathcal{B}(i)$ be the boundary of the vertices chosen until step i. Then, the $(i+1)$th vertex is chosen uniformly at random in the boundary $\mathcal{B}(i)$. In fact, an even more simplified model will be considered. Instead of choosing randomly and uniformly from $\mathcal{B}(i)$, we consider the model where the $(i+1)$th vertex is chosen uniformly from a subset of $\mathcal{B}(i)$, namely that of the deepest vertices generated so far. Here is some intuition behind this new choice. If, instead of choosing among all vertices we choose among, say, the half deepest ones, then we would expect, at least intuitively, that the growth of the height would be "faster" than that of the height if we choose among all the vertices. Indeed, it turns out the second process *stochastically dominates* the first one, in the sense that the expected height in the second model is greater than or equal to that of the first one. So it suffices to find an upper bound for the expected height of the tree generated in the second model. A formal definition of *stochastic dominance* (see, e.g., [Roh76]) follows.

Definition 9 *If X and Y are two random variables with distribution functions $F_x(t)$ and $F_y(t)$, then we say that X stochastically dominate Y ($X \succeq Y$), if $F_x(t) \leq F_y(t)$, $\forall t$.*

It is clear that if X stochastically dominates Y then $E(X) \geq E(Y)$. The definition given above is naturally extended to sequences of random variables.

Definition 10 *Let $X = X_1, X_2, \ldots$ and $Y = Y_1, Y_2, \ldots$ be two sequences of random variables. We say that X stochastically dominates Y if, $\forall i$, $X_i \succeq Y_i$.*

Let $r = q \log n$ and $P_k = \sum_{j=0}^{k} \binom{r}{j}$, for some integer k. The following lemma gives a lower bound on the size of the boundary of a set of vertices in the r-hypercube.

Lemma 11 *Let $S(i)$ be the size of the smallest boundary of a set of size i. Then,*

- $S(i) = \binom{r}{k+1}$, *if* $i = P_k$;

otherwise we have

- $\binom{r}{k+1} \leq S(i)$, *if* $P_k < i < P_{k+1}$ *and* $k \leq (n-3)/2$;
- $\binom{r}{k+2} \leq S(i)$, *if* $P_k < i < P_{k+1}$ *and* $k \geq (n-3)/2$.

PROOF: It is clear that, for the value of i as specified above, the boundary of a set of i vertices in the $q \log n$-hypercube has at least $(q \log n)^k$ vertices. Then we apply Kruskal-Katona theorem [Kle81] that shows how to find a set of i vertices in a hypercube with minimal boundary size. From this theorem the bounds on $S(i)$ follow. □

The stochastic process described below, will stochastically dominate the pathlengths of the vertices of the tree. This process is called the *largest k-process* and is denoted by L^k. Let $k = k_1, k_2, \ldots$ be a sequence of positive integers. The *largest k-process* is the the sequence of random variables $L^k = L_1^k, L_2^k, \ldots$ whose values are generated as follows: $L_1^k = 1$; given the values of $L_1^k, L_2^k, \ldots, L_{i-1}^k$, we choose randomly and uniformly one of the k_i largest values and set this value plus one to L_i^k.

Lemma 12 *Given a set of i vertices on the $q \log n$-hypercube, let $B(i)$ denote a lower bound on its size. Let $k = k_1, k_2, \ldots, k_{n^q}$ be the sequence of integers where k_i is defined as $k_i = max(1, \lfloor B(i)/(q \log n - 1) \rfloor)$ and let $H = \{H_i\}$ be the sequence of random variables such that H_i denotes the height of the vertex i in the tree generated by the local search algorithm under boundary distribution. Then $L^k \succeq H$.*

PROOF: We observe that L^k is generated by choosing among the largest values, that means choosing among the deepest vertices generated so far. This fact assures that L^k stochastically dominates the pathlength of the vertices. Furthermore, by choosing the value for k_i, $k_i = \max(1, \lfloor B(i)/(q\log n - 1)\rfloor)$ it is guaranteed that the vertices are chosen accordingly the boundary distribution. $\square$

From this lemma, an upper bound for L^k is also an upper bound for the maximum pathlength on the tree. Let us denote by μ_k, the *growth rate* of L^k, i.e., its average increase. The key fact is the following theorem [AP81].

Theorem 13 [*AP81*] *Let m be a positive integer and let M be a sequence of m's. Then the expected rate of growth, μ_m, of the sequence L^M is less than or equal to e/m, for large m.*

Now, we are able to state the following result:

Theorem 14 *The expected number of iterations of any local improvement algorithm is less than:*

(*a*) *$e\alpha\log^2 n \log\log n$, if the problem is local-global and the probability distribution used is the boundary distribution.*

(*b*) *$e\beta\log n$, if the problem has multiple local optimas and under any probability distribution,*

where α and β are constants (that depend only on the problem).

Proof: The idea is to see the *largest k-process* as formed of subsequences each of them simulated for a fixed m. The rate growth for each subsequence is then given by Theorem 13. Let $s = \lfloor (r-1)/2 \rfloor$ and let us divide the set of 2^r vertices of the r-hypercube into the segments:

$$1 \le i \le P_s,\ P_s < i \le P_{s+1}, \cdots, P_{2s} < i \le P_{2s+1}.$$

The pathlengths of the vertices of the tree corresponding to each segment j, $1 \le j \le r$ are stochastically dominated by the subsequence of L^k with $m_j = k_j$, where k_j is given in Lemma 12. Thus, $L^k = L^{m_1},\ L^{m_2}, \ldots, L^{m_r}$. Therefore, the total expected height is less than

$$\textstyle\sum_{i=1}^{P_s} e(r-1)/B(i) + \sum_{P_s+1}^{2^r} e(r-1)/B(i)$$

$$\textstyle 1 + e(r-1)\sum_{k=1}^{s} \binom{r}{k}/\binom{r}{k} + e(r-1)\sum_{k=s}^{r-1} \binom{r}{k}/\binom{r}{k+1} + 1$$

$$\approx 2 + e(r-1)^2 + e(r-1)(r/2\ + r\log r/2)$$

$$< eq^2 \log^2 n \log\log n.$$

So, this is an upper bound for the expected number of iterations of the algorithm.

The proof for the case (b) uses similar arguments. We notice that in this case no matter how do we choose the vertex but, however, the way we choose must assure that all the *orderings* are equally likely. $\square$

Notice also that from this theorem we have that, in particular, oblivious and non-oblivious local search problems are in average NC, just use local improvement algorithms under any reasonable probabilistic model.

6 Acknowledgement

We thank P. Crescenzi for many helpful comments on a draft of this paper.

References

[ACP94] G. Ausiello, P. Crescenzi, and M. Protasi. Approximate Solution of NP Optimization Problems. Technical Report SI/RR - 94/03, Dipartimento di Scienze dell'Informazione. University of Rome, La Sapienza, 1994.

[AP81] D. Aldous and J. Pitman. *The Asymptotic Speed and Shape of a Particle System.* Cambridge University Press, 1981.

[CP91] P. Crescenzi and A. Panconesi. Completeness in Approximation Classes. *Information and Computation*, 93:241–262, 1991.

[DST93] J. Díaz, M. Serna, and J. Torán. Parallel Approximation Classes. In *Workshop on Parallel Algorithms, WOPA'93, San Diego*, 1993.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

[JPY88] D. Jonson, Ch. Papadimitriou, and M. Yannakakis. How Easy Is Local Search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

[Kle81] J.K. Kleitman. Hypergraphic Extremal Properties. In *Proceedings of the 7th British Combinatorial Conference.*, pages 59–78, 1981.

[KMSV94] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On Syntactic versus Computational Views of Approximability. In *Proceedings of 35th IEEE Symposium on Foundations of Computer Science*, pages 819–830, 1994.

[Lub88] M. Luby. Removing Randomness in Parallel Computation without a Processor Penalty. In *Proceedings of 29th IEEE Symposium on Foundations of Computer Science*, pages 162–173, 1988.

[PY91] C.H. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. *Computer and System Sciences*, 43:425–440, 1991.

[Roh76] V. Rohatgi. *An Introduction to Probability Theory and Mathematical Satistics.* John Wiley, 1976.

[Ser91] M. Serna. Approximating Linear Programming is Logspace Complete for P. *Information Processing Letters*, 37:233–236, 1991.

[SY91] A.A. Schaffer and M. Yannakakis. Simple Local Search Problems that are Hard to Solve. *SIAM Journal of Computing*, 20:56–87, 1991.

[Tov86] C. Tovey. Low Order Polynomial Bounds on the Expected Performance of Local Improvment Algorithms. *Journal of Mathematical Programming*, 35:193–224, 1986.