On the Complexity of Approximation, Local Search, and Local Approximation

Diplomarbeit

Hartmut Klauck

Uni-GH Paderborn

e-mail: hklauck@gmail.com

Supervisor: Prof. Dr. Georg Schnitger

July, 1995

Contents

1 Introduction

During the first half of the 1980's decade two seemingly unrelated areas of computer science came into contact: the once more awakening investigation of possible applications for artificial neural nets and the search for efficient methods to cope with the $\mathcal{NP}$-hardness of optimization problems. Hopfield introduced in 1982 a model of neural nets with symmetrically weighted connections performing an associative memory task [Ho82], which was later [HoTa85] found to be capable of expressing hard optimization problems. The energy function used in the earlier paper to describe the fact that stored patterns acted like "attractors" was found to be usable both as a language to express e.g. the TRAVELING SALESMAN PROBLEM (TSP), and as a definition for a network determining local optima of the energy function automatically. This interesting feature lead to a lot of experiments with the now so-called "Hopfield nets" (for an overview see [HKP91]). But disappointingly Hopfield nets, though consisting of fine-grained processors, seem to be working rather slowly, and the only available algorithm controlling their computation is sequential: one processor updates its state at a time.

While Hopfield's work (together with the reinvention of backpropagation) led to a new wave of interest in neural networks, more "traditionally" oriented computer scientists began to investigate the complexity of two approaches to optimization: approximation and local search. An approximation algorithm tries to find a solution that is "almost as good" as the global optimum. A local search algorithm looks for a local optimum, a solution which cannot be improved by a slight change, a step from one feasible solution to its (somehow defined and easily accessible) neighbor. Local search algorithms are known to be quite good in many practical applications (for more information see [PapSt82] and [JPaY88]), but finding local optima was shown to be computationally hard even for simple problems like MAX CUT (for exact definitions of all optimization problems considered in this paper see appendix A). In some cases approximation should be preferred, in others local search.

In [JPaY88] the complexity class $\mathcal{PLS}$ was introduced containing those local search problems for which every single search step takes polynomial time. This class lies somewhere between the functional equivalents of $\mathcal{P}$ and $\mathcal{NP}$, but seems to equal none of these. So $\mathcal{PLS}$-complete problems may have no efficient local search algorithm at all. We will refer to some results on this topic later. Another reason for the investigation of local search algorithms is the so-called "simulated annealing" technique (see [KiGV83]). This flamboyant metaphor names local search procedures which are randomized such that one can hope to be "shot" out of bad local optima by stochastic changes applied to solution vectors. The probability af these changes (referred to as "temperature") is decreased slowly during the stochastic optimization process in order to stay in a (hopefully good) local optimum at last.

At the same time the interest in parallel computing increased due to the progress in hardware technology. Whereas it is improbable that $\mathcal{NP}$-hard optimization problems have fast parallel algorithms (unless $\mathcal{NP} = \mathcal{NC}$) this is not clear for approximation or local search problems. For optimization problem with polynomially bounded optima local

search clearly takes at most polynomially many search steps. In this case a $\mathcal{PLS}$ problem may even have a fast parallel algorithm. Approximations are often also computable in $\mathcal{NC}$.

Differences between $\mathcal{NP}$-hard optimization problems regarding approximation and local search lead to complex situations. Some problems are hard for both approximation and local search (the TRAVELING SALESMAN PROBLEM cannot be approximated in polynomial time [PapSt82] (unless $\mathcal{P} = \mathcal{NP}$), local search versions of the TRAVELING SALESMAN PROBLEM are $\mathcal{PLS}$-complete [Pap92]). Some problems are hard to approximate and easy for local search (the INDEPENDENT SET problem cannot be approximated in polynomial time [BeSc92] (unless $\mathcal{P} = \mathcal{NP}$), but has an efficient parallel local search algorithm [Lu86]). Some problems are easy to approximate, but have no efficient local search algorithm unless $\mathcal{PLS} = \mathcal{P}$ resp. $\mathcal{P} = \mathcal{NC}$ (the MAX CUT problem, see section 4.2). Some problems are easy for both approximation and local search (INDEPENDENT SET restricted to constant degree graphs).

Hopfield's energy function can easily express both INDEPENDENT SET and MAX CUT and therefore belongs to the first category, the real "hard" problems. This result has two possible interpretations: One may consider such hard problems simply as intractable, or one may think of them as very expressive programming languages where computational hardness only reflects the power to encode problems. This second interpretation could be sensible in the case of the Hopfield energy function since it offers both a rather comfortable language using quadratic programming and a mechanism performing local search.

Whereas the original purpose of Hopfield nets was the solution of an associative memory task, this application was soon found to be not too successful. If the right application for Hopfield nets (assuming one exists) is combinatorial optimization, then practical considerations should govern the research interest. Most practical applications of Hopfield nets used a stochastic or simulated annealing approach. One is not really interested in solutions having the structural property of local optimality (as in the associative memory problem), but in "good" solutions, i.e., solutions that are at least as good as local optima. This leads to the question whether approximation and local search may be combined in an attempt to approximate local instead of global optima. For a heuristic approach to optimization this is as good as determining local optima exactly. The questions arising from these considerations are:

- Do $\mathcal{P}$-hard local search problems have fast parallel algorithms that approximate local optima?
- Do $\mathcal{PLS}$-complete local search problems have polynomial time algorithms that approximate local optima?

This paper tries to investigate the complexity of local approximation, i.e., of the task to find solutions to optimization problems with cost approximatively as good as the worst local optimum that has nonnegative cost — or arbitrarily better. The main tool to show that this task is hard will be a notion of reducibility. We define a reduction that preserves approximability with respect to the worst local optimum with nonnegative cost and show

that complete problems (for classes of local search problems) under this reduction exist. These problems have no considerably more efficient local approximation algorithms than local optimization algorithms.

The main part of the paper is an investigation of the complexity of the Hopfield energy function under the four approaches global optimization, global approximation, local optimization, and local approximation. The general Hopfield function is shown to be a very hard problem: it is complete for the class of $\mathcal{NP}$-maximization problems, and it is complete for the class of $\mathcal{PLS}$-maximization problems, both via approximability preserving reductions. This implies that no efficient algorithms exist that find or approximate local or global optima of the Hopfield energy function (unless $\mathcal{P} = \mathcal{NP}$ resp. $\mathcal{P} = \mathcal{PLS}$).

Therefore we investigate restrictions on the Hopfield energy function: restrictions on the sign of the weights and restrictions on the size of the weights. In the case of positive weights the complexity of global optimization collapses, but local approximation is shown to be rather hard compared to global optimization. The case of negative weights should also be easier than the general one (though very much harder than the case of positive weights), but this remains a conjecture. A restriction to polynomially or unit size weights is another possibility to decrease the complexity of the Hopfield energy function. It seems as if the Hopfield function were usable in different strengths with different complexities thus being a rather flexible tool for the expression of optimization problems.

We also consider approximation and local search for three graph cut problems that are very closely related to the three versions of the Hopfield energy function with restricted signs: a cut problem and its corresponding version of the Hopfield energy function are equivalent regarding local resp. global optimization. MAX CUT is related to the negatively weighted Hopfield function. MAX CUT and the MAX NP/MAX SNP problems defined in [PapY91] have very fast parallel approximation algorithms in contrast to the negatively weighted Hopfield function. s,t-MIN CUT is related to the positively weighted Hopfield function as well as to the famous s,t-MAX FLOW. Also a generalized MAX CUT problem with positive and negative weights is considered, that is related to the general Hopfield function.

The organization of this paper is as follows: section 2 presents detailed material and definitions on approximation, local search, and the new paradigm of local approximation. Section 3 investigates the Hopfield energy function. Section 4 investigates the graph cut problems. Definitions of optimization problems are provided in appendix A.

These are the contributions of the paper: We define a special kind of reduction that preserves approximability with respect to an approximation quality measure called "local performance ratio" and exhibit complete problems under such reductions for a hierarchy of classes containing local search problems. These complete problems are shown to be as hard to approximate as to optimize (locally). The main part of the paper investigates the complexity of the Hopfield energy function. The results of this section can be found in the following table.

MAX $\{0,1\}$-HOPFIELD $\sum_{i<j} w_{i,j}s_is_j - \sum_i t_is_i$		Positive Weights	Negative Weights	Pos./Neg. Weights
$\|1\|$-weights	Opt.	$\mathcal{RTC}^1$	$\mathcal{NP}$-cpl.	$\mathcal{NP}$-cpl.
Global	App.	n^ϵ-App. $\mathcal{NL}$-hard	n^ϵ-App. $\mathcal{NP}$-cpl.	n^ϵ-App. $\mathcal{NP}$-cpl.
$\|1\|$-weights	Opt.	$\mathcal{RTC}^1$	$\mathcal{P}$-cpl.	$\mathcal{P}$-cpl.
Local	App.	n^ϵ-App. $\notin \mathcal{AC}^0$	?	n^ϵ-App. $\notin \mathcal{AC}^0$
pol.-weights	Opt.	$\mathcal{RTC}^1$	$\mathcal{NP}$-cpl.	$\mathcal{NP}$-cpl.
Global	App.	n^k-App. $\mathcal{NL}$-hard	n^ϵ-App. $\mathcal{NP}$-cpl.	n^ϵ-App. $\mathcal{NP}$-cpl.
pol.-weights	Opt.	$\mathcal{RTC}^1$	$\mathcal{P}$-cpl.	$\mathcal{P}$-cpl.
Local	App.	n^k-App. $\mathcal{L}$-hard	?	n^k-App. $\mathcal{P}$-cpl.
exp.-weights	Opt.	$\mathcal{P}$-cpl.	$\mathcal{NP}$-cpl.	$\mathcal{NP}$-cpl.
Global	App.	2^n-App. $\mathcal{NL}$-hard	n^ϵ-App. $\mathcal{NP}$-cpl.	2^{n^ϵ}-App. $\mathcal{NP}$-cpl.
exp.-weights	Opt.	$\mathcal{P}$-cpl.	$\mathcal{PLS}$-cpl.	$\mathcal{PLS}$-cpl.
Local	App.	2^n-App. $\mathcal{NL}$-hard	?	2^{n^ϵ}-App. $\mathcal{PLS}$-cpl.

n denotes the number of vertices of a net, ϵ some positive constant, and k an arbitrarily large positive constant that depends on the polynomial bound attached to the weights. "Opt." refers to the complexity of computing optima, "App." to the complexity of approximation. $\mathcal{L}$ and $\mathcal{NL}$ abbreviate $\mathcal{LOGSPACE}$ resp. $\mathcal{NLOGSPACE}$. $\mathcal{RTC}^1$ stands for the class of functions computable by uniform probabilistic threshold circuit families of logarithmic depth.

In section 4 there is a an amplification result for s,t-MAX FLOW and s,t-MIN CUT (showing that they either have an approximation scheme in $\mathcal{NC}$ or cannot be approximated in $\mathcal{NC}$ within any constant in the case of unbounded weights). Also a $\mathcal{NLOGSPACE}$-hardness result for approximations of s,t-MAX FLOW, s,t-MIN CUT, and s,t-BLOCKING FLOW is derived. In contrast to this we have a constant quality $\mathcal{TC}^0$ approximation algorithm for all MAX NP problems (including MAX CUT). Cited material is marked explicitly or provided as "fact".

I would like to thank Georg Schnitger for his support during the development of this paper and especially for suggesting the general theme of approximation of local optima.

2 Definitions and Preliminaries

This section provides the necessary background for the development of a theory of local approximation. First there is some material on sequential and parallel complexity classes and on neural nets, afterwards information about approximation algorithms, local search, and a new combination of both is given.

2.1 Complexity Classes and Neural Nets

We want to investigate how hard it is to solve computational problems. A basic notion is that of a "complexity class". We begin by defining the most important complexity classes.

Definition 2.1 *A function $f : \{0,1\}^* \to \{0,1\}$ is called a "decision".*

The class $\mathcal{P}$ consists of all decisions computable by deterministic Turingmachines in polynomially bounded time.

The class $\mathcal{LOGSPACE}$ consists of all decisions computable by deterministic Turingmachines with logarithmically bounded worktape.

The class $\mathcal{PSPACE}$ consists of all decisions computable by deterministic Turingmachines with polynomially bounded worktape.

The classes $\mathcal{NP}$, $\mathcal{NLOGSPACE}$, $\mathcal{NPSPACE}$ consist of all decisions computable with the same resource bounds as $\mathcal{P}$ resp. $\mathcal{LOGSPACE}$ resp. $\mathcal{PSPACE}$, while allowing the Turingmachines to be nondeterministic.

Decisions f will be identified with the formal languages $f^{-1}(1)$.

The following fact is well known (see e.g. [J90]):

Fact 2.1 *1.* $\mathcal{LOGSPACE} \subseteq \mathcal{NLOGSPACE} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE} = \mathcal{NPSPACE}$

2. $\mathcal{NLOGSPACE} \neq \mathcal{PSPACE}$

It is unknown which of these inclusions (except the one noted in 2.) are strict. An exact comparison of these fundamental complexity classes belongs to the most important problems in computational complexity theory. Commonly it is assumed that all these classes are different from each other. A key concept to find problems that probably separate two complexity classes is reducibility. We now define the most basic reductions.

Definition 2.2 *A language L is said to be "polynomial time reducible" to a language M ($L \leq_p M$) if a mapping f exists such that $x \in L \iff f(x) \in M$ and f is computable by a Turingmachine in polynomially bounded time.*

A language L is said to be "$\mathcal{LOGSPACE}$ reducible" to a language M ($L \leq_{\mathcal{LOG}} M$) if a mapping f exists such that $x \in L \iff f(x) \in M$ and f is computable by a Turingmachine with logarithmically bounded tape.

A language L is said to be "hard" for a class of problems $\mathcal{K}$ with respect to some reduction R if all problems in $\mathcal{K}$ are R-reducible to L. L is called "complete" for $\mathcal{K}$ with respect to some reduction R if L is hard for $\mathcal{K}$ via R and L is a member of $\mathcal{K}$.

The following fact illustrates the way reductions are used (see [J90]):

Fact 2.2 *1. Polynomial time reductions are transitive.*

2. $\mathcal{LOGSPACE}$ reductions are transitive.

3. If a problem that is $\mathcal{NP}$-hard via polynomial time reductions is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$.

4. If a problem that is $\mathcal{P}$-hard via $\mathcal{LOGSPACE}$ reductions is in $\mathcal{LOGSPACE}$, then $\mathcal{LOGSPACE} = \mathcal{P}$.

Since we are interested in weak versions of hard problems complexity classes for parallel computations are important. We choose the most common ones based on Boolean circuits.

Definition 2.3 *A "Boolean circuit" on a basis Ω is a 5-tuple (V, I, y, E, ℓ), where I (the "input gates") is the set of sources of a directed acyclic graph $(V \cup I, E)$. V consists of the "internal gates", $y \in V$ is the "output gate" of the circuit. If $v \in V$ has indegree r, then $\ell(v)$ is a function ω from $\{0,1\}^r$ to $\{0,1\}$, the "gate function" computed by v. ω must be a member of Ω, which is a set of functions $\{0,1\}^r \to \{0,1\}$ for all natural numbers r. The function computed by the circuit on an input $x \in \{0,1\}^{|I|}$ is defined in an obvious recursive manner.*

A circuit has "fan-in" c if all its gates have indegree at most c. The "size" of a circuit is the cardinality of $V \cup I$, the "depth" is the length of a longest path from an input gate $i \in I$ to the output gate y.

The size of I is called the "input length". A single circuit has fixed input length and thus cannot compute functions on $\{0,1\}^*$.

Definition 2.4 *A "family of circuits" is a sequence $(C_n)_{n \in \mathbf{N}}$ of Boolean circuits, where circuit C_n has input length n. For a "uniform" family of circuits a standard representation of the circuit C_n must be computable (given n) by a Turingmachine with worktape bounded logarithmically in the size of C_n.*

The size and the depth of a circuit family are functions of n determining the size resp. the depth of each circuit C_n of the family.

A circuit family is said to have constant fan-in if some constant c exists that bounds the fan-in of every C_n. Otherwise the circuit family is said to have unbounded fan-in.

Circuit families that are nonuniform are able to compute any function on $\{0,1\}^*$, because every function restricted to $\{0,1\}^n$ is computable by a single circuit. Uniform circuit families on the other hand are not more powerful than Turingmachines.

We now define the classes of problems that are solvable by most common theoretical models of fast and efficient parallel computers.

Definition 2.5 *The class $\mathcal{NC}^k$ consists of all decisions computable by uniform fan-in 2 circuit families of size $O(p(n))$ and depth $O((\log n)^k)$ for a polynomial p and an integer constant $k \geq 1$.*

The class $\mathcal{AC}^k$ consists of all decisions computable by uniform unbounded fan-in circuit families of size $O(p(n))$ and depth $O((\log n)^k)$ for a polynomial p an a integer constant $k \geq 0$. The basis of $\mathcal{AC}^k$ circuits is restricted to AND, OR, and NOT-functions.

$\mathcal{NC}$ denotes the union of all $\mathcal{NC}^k$, $\mathcal{AC}$ the union of all $\mathcal{AC}^k$.

The class $\mathcal{NC}$ is called the class of "massively parallelizable problems". It can be defined in terms of circuits, computer networks, or PRAMS (see [KarRa90]), and is therefore robust. Note that $\mathcal{NC}^0$ has not been defined because constant fan-in circuits of sublogarithmic depth cannot use all of their input. The following fact is discussed in [J90].

Fact 2.3 *1.* $\mathcal{NC} = \mathcal{AC}$

2. $\mathcal{NC}^1 \subseteq \mathcal{LOGSPACE} \subseteq \mathcal{NLOGSPACE} \subseteq \mathcal{AC}^1$

Another important tool in algorithm design is randomization.

Definition 2.6 *A circuit that has additionally to its normal inputs m further inputs each of which takes the values 0 and 1 independently with probability $1/2$ is called a "probabilistic circuit". We say that a probabilistic circuit C performs a "randomized separation" of a decision $f(x)$ if the random variable $C(x)$ being the output of C on x fulfills*

$$prob[C(x) = 0 | f(x) = 0] = 1 \ \wedge \ prob[C(x) = 1 | f(x) = 1] \geq 1/2$$

for the described probability distribution on the new input variables. The definition is extended to circuit families by demanding that every circuit of the family performs a randomized separation.

A circuit complexity class that is named beginning with $\mathcal{R}$ (e.g. $\mathcal{RNC}^1$) contains those decisions for which a randomized separation can be performed by a uniform circuit family with the same restrictions as its deterministic counterpart.

In 1943 McCulloch and Pitts published [McPi43] a mathematical model of the activity of single neurons. This model is based on threshold-functions. We will consider threshold-functions as gate functions of unbounded fan-in circuits thus generalizing the classical circuit model. Other gate-functions used in neural networks are not considered.

Definition 2.7 *A decision $f : \{0,1\}^n \to \{0,1\}$ is called a (linear) threshold function if real numbers $w_1, \ldots, w_n, t$ exist such that $f = \theta_n(w_1, \ldots, w_n, t)$, where*

$$\theta_n(w_1, \ldots, w_n, t)(x) = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^n w_i x_i \geq t \\ 0 & \text{otherwise.} \end{cases}$$

A threshold circuit is an unbounded fan-in circuit whose basis is the set of all threshold-functions. The classes $\mathcal{TC}^k$ contain those functions computable by uniform polynomial size and $O((\log n)^k)$ depth threshold circuit families, where $k \geq 0$ is an integer constant. $\mathcal{TC} = \cup_0^\infty \mathcal{TC}^k$.

It is important that every threshold function can be written with integer weights of size $(n+1)^{(n+1)/2}/2^n$ (see [Mu71]). Another interesting fact is that (even exponential) weights have only small power: Replacing the threshold gates of a circuit by MAJORITY-gates (and thus by threshold gates with unit sized weights) costs a polynomial factor in size and only one additional layer in depth ([Gol92]). The first part of the following fact is obvious since AND, OR, NOT are threshold functions, and any threshold function can be evaluated by a $\mathcal{NC}^1$ circuit (see [Par94]).

Fact 2.4 *1. Let $k \geq 0$. Then: $\mathcal{AC}^k \subseteq \mathcal{TC}^k \subseteq \mathcal{NC}^{k+1} \subseteq \mathcal{AC}^{k+1}$.*

2. $\mathcal{AC}^0 \subseteq \mathcal{TC}^0 \subseteq \mathcal{NC}^1 \subseteq \mathcal{LOGSPACE} \subseteq \mathcal{NLOGSPACE} \subseteq \mathcal{AC}^1 \subseteq \mathcal{TC}^1 \subseteq \cdots \subseteq \mathcal{NC}$.

3. $\mathcal{NC} = \mathcal{AC} = \mathcal{TC} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE}$.

The only class in this hierarchy that is known to be strictly separated from its successors is $\mathcal{AC}^0$, by the functions PARITY and MAJORITY (see [J90]), which lie in $\mathcal{TC}^0$. Of course also $\mathcal{NLOGSPACE} \neq \mathcal{PSPACE}$ is well known.

We will use the convention to say that a function $f : \{0,1\}^* \to \{0,1\}^*$ is in "functional $\mathcal{K}$" for a complexity class $\mathcal{K}$ if the decisions f_i are in $\mathcal{K}$, where $f_i(x) = 1$ iff $f(x)$ has at least i outputs and the ith of these outputs is 1, and if the language consisting of all pairs $(x, |f(x)|)$ is in $\mathcal{K}$. Members of functional $\mathcal{P}$ will be called "computable in polynomial time".

Threshold circuits have been proved to be astonishingly powerful. The following results can be found in [SBKH93] and [SRo94].

Fact 2.5 *The following problems are computable in functional $\mathcal{TC}^0$:* ITERATED SUM *(of n numbers with n bits),* MULTIPLICATION *(of 2 numbers with n bits),* SORTING *(of n numbers with n bits),* MATRIX MULTIPLICATION *(of 2 $n \times n$-matrices with n-bit integers). This implies ([Pan85]) that* DETERMINANT *(of a $n \times n$-matrix with n-bit integers) can be computed in functional $\mathcal{TC}^1$.*

The neural networks described so far and those used in learning algorithms like back-propagation (see [HKP91]) are acyclic. General recurrent networks have not been investigated as thoroughly (see [Par94]). The most well-known type of recurrent neural network is the Hopfield net (see [Ho82]).

Definition 2.8 *A "Hopfield net" is a tuple (V, E, ℓ), where V is the set of neurons, E contains their connections, (V, E) forms an undirected graph (without self-loops), and ℓ is a mapping assigning integer weights to edges and thresholds to vertices.*

At the beginning of a computation of a Hopfield net a "state" $s_i \in \{0,1\}$ is assigned to every neuron v_i. Then some neuron takes the states of its neighbors v_j as inputs and computes the threshold function determined by the threshold $\ell(v_i)$ and the weights $\ell(v_i, v_j)$. The neuron changes its state when the output of this computation is different from its state. The goal of this process is to reach a stable state of the net, i.e., where no neuron will change its state.

As we will see later, reaching a stable state in a Hopfield net is the same as finding a local optimum of a suitably defined energy function.

2.2 Optimization Problems and Approximation Algorithms

Optimization problems make up a large and important part of the problems known to be $\mathcal{NP}$-complete (see [GJ79]) and thus commonly believed to be intractable by computers. But the traditional theory of reductions and complete problems is not perfectly suited to optimization problems for two reasons: first optimization problems are not decision problems and are normally placed in complexity classes by introducing some bound on the objective function. Secondly classical reductions do not preserve approximability. There are many approximation degrees among $\mathcal{NP}$-hard optimization problems (see [Ka92] for an overview). Now we define a general framework for $\mathcal{NP}$-optimization problems.

Definition 2.9 *A $\mathcal{NP}$-maximization problem $L = (P, C)$ is a function $opt_{P,C}(x) = \max\{C(s,x) | P(s,x)\}$, where $C : \{0,1\}^* \times \{0,1\}^* \to \mathbb{Z}$ is the "objective" (or "cost") function and $P : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ is the "feasibility predicate". P and C must be computable in polynomial time. x will be referred to as an "instance" of L, s as a "feasible solution" to x. Additionally we demand that $opt_{P,C}(x) \geq 0$ for all x, that $|s|$ is polynomially bounded in $|x|$ for all feasible s, and that it is possible to construct a feasible solution of nonnegative cost to any x in polynomial time.*

$\mathcal{NP}$-minimization problems are defined analogously.

Since probably no efficient algorithms for $\mathcal{NP}$-hard problems exist, one is interested in algorithms for hard optimization problems that find "good" solutions. An algorithm which produces a solution with cost near the optimum is called an "approximation algorithm". The quality of an approximation algorithm is commonly measured either with the "relative error" or the "performance ratio". We choose the latter to be concordant with the definitions we will make for local approximation algorithms. These will be allowed to give solutions arbitrarily better than (local) optima thus making relative errors useless.

Definition 2.10 *Let $L = (P, C)$ be some $\mathcal{NP}$-optimization problem. The "performance ratio" of a solution s to some instance x of L with global optimum $C(s_{opt}(x), x)$ is:*

$$R_L(s,x) = \begin{cases} C(s_{opt}(x),x)/C(s,x) & \text{for maximization problems} \\ C(s,x)/C(s_{opt}(x),x) & \text{for minimization problems} \end{cases}$$

if $C(s,x) > 0$ and $C(s_{opt}(x),x) > 0$. If $C(s,x) \leq 0$ or $C(s_{opt}(x),x) = 0$, then $R_L(s,x)$ is defined as $|C(s,x)| + C(s_{opt}(x),x) + 1$.

Let $\mathcal{A}$ be an algorithm that produces some feasible solution $s_{\mathcal{A}}(x)$ given an instance x of L. Then $\mathcal{A}$ is called a polynomial time ρ-approximation algorithm for L (for a function $\rho(|x|) \geq 1$) if $R_L(s_{\mathcal{A}}(x),x) \leq \rho(|x|)$ for all instances x of L, and if $\mathcal{A}$ runs in polynomial time (we say "$\mathcal{A}$ approximates L within ρ").

A family of algorithms $\mathcal{A}_\rho$ for every constant $\rho > 1$ is called a "polynomial time approximation scheme" (PTAS) for L if each $\mathcal{A}_\rho$ is a ρ-approximation algorithm for L, and if the running time of every algorithm $\mathcal{A}_\rho$ is bounded by a polynomial in the length

of its input, where $1/(\rho - 1)$ may appear in an exponent of the time bound. If ρ does not appear in an exponent of the time bound, then $\mathcal{A}$ is called a "fully polynomial time approximation scheme" (FPTAS).

An analogous definition can be made for $\mathcal{NC}$ ρ-approximation algorithms and a NCAS.

$\mathcal{NP}$-hard optimization problems can be distinguished by their approximability. There are problems which have a FPTAS or a PTAS, others are approximable within a constant but have no PTAS, and there are problems that cannot be approximated within a constant in polynomial time (assuming $\mathcal{P} \neq \mathcal{NP}$).

An especially interesting class of problems was introduced by Papadimitriou and Yannakakis in [PapY91]. They characterized a class of $\mathcal{NP}$-maximization problems with purely syntactical means based on a syntactical characterization of $\mathcal{NP}$ by Fagin [Fa74]. Their classes MAX NP and MAX SNP contain only problems that can be approximated within a constant in polynomial time, but problems hard for these classes have no PTAS (unless $\mathcal{P}$=$\mathcal{NP}$). More about MAX (S)NP in section 4.2.

To show that an optimization problem is hard to approximate one has to create a "gap" in its cost function, i.e., express some hard decision problem such that the cost of an optimum is at most W if the decision is negative, and larger than cW if the decision is positive (or conversely). Then a sufficiently good approximation (performance ratio c) allows to solve the decision problem and is thus shown to be hard.

Another important way to prove that a problem is hard to approximate is to use reductions. But polynomial time reductions are too weak for this: they are defined among language problems and do not take the approximability of optimization problems into account (these are turned into language problems). So a stronger type of reduction is needed, namely a mapping that preserves approximability. Then it is possible to create a gap for one problem and transfer this gap to other problems by reductions. Several attempts have been made to find a good notion for such a reduction. Performance ratio preserving reductions were introduced in [Sim90]. We choose a similar reduction defined in [Ka92] which is especially suited to reductions between problems that cannot be approximated within a constant. For such problems performance ratios must be measured as functions of the input size. The so-called "S-reductions" consider the **S**ize amplification of the mapping among two optimization problems carefully.

Definition 2.11 *Let L, M be two $\mathcal{NP}$-optimization problems. L "S-reduces with size amplification $a(n)$" to M if $a(n)$ is a monotone increasing positive function, and if there are two polynomial time computable functions f, g such that*

1. *f maps instances of L to instances of M. g maps pairs (solution to $f(x)$, x) to solutions to x.*

2. *For every instance x of L and every solution s to $f(x)$ the following holds:*

$$R_L(g(s,x),x) = O(R_M(s,f(x))).$$

3. *For every instance x of L: $|f(x)| \leq a(|x|)$.*

The following two facts are implicit in [Ka92].

Fact 2.6 *1. S-reductions are transitive.*

2. If L, M are both maximization (or both minimization) problems and $L \leq_S M$ with size amplification $a(n)$, and if there is a polynomial time ρ-approximation algorithm for M, where $\rho(n)$ is a monotone increasing function, then there is a polynomial $O(\rho(a(n)))$-approximation algorithm for L.

Fact 2.7 *Problems complete (via S-reductions) for all $\mathcal{NP}$-maximization (or minimization) problems with polynomially bounded optima cannot be approximated within n^ϵ in polynomial time for some constant $\epsilon > 0$.*

Problems complete (via S-reductions) for all $\mathcal{NP}$-maximization (or minimization) problems cannot be approximated within 2^{n^ϵ} in polynomial time for some constant $\epsilon > 0$.

A few problems with their approximation degree (see [Ka92]).

Problem	Approximation
MAX 2-SAT	MAX SNP-complete
MAX CUT	MAX SNP-complete
INDEPENDENT SET	no pol. time n^ϵ-app.
LONGEST PATH/FORBIDDEN PAIRS	cpl. for pol. bounded $\mathcal{NP}$-max. problems
MAX CIRCUIT OUTPUT	cpl. for $\mathcal{NP}$-max. problems

2.3 Local Search

Another approach to find "good" solutions for $\mathcal{NP}$-hard optimization problems is local search. Some polynomial time computable neighborhood structure is defined on the space of feasible solutions and one looks for a solution which has no neighbor that is better than itself. If all optima have polynomially bounded cost this yields a polynomial time algorithm, because every local search step produces a strictly improved solution. If the cost function is unbounded, local search can take exponential time, however.

Of course one can easily define instances of $\mathcal{NP}$-hard optimization problems where local search does not necessarily find a global optimum. Which local optimum is found depends on the initial solution from which local search starts (which is usually chosen randomly) and the rule that determines which one of the (possibly many) improving local changes is chosen. For some problems (e.g. MAX CUT) local optima approximate the global optimum in the sense of the previous section, but for many problems this is not true. Nevertheless local search has become one of the most popular approaches to $\mathcal{NP}$-hard optimization and finds good solutions in many practical situations (e.g. the Lin-Kernighan algorithm [LiKe73] for the TSP).

A question raised in [JPaY88] is how much time local search needs. There local search problems were examined for which every single search step takes polynomial time. These problems form the class $\mathcal{PLS}$. Since the cost function is unbounded, the problems may have exponential optima and local search can take exponential time. Later (in [SchY91])

several $\mathcal{P}$- and $\mathcal{PLS}$-completeness results were proved for seemingly easy problems like MAX CUT with the neighborhood that allows to change the side of one vertex in one step ($\mathcal{P}$- or $\mathcal{PLS}$-completeness depends on the size of weights).

A related question is whether local search can be parallelized. For the (non-approximable) unweighted INDEPENDENT SET problem an efficient ($\mathcal{NC}$) parallel algorithm is known, but the problem is monotone and a parallel greedy approach is possible (see [Lu86]). For other simple (and unweighted) problems $\mathcal{P}$-completeness results exclude an efficient parallel algorithm (unless $\mathcal{P} = \mathcal{NC}$). This is one of the major motivations for the consideration of algorithms that approximate local optima.

We now define local search problems in the same way as in [JPaY88].

Definition 2.12 *An $\mathcal{NP}$-optimization problem $L = (P, C)$ together with a neighborhood structure $N(s, x)$ that maps a solution and an instance to a set of neighboring solutions is called a $\mathcal{PLS}$ problem (for "polynomial local search"), if three polynomial time algorithms $\mathcal{A}_L$, $\mathcal{B}_L$, and $\mathcal{C}_L$ exist such that:*

1. *$\mathcal{A}_L$ produces (given an instance x of L) some feasible solution to x with nonnegative cost that can be taken as a starting point for local search.*

2. *$\mathcal{B}_L$ decides (given a string s and an instance x) whether s is a feasible solution to x and computes (if s is feasible) the cost $C(s, x)$.*

3. *$\mathcal{C}_L$ has two possible outputs given an instance x and a solution s to x: if s is a local optimum of x then $\mathcal{C}_L$ reports this fact and stops, otherwise $\mathcal{C}_L$ produces some strictly better solution s' out of $N(s, x)$.*

If the three algorithms are computable in the functional equivalent of a complexity class $\mathcal{K}$, then L belongs to a class we will call $\mathcal{KLS}$. If the cost function is polynomially bounded the class will be called $\mathcal{KLS}^{pol}$.

The three algorithms allow the (obvious) design of a local search algorithm that takes polynomial time (resp. has the complexity of $\mathcal{K}$) for every step. But this algorithm may have to do hard work [PapSY90]:

Fact 2.8 *There is a $\mathcal{PLS}$ problem L, which has a $\mathcal{PSPACE}$-complete standard algorithm problem, i.e., finding the solution that the standard algorithm induced by $\mathcal{A}_L$, $\mathcal{B}_L$, and $\mathcal{C}_L$ will output solves a $\mathcal{PSPACE}$-complete problem.*

Normally one is not interested in a special local optimum, but wants to find *some* local optimum of a $\mathcal{PLS}$ problem. To see that this task is possibly easier than the standard algorithm problem the class $\mathcal{PLS}$ can be compared to classes of search problems.

Definition 2.13 *A "search problem" is a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$. An algorithm "solves" a search problem R if, when given an $x \in \{0,1\}^*$, it either returns an y such that $(x, y) \in R$ or (correctly) reports that such a y does not exist.*

The class $\mathcal{P}_S$ contains those search problems that can be solved in polynomial time. The class $\mathcal{NP}_S$ contains those search problems that can be solved by nondeterministic polynomially time bounded Turingmachines.

The following is shown in [JPaY88].

Fact 2.9 *1.* $\mathcal{P} \subseteq \mathcal{P}_{\mathcal{S}} = \mathcal{PLS}^{pol} \subseteq \mathcal{PLS} \subseteq \mathcal{NP}_{\mathcal{S}}$.

2. If a $\mathcal{PLS}$*-problem solves a* $\mathcal{NP}$*-hard problem, then* $\mathcal{NP}$=Co-$\mathcal{NP}$.

Though $\mathcal{PLS}$ lies somewhere between $\mathcal{P}_{\mathcal{S}}$ and $\mathcal{NP}_{\mathcal{S}}$ it is rather improbable that $\mathcal{PLS}$ equals one of these. Equivalence to $\mathcal{NP}_{\mathcal{S}}$ would imply $\mathcal{NP}$=Co-$\mathcal{NP}$ (which is very improbable). Equivalence to $\mathcal{P}_{\mathcal{S}}$ is not known to have such drastical consequences. But a polynomial time algorithm for all of $\mathcal{PLS}$ would have to be a very sophisticated general approach and would e.g. provide a new proof that LINEAR PROGRAMMING has a polynomial time algorithm (LINEAR PROGRAMMING with the simplex neighborhood is a $\mathcal{PLS}$ problem for which local optimality implies global optimality).

For the purpose of investigating which $\mathcal{PLS}$-problems are hard a special kind of reduction that preserves local optimality was defined in [JPaY88].

Definition 2.14 *A* $\mathcal{PLS}$ *problem* L *is "*$\mathcal{PLS}$*-reducible" to a problem* M *(*$L \leq_{\mathcal{PLS}} M$*), if there are polynomial time computable functions* f *and* g *such that*

1. f *maps instances of* L *to instances of* M,
2. g *maps pairs (solution to* $f(x)$*,* x*) to solutions to* x,
3. *for all instances* x *of* L*: if* s *is a local optimum of* $f(x)$*, then* $g(s,x)$ *is a local optimum of* x.

If f *and* g *are computable in functional* $\mathcal{LOGSPACE}$*, then the reduction is called a "*$\mathcal{LLS}$*-reduction".*

This notion of reducibility makes sense:

Fact 2.10 *1.* $\mathcal{PLS}$*- and* $\mathcal{LLS}$*-reductions are transitive.*

2. *If* $L \leq_{\mathcal{PLS}} M$ *then a polynomial algorithm that finds local optima for* M *induces a polynomial algorithm that finds local optima for* L.
3. *If* $L \leq_{\mathcal{LLS}} M$ *then a* $\mathcal{NC}$ *algorithm that finds local optima for* M *induces a* $\mathcal{NC}$ *algorithm that finds local optima for* L.

An important aspect of local search problems is the neighborhood. A "larger" neighborhood structure can make local search harder, but may yield better local optima (e.g. the Lin-Kernighan neighborhood for the TSP compared to a neighborhood that allows only to remove two edges and insert two other edges). We are mostly interested in problems having simple neighborhoods.

Definition 2.15 *If a local search problem* L *allows all of* $\{0,1\}^n$ *as solutions to the instances with solution length* n*, then the neighborhood structure where for a solution* s *exactly those strings in Hamming distance* k *from* s *are neighbors is called the "*k*-flip neighborhood".*

The following fact is easy to infer from [JPaY88].

Fact 2.11 *For all $\mathcal{PLS}$ problems $L = (P, C, N)$ there is a $\mathcal{LLS}$-reduction to a problem $M = (P', C', N')$, such that P' is trivial, i.e., all strings of suitable length are solutions, and N' is the 1-flip neighborhood. Moreover, polynomially bounded optima keep this property.*

The result can even be strengthened regarding the complexity of the cost function.

Theorem 2.12 *1. Unbounded weight* MAX CUT *is in $\mathcal{TC}^0\mathcal{LS}$.*

2. Unweighted MAX CUT *is in $\mathcal{TC}^0\mathcal{LS}^{pol}$.*

3. Unbounded weight MAX CUT *is $\mathcal{PLS}$-complete via $\mathcal{PLS}$-reductions.*

4. Unweighted MAX CUT *is $\mathcal{PLS}^{pol}$-complete via $\mathcal{LLS}$-reductions.*

PROOF: The first two statements hold because all strings of suitable length are feasible solutions (making feasibility trivial), and computing the cost of a cut as well as testing whether a solution is locally optimal is possible in functional $\mathcal{TC}^0$ for MAX CUT regardless of the size of the weights: a string of length n over $\{0,1\}$ codes a cut by determining the side of every vertex as 0 or 1. For computing the cost of a given cut the weights of external edges have to be found (which is possible in functional $\mathcal{AC}^0$) and have to be summed (which is possible in functional $\mathcal{TC}^0$ due to fact 2.5). The algorithms $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ from definition 2.12 are easy to derive from this. Thus the unweighted problem is in $\mathcal{TC}^0\mathcal{LS}^{pol}$, the unbounded weight problem in $\mathcal{TC}^0\mathcal{LS}$.

The third statement is proved in [SchY91] as well as the key result leading to the fourth statement. A $\mathcal{PLS}^{pol}$ problem can be reduced to unweighted MAX CUT, because its standard algorithm can be implemented on a polynomial size Boolean circuit, and this circuit can be evaluated by finding local optima for unweighted MAX CUT.

It was shown in [SchY91] that the ($\mathcal{P}$-complete [J90]) CIRCUIT VALUE PROBLEM can be solved by finding local optima for the unweighted MAX CUT problem with the 1-flip neighborhood. The MAX CUT instances allow to read off the output(s) of a Boolean circuit directly from the unique locally optimal solution vector.

An instance x of a $\mathcal{PLS}^{pol}$ problem L can be mapped to a circuit computing the outputs of the standard local search algorithm of L on x (the circuit can be constructed in $\mathcal{LOGSPACE}$ from the Turingmachines of $\mathcal{A}_L$, $\mathcal{B}_L$, and $\mathcal{C}_L$). This circuit can be mapped in functional $\mathcal{LOGSPACE}$ to an unweighted MAX CUT instance as shown in [SchY91]. The outputs of the circuit can be read off from a locally optimal solution to the MAX CUT instance. This defines the two mappings of a $\mathcal{LLS}$ reduction. □

The problem FLIP is the originally used generic $\mathcal{PLS}$-complete problem ([JPaY88]): given a size n circuit with p inputs and q outputs, find an input assignment such that the natural number encoded by the outputs of the circuit is locally maximal over the 1-flip neighborhood. If the optimum is polynomially bounded (i.e., a size n circuit has $O(\log n)$ outputs) we will call the problem FLIP_{log} (or more exactly $\text{FLIP}_{k \cdot log}$ with $k \cdot \log n$ outputs).

FLIP is the local optimization version of MAX CIRCUIT OUTPUT. Generally the circuit must consist of fan-in 2 and fan-out 2 AND/OR gates and of NOT gates. If the depth is restricted to $\log^c n$, then the problem is called $\mathcal{NC}^c$-FLIP resp. $\mathcal{NC}^c$-FLIP$_{log}$.

Theorem 2.13 *1. $\mathcal{NC}^1$-FLIP$_{log}$ is $\mathcal{PLS}^{pol}$-complete via $\mathcal{LLS}$-reductions.*

2. $\mathcal{NC}^1$-FLIP is $\mathcal{PLS}$-complete via $\mathcal{PLS}$-reductions.

PROOF: It is easy to see that $\mathcal{NC}^1$-FLIP$_{log}$ is in $\mathcal{PLS}^{pol}$ and $\mathcal{NC}^1$-FLIP in $\mathcal{PLS}$. Now to the hardness results. Theorem 2.12 implies that unweighted MAX CUT is $\mathcal{PLS}^{pol}$-complete via $\mathcal{LLS}$-reductions. The $\mathcal{LLS}$-reduction from MAX CUT to $\mathcal{NC}^1$-FLIP$_{log}$ consists first of a mapping f from a graph G to a circuit C computing the cost of a cut in G (given an input coding this cut). Clearly G has the same locally optimal solutions as C. The reduction consists of the mapping f and a mapping g defined by $g(s, x) = s$.

The circuit C can be a $\mathcal{NC}^1$ circuit, because the cost of the cut can be computed by a layer determining the weights of external edges, and a circuit summing these (in functional $\mathcal{NC}^1$ due to facts 2.5/2.4). The cost function is polynomially bounded, and so C is an instance of $\mathcal{NC}^1$-FLIP$_{log}$. C can be constructed in $\mathcal{LOGSPACE}$ due to fact 2.5, leading to a $\mathcal{LLS}$-reduction.

$\mathcal{PLS}$-completeness was proved for unbounded weight MAX CUT in [SchY91]. The same argument as above yields a $\mathcal{PLS}$-reduction to $\mathcal{NC}^1$-FLIP. □

This subsection concludes with a list of completeness results ([SchY91]/[Pap92]):

Unweighted or polynomial weights; $\mathcal{PLS}^{pol}$-complete	$\mathcal{NC}^1$-FLIP$_{log}$ MAX CUT MAX 2-SAT
Unbounded weights; $\mathcal{PLS}$-complete	$\mathcal{NC}^1$-FLIP MAX CUT MAX 2-SAT TSP, Lin-Kernighan or k-change

2.4 Approximation of Local Optima

In practical applications local search is often associated with some probabilistic approach. An initial solution is chosen randomly and the search for a local optimum begins (improving neighbors may also be chosen randomly). To avoid being trapped in a bad local optimum this process can be repeated some times. Another variant is the simulated annealing approach that allows search steps that worsen the solution, but decreases their probability with time thus "annealing" the "temperature" of the stochastic optimization process. These methods often yield quite good solutions. It is obvious that it makes no difference if the obtained solution is really locally optimal or if it is as good as a local optimum—one is not interested in the structural property of local optimality but only in a heuristic for "good" solutions. Fast parallel approximation of local optima would be a desirable improvement to the exact solution of $\mathcal{PLS}^{pol}$-hard local search problems.

Note that simulated annealing algorithms also solve their corresponding local search problem. The interest in parallel computation and the completeness results of the previous subsection raise the following questions:

- May local optima of interesting $\mathcal{PLS}^{pol}$-complete problems be approximated by fast parallel algorithms (i.e., in functional $\mathcal{NC}$)?
- May local optima of interesting $\mathcal{PLS}$-complete problems be approximated in polynomial time?

In order to create a general framework for these problems the following definition is helpful.

Definition 2.16 *Let $L = (P, C, N)$ be a $\mathcal{PLS}$ problem. The "local performance ratio" of a solution s to an instance x of L with "worst" nonnegative local optimum $C(s_{opt}(x))$ is in the case of maximization*

$$R_L^{loc}(s,x) = \begin{cases} 1 & \text{if } C(s,x) \geq C(s_{opt}(x),x) \\ C(s_{opt}(x),x)/C(s,x) & \text{if } 0 < C(s,x) < C(s_{opt}(x),x) \\ C(s_{opt}(x),x) + |C(s,x)| + 1 & \text{if } C(s,x) \leq 0 \leq C(s_{opt}(x),x) \end{cases}$$

and in the case of minimization

$$R_L^{loc}(s,x) = \begin{cases} 1 & \text{if } 0 \leq C(s,x) \leq C(s_{opt}(x),x) \\ C(s,x)/C(s_{opt}(x),x) & \text{if } C(s,x) > C(s_{opt}(x),x) > 0 \\ C(s,x) + 1 & \text{if } C(s,x) > C(s_{opt}(x),x) = 0 \end{cases}$$

Let $\mathcal{A}$ be an algorithm that produces some feasible solution $s_{\mathcal{A}}(x)$ given an instance x. $\mathcal{A}$ is called a polynomial time [$\mathcal{NC}$] local ρ-approximation algorithm for a function $\rho(|x|) \geq 1$ if $R_L^{loc}(s_{\mathcal{A}}(x),x) \leq \rho(|x|)$ for all instances x, and $\mathcal{A}$ can be implemented on a uniform polynomial size [and polylogarithmic depth] circuit family.

A family of algorithms $\mathcal{A}_\rho$ for every constant $\rho > 1$ is called a polynomial time [$\mathcal{NC}$] local approximation scheme, PTLAS [NCLAS], if each $\mathcal{A}_\rho$ is a ρ-approximation algorithm and these algorithms are implementable on unifom polynomial size [and polylogarithmic depth] circuit families, where the bounds depend on $1/(\rho - 1)$ and the input length.

Other definitions for local approximation would be possible, for instance demanding from an approximating solution to have cost really near the cost of a local optimum, or to be near a local optimum regarding some distance measure on solution vectors. But since we are interested in good optimization heuristics we chose this notion of "being almost as good as (or even better than)" a local optimum. We are interested either in designing fast local approximation algorithms, or in proofs that such algorithms do not exist. For the latter purpose reductions will be useful again. This time we need a very special kind of reduction—one that preserves approximability with respect to local optima.

Definition 2.17 *A local performance ratio preserving reduction (LPR-reduction) among $\mathcal{PLS}$ problems L and M is defined as follows: L is $\mathcal{K}$-LPR-reducible to M ($L \leq_{LPR}^{\mathcal{K}} M$) with size amplification $a(n)$ for a monotone increasing positive function $a(n)$, if two functions f, g computable in functional $\mathcal{K}$ exist such that*

1. *f maps instances of L to instances of M,*

2. *g maps pairs (solution to $f(x)$, x) to solutions to x,*

3. *for all instances x of L and solutions s to $f(x)$: $R_L^{loc}(g(s,x),x) = O(R_M^{loc}(s,f(x)))$,*

4. *$|f(x)| \leq a(|x|)$.*

Theorem 2.14 *1. $\mathcal{K}$-LPR-reductions are transitive for $\mathcal{K} = \mathcal{LOGSPACE}$, $\mathcal{K} = \mathcal{NC}^k$, $\mathcal{K} = \mathcal{NC}$, $\mathcal{K} = \mathcal{P}$.*

2. *If two $\mathcal{PLS}$ problems L and M fulfill $L \leq_{LPR}^{\mathcal{NC}} M$ with size amplification $a(n)$, and M has a $\mathcal{NC}$ local ρ-approximation algorithm for a monotone increasing function ρ, then L has a $\mathcal{NC}$ local $O(\rho(a(n)))$-approximation algorithm.*

3. *If two $\mathcal{PLS}$ problems L and M fulfill $L \leq_{LPR}^{\mathcal{P}} M$ with size amplification $a(n)$ and M has a polynomial time local ρ-approximation algorithm for a monotone increasing function ρ, then L has a polynomial time local $O(\rho(a(n)))$-approximation algorithm.*

PROOF:

1. Assume $L \leq_{LPR} M \leq_{LPR} N$ with functions f_1, g_1, a_1 and f_2, g_2, a_2 for the first resp. second reduction. Then $f = f_1 \circ f_2$ maps an instance of L to an instance of N. g_2 maps a solution s to $f(x)$ and the instance $f_1(x)$ of M to a solution to $f_1(x)$, g_1 maps this solution and instance x to a solution to x. This defines a mapping g that produces a solution with local performance ratio

$$\begin{aligned} R_L^{loc}(g(s,x),x) &= R_L^{loc}(g_1(g_2(s,f_1(x)),x),x) \\ &= O(R_M^{loc}(g_2(s,f_1(x)),f_1(x))) \\ &= O(R_N^{loc}(s,f_2(f_1(x)))) \\ &= O(R_N^{loc}(s,f(x))). \end{aligned}$$

   Thus f and g define a LPR-reduction. The size amplification is $a_1 \circ a_2$. It is well known that the composition of two $\mathcal{LOGSPACE}, \mathcal{NC}^k, \mathcal{NC}, \mathcal{P}$-reductions is computable in the same functional class.

2. An algorithm for L works as follows: given an instance x first compute $f(x)$, then approximate a local optimum with the algorithm for M, then map the obtained solution s to a solution to x using g. Then

$$R_L^{loc}(g(s,x),x) = O(R_M^{loc}(s,f(x))) = O(\rho(|f(x)|)) = O(\rho(a(|x|))).$$

3. Analogous to 2. □

Theorem 2.15 *1.* $\mathcal{NC}^k$-FLIP$_{log}$ *is complete for the* $\mathcal{NC}^k\mathcal{LS}^{pol}$*-maximization problems via* $\mathcal{NC}$*-LPR-reductions.*

2. $\mathcal{NC}^k$-FLIP *is complete for the* $\mathcal{NC}^k\mathcal{LS}$*-maxim. problems via* $\mathcal{NC}$*-LPR-reductions.*

3. FLIP *is complete for the* $\mathcal{PLS}$*-maximization problems via* $\mathcal{P}$*-LPR-reductions.*

PROOF: First to the inclusion statements. Clearly the bound on the cost function in 1. holds. Furthermore it has to be shown that the three algorithms of definition 2.12 belong to functional $\mathcal{NC}^k$ (statement 1. and 2.) resp. functional $\mathcal{P}$ (statement 3.). For this it suffices to show that the cost function is computable in these classes (given a solution and an instance, i.e., a string and a circuit). It is well known that universal circuit families exist that can simulate any circuit of size at most c and depth at most d with polynomial size (in c) and depth $O(d)$ (see [We87], theorem 8.3). The universal circuits compute the cost of solutions to FLIP instances, while having asymptotically the same depth that the input circuits are allowed to have. The universal circuit family has polynomial size. This places the cost function in the desired class.

Now to the hardness results. We show how to LPR-reduce every $\mathcal{PLS}$-maximization problem L to FLIP in a way that preserves polynomial bounds on the cost function and leads to circuits (FLIP-instances) having asymptotically at most the same depth as the circuits of the algorithms $\mathcal{A}_L$, $\mathcal{B}_L$, $\mathcal{C}_L$. This implies the theorem.

Let $L = (P, C, N)$ be some maximizing local search problem in $\mathcal{PLS}$. First L is LPR-reduced to an intermediate problem Q with a trivial feasibility predicate, the 1-flip neighborhood, and only solutions of nonnegative cost. This can be done by a slight modification of the proof that every $\mathcal{PLS}$-problem can be $\mathcal{PLS}$-reduced to a $\mathcal{PLS}$ problem with these properties given in [JPaY88]. There FLIP was defined as a minimization problem and its maximization version was reduced to this minimization problem by flipping all output bits of a given circuit, a technique that clearly does not preserve approximability. So the proof is restated here with the necessary modifications for maximization, and the necessary observations for showing that this is a LPR-reduction (remember also that we have allowed local search problems to have solutions with negative cost, which causes some extra trouble). After the reduction to Q it is easy to LPR-reduce Q to FLIP, because all structure of Q lies in its (nonnegative) cost function.

Lemma 2.16 $\mathcal{PLS}$*-maximization problems* L *can be* $\mathcal{P}$*-LPR-reduced to* $\mathcal{PLS}$*-maximization problems* Q *with trivial feasibility predicate, the 1-flip neighborhood, and only solutions of positive cost, where polynomial bounds on the cost function are preserved. If* $L \in \mathcal{NC}^k\mathcal{LS}$*, then* $Q \in \mathcal{NC}^k\mathcal{LS}$ *and the reduction is computable in functional* $\mathcal{NC}$*.*

We may assume that all solutions to an instance x of L have the same length $p = poly(|x|)$ and that no two of them are within Hamming distance 1 of each other. The neighborhood can be restricted so that each solution has at most one neighbor, which is the one returned by algorithm $\mathcal{C}_L$ from the standard local search procedure implicit in 2.12

(note that neighborhoods are allowed to be asymmetric). This modified problem has the same local optima as L, now these optima have no neighbor. Note that the modification of L can be computed in functional $\mathcal{LOGSPACE}$.

Now the modified problem L will be reduced to Q. The set of solutions to $f(x)$ in Q is $\{0,1\}^{2p+2}$. The neighborhood is 1-flip. The cost function will be designed such that only strings corresponding to solutions of nonnegative cost to x in L are candidates for local optima of $f(x)$ in Q. The cost function C_Q of Q is defined as follows (depending on the cost function C_L of L): Let u be any feasible solution to x with nonnegative cost.

1. $uu11$: cost $(2p+5)(C_L(u,x)+1)-2$
2. $uu10$: cost $(2p+5)(C_L(u,x)+1)-1$
3. $uu00$: cost $(2p+5)(C_L(u,x)+1)$
4. $uv00$, where u is not locally optimal and v is on the shortest Hamming path from u to its neighbor w: cost $(2p+5)(C_L(w,x)+1)-p-h-4$, where h is the Hamming distance between v and w $(h \geq 0)$
5. $vu10$, v is any string of length p: cost $(2p+5)(C_L(u,x)+1)-p-3$
6. $vu11$, v is any string of length p: cost $(2p+5)(C_L(u,x)+1)-h-2$, where h is the Hamming distance between v and u.

Every string $vwyz$ of length $p+p+1+1$ that is *not* in this list has cost $p-h+1$ for the Hamming distance h between w and the standard solution u_s given by algorithm $\mathcal{A}_L$.

First note that every solution to $f(x)$ has positive cost, because u and u_s have nonnegative cost, and $h \leq p$ is always true.

Now consider any solution $vwyz$ to $f(x)$ that is not in the list 1)–6). There is a local search path from $vwyz$ to vu_s11, because $C_Q(vu_s11, f(x)) \geq p+3$ due to 6) and $C_Q(vwyz, f(x)) = p-h+1$ for all w in Hamming distance h from u_s. $vwyz$ can be changed to vu_syz and afterwards to vu_s11 via 5) and 6). A solution $vu11$ (for some feasible solution u to x with nonnegative cost) can be changed to $uu00$ via 6) and 1),2),3).

Only solutions $uu00$ to $f(x)$ (for some feasible solution u to x with nonnegative cost) can be locally optimal, because all other solutions in the list can be improved. $uu00$ is locally optimal (for $f(x)$) iff u is locally optimal (for x): if u is no local optimum then $uu00$ can be changed to $uw00$ using 4), to $ww11$ using 5),6), and to $ww00$ using 1), 2), 3). If u is a local optimum then u has no neighbor w, and $uu00$ is locally optimal, because no solution to x is in Hamming distance 1 from u, and thus no neighbor of $uu00$ in the list has larger cost than $uu00$.

Now to the exact definition of the reduction to Q: an instance x of L is mapped to an instance $f(x)$ of Q as described: all strings of length $2p+2$ are solutions, the neighborhood is 1-flip, the cost is as just defined. The three algorithms $\mathcal{A}_Q$, $\mathcal{B}_Q$, $\mathcal{C}_Q$ are easy to derive from that. A solution $vwyz$ to $f(x)$ is mapped by g to w, if w is a solution to x with nonnegative cost, to $N(v)$ (the neighbor of v in x) if $vwyz$ fulfills case 4) of the list, otherwise to the standard solution u_s given by $\mathcal{A}_L$. The mapping f is computable in

functional $\mathcal{LOGSPACE}$. g is computable in functional $\mathcal{NC}^k$ resp. polynomial time if $\mathcal{A}_L$, $\mathcal{B}_L$, and $\mathcal{C}_L$ are. Note that $Q \in \mathcal{NC}^k\mathcal{LS}$ if $L \in \mathcal{NC}^k\mathcal{LS}$, and that a polynomial bound on the cost function is preserved.

We have to show that $R_L^{loc}(g(vwyz, x), x) = O(R_Q^{loc}(vwyz, f(x)))$ for every solution $vwyz$ to $f(x)$. Let $s_{opt}(x)$ and $s_{opt}(f(x))$ denote locally optimal solutions to x resp. $f(x)$ with smallest nonnegative cost.

Let $s = g(vwyz, x)$. Then $s = u_s$ implies $C_L(w, x) < 0$ and $0 < C_Q(vwyz, f(x)) \leq p$. Otherwise $s = w$ or $s = N(v)$. Note that $C_L(s, x) \geq 0$. It will be important that $(2p + 5)(C_L(s, x) + 1) = C_Q(ss00, f(x)) \geq C_Q(vwyz, f(x))$.

1. If $C_L(s, x) \geq C_L(s_{opt}(x), x)$ then $R_L^{loc}(s, x) = 1 \leq R_Q^{loc}(vwyz, f(x))$.

2. If $C_L(s_{opt}(x), x) > C_L(s, x) = 0$ then

$$\begin{aligned} R_Q^{loc}(vwyz, f(x)) &= \frac{C_Q(s_{opt}(f(x)), f(x))}{C_Q(vwyz, f(x))} \\ &\geq \frac{(2p + 5)(C_L(s_{opt}(x), x) + 1)}{(2p + 5)(C_L(s, x) + 1)} \\ &= C_L(s_{opt}(x), x) + 1 = R_L^{loc}(s, x). \end{aligned}$$

3. If $C_L(s_{opt}(x), x) > C_L(s, x) > 0$ then

$$\begin{aligned} R_Q^{loc}(vwyz, f(x)) &= \frac{C_Q(s_{opt}(f(x)), f(x))}{C_Q(vwyz, f(x))} \\ &\geq \frac{(2p + 5)(C_L(s_{opt}(x), x) + 1)}{(2p + 5)(C_L(s, x) + 1)} \\ &= \frac{C_L(s_{opt}(x), x) + 1}{C_L(s, x) + 1} \geq \frac{1}{2} R_L^{loc}(s, x). \end{aligned}$$

This proves the lemma.

Now we reduce Q to FLIP. An instance x of Q is mapped to a circuit that computes C_Q. This defines the instance mapping f. For every s: $g(s, x) = s$. This is a LPR-reduction, because $f(x)$ exactly simulates x. $Q \in \mathcal{NC}^k\mathcal{LS}$ implies that this is a reduction to $\mathcal{NC}^k$-FLIP. $Q \in \mathcal{NC}^k\mathcal{LS}^{pol}$ implies that this is a reduction to $\mathcal{NC}^k$-FLIP_{log}. □

Theorem 2.17 *1. Local $n^{k/2}$-approximation of $\mathcal{NC}^1$-$\text{FLIP}_{k \cdot log}$ solves a $\mathcal{P}$-hard problem.*

2. Local $2^{\epsilon \cdot n}$-approximation of $\mathcal{NC}^1$-FLIP solves a $\mathcal{PLS}$-hard problem (for an $\epsilon > 0$).

3. Local n^{ϵ}-approximation of a problem that is $\mathcal{NC}^1\mathcal{LS}^{pol}$-hard via $\mathcal{NC}$-LPR-reductions solves a $\mathcal{P}$-hard problem (for an $\epsilon > 0$).

4. Local $2^{n^{\epsilon}}$-approximation of a problem that is $\mathcal{NC}^1\mathcal{LS}$-hard via $\mathcal{P}$-LPR-reductions solves a $\mathcal{PLS}$-hard problem (for an $\epsilon > 0$).

PROOF:

1. Fact 2.9 and theorem 2.13 imply that finding local optima for $\mathcal{NC}^1$-FLIP$_{k \cdot log}$ solves a $\mathcal{P}$-hard problem. We show how a local $n^{k/2}$-approximation can be used to determine local optima exactly.

   A given logarithmic depth circuit C of size n with $k \cdot \log n$ outputs can be mapped to a circuit C' that computes the output of C on an input and on all neighbors of this input. After this C' determines whether its input is locally optimal or not. If the input is not locally optimal, then C' outputs the cost of the input as computed by C. If the input is locally optimal, then C' increases its cost to n^{2k} (C' has twice as many outputs as C).

   The test whether one number is the maximum of n numbers with n bits (here we have only $O(\log n)$ bits) can be done in logarithmic depth and size $O(n^2)$ by n parallel standard algorithms for comparison (see [We87]). Thus the size of C' is $O(n^2)$, the depth $O(\log n)$.

   The construction yields an instance of $\mathcal{NC}^1$-FLIP$_{k \cdot log}$, because the size of C' is $O(n^2)$ and C' has $2k \log n = k \log(n^2)$ outputs. A $(n^2)^{k/2}$-approximation finds a solution that is locally optimal for C, because local optima of C' have cost n^{2k} and a n^k-approximation must find a solution of cost at least n^k. Only local optima of C' (and thus of C) have such large cost. Relative to the circuit size m of C' the approximation has a local performance ratio of $m^{k/2}$.

2. The same construction as in 1) applied to a logarithmic depth circuit of size n yields a logarithmic depth circuit C' of size $O(n^2)$ again. This time it is possible to use n^2 outputs. Local optima are increased to cost 2^{n^2}. Other solutions have cost at most 2^n. The size m of C' is $O(n^2)$, the depth logarithmic, a $2^{\epsilon \cdot m}$-approximation is sufficient to find local optima (for some $\epsilon > 0$).

3. When a problem L that is $\mathcal{NC}^1\mathcal{LS}^{pol}$-hard via $\mathcal{NC}$-LPR-reductions can be approximated within n^ϵ, then $\mathcal{NC}^1$-FLIP$_{k \cdot log}$ can be reduced to L with size amplification $n^{k'}$ (for some k' possibly depending on k), and can thus be approximated locally within $O(n^{k' \cdot \epsilon})$ due to theorem 2.14. If $\epsilon \leq k/(2k')$ then this yields a $n^{k/2}$-approximation of $\mathcal{NC}^1$FLIP$_{k \log}$ and solves a $\mathcal{P}$-hard problem due to part 1.

4. If a problem L that is $\mathcal{NC}^1\mathcal{LS}$-hard via $\mathcal{NC}$-LPR-reductions can be approximated within 2^{n^ϵ}, then $\mathcal{NC}^1$-FLIP can be reduced to L with some size amplification n^k and can thus be approximated locally within $O(2^{(n^k)^\epsilon})$ due to theorem 2.14. This solves a $\mathcal{PLS}$-hard problem for $\epsilon < 1/k$ due to part 2. □

Now we know that complete problems under LPR-reductions exist and have no considerably more efficient local approximation than local optimization algorithms. In section 3.3 we will encounter another problem that is complete for $\mathcal{PLS}$ via $\mathcal{P}$-LPR-reduction.

3 Investigating the Hopfield Energy Function

Remember the definition of Hopfield nets given in section 2.1. The goal of a Hopfield net is to reach a stable configuration, i.e., a state s in which for any neuron v_i and its state s_i the following holds: $s_i = 1 \iff \sum_{j \neq i} w_{i,j} s_j \geq t_i$. Now consider the following quadratic expression:

$$H[s] = \sum_{i<j} w_{i,j} s_i s_j - \sum_i t_i s_i.$$

$w_{i,j} = w_{j,i}$ is the symmetric connection strength between v_i and v_j, and t_i is the threshold of unit v_i. The first sum counts every undirected edge once. We now require that for neuron v_i the sum $\sum_{j \neq i} w_{i,j} s_j$ never equals t_i. This can easily be fulfilled by choosing t_i to be .5 larger than an integer and all weights as integers. If neuron v_k computes its threshold function $\theta_r(w_{k,1}, \ldots, w_{k,r}, t_k)$ and changes its state, then the following holds:

$$s_k^{new} \cdot (\sum_{i \neq k} w_{k,i} s_i - t_k) > s_k^{old} \cdot (\sum_{i \neq k} w_{k,i} s_i - t_k).$$

This implies that

$$\begin{aligned}
H[s^{new}] &= \sum_{i<j} w_{i,j} s_i s_j - \sum_i t_i s_i \\
&= \frac{1}{2} \sum_i \sum_{j \neq i} w_{i,j} s_i s_j - \sum_i t_i s_i \\
&= \sum_i s_i (\frac{1}{2} \sum_{j \neq i} w_{i,j} s_j - t_i) \\
&= \sum_{i \neq k} s_i (\frac{1}{2} \sum_{j \neq i,k} w_{i,j} s_j - t_i) + s_k^{new} (\sum_{i \neq k} w_{k,i} s_i - t_k) \\
&> \sum_{i \neq k} s_i (\frac{1}{2} \sum_{j \neq i,k} w_{i,j} s_j - t_i) + s_k^{old} (\sum_{i \neq k} w_{k,i} s_i - t_k) \\
&= H[s^{old}]
\end{aligned}$$

Definition 3.1 *The function*

$$H[s] = \sum_{i<j} w_{i,j} s_i s_j - \sum_i t_i s_i$$

is called the "Hopfield energy function" for the Hopfield net (V, E, ℓ) *with* $w_{i,j} = \ell(\{v_i, v_j\})$ *for* $\{v_i, v_j\} \in E$ *and* $w_{i,j} = 0$ *for* $\{v_i, v_j\} \notin E$ *and* $t_i = \ell(v_i)$.

Theorem 3.1 *A Hopfield net is in a stable state iff its energy function is in a local maximum. Each computation of a neuron that changes its state increases the value of the energy function.*

Hopfield nets and the energy function are usually defined differently in two aspects. First the energy function is often multiplied by -1 in order to let energy decrease (Hopfield nets were first introduced by a physicist who did probably not like the idea of a system stabilizing itself at maximum energy). We leave the negation out and view the energy function as a maximization problem.

The other difference concerns the domain of the state variables. Usually $\{-1,1\}$ is preferred. This makes the application for associative memory slightly easier. Clearly a translation between $\{-1,1\}$ and $\{0,1\}$ is easy and both energy functions have the same optima. The difference is that the translation introduces constants into the function which can be omitted if one is only interested in exact optimization. If one wants to approximate local optima these constants however become important. We are not aware of any mechanism translating our lower bounds for the complexity of approximation over $\{0,1\}$ to the domain $\{-1,1\}$, at least not if we forbid constants in the energy function, i.e., self-loops (since $s_i s_i = 1$) in the net. All results for the $\{0,1\}$-Hopfield function are valid for $\{-1,1\}$-nets with self loops.

But self loops have brutal effects: it is easy to show that the polynomially weighted Hopfield function over $\{-1,1\}$ with self loops and only negative weights cannot be n^k-approximated locally in $\mathcal{NC}$ (unless $\mathcal{P} = \mathcal{NC}$). A slight modification of the $\mathcal{P}$-completeness proof for the problem to find local optima of MAX CUT in [SchY91] can induce a tiny gap between "accepting" and "rejecting" local optima (by adding 1 for accepting). Then the overall energy can be decreased to 0 resp. 1 by a negatively weighted self-loop. Now all weights can be multiplied by a polynomially large number which increases the gap between accepting and rejecting. But this method can clearly not be used if self-loops are forbidden—and there is no reason to allow them since they carry no information. For other effects of self-loops see [Par94].

The reason why we chose the domain $\{0,1\}$ is that the main interest in Hopfield nets stems from their abilities in combinatorial optimization. This domain seems to allow more comfortable ways of encoding optimization problems. Both possibilities offer the same profile concerning local optima, but whereas the $\{0,1\}$ domain allows to encode problems in a way that the quality of approximations is preserved, this seems to be more difficult on the domain $\{-1,1\}$ (examples will be given in section 3.2).

Another aspect of Hopfield nets is the size and the sign of the weights. The Hopfield energy function will be examined with several restrictions: first with weights of unit size, polynomial size, unbounded (exponential) size. This distinction is important (making a big difference to acyclic nets, see [Gol92]). Secondly the sign of weights will be restricted: positive weights, negative weights, and both. This has an important effect on the complexity and the power to express problems, too. Thresholds will not be restricted since it is easy to see that the only difference made by such restrictions is that the problem can be trivialized (e.g. a net with negative weights and positive thresholds that has only the vector of zeroes as local optimum). The different versions of the Hopfield function will be called P-HOPFIELD$^{|1|}$, N-HOPFIELD^{pol}, PN-HOPFIELD^{exp} and so on.

The results of this section can be found in the following table.

MAX $\{0,1\}$-HOPFIELD $\sum_{i<j} w_{i,j}s_is_j - \sum_i t_is_i$		Positive Weights	Negative Weights	Pos./Neg. Weights
$\|1\|$-weights	Opt.	$\mathcal{RTC}^1$	$\mathcal{NP}$-cpl.	$\mathcal{NP}$-cpl.
Global	App.	n^ϵ-App. $\mathcal{NL}$-hard	n^ϵ-App. $\mathcal{NP}$-cpl.	n^ϵ-App. $\mathcal{NP}$-cpl.
$\|1\|$-weights	Opt.	$\mathcal{RTC}^1$	$\mathcal{P}$-cpl.	$\mathcal{P}$-cpl.
Local	App.	n^ϵ-App. $\notin \mathcal{AC}^0$	?	n^ϵ-App. $\notin \mathcal{AC}^0$
pol.-weights	Opt.	$\mathcal{RTC}^1$	$\mathcal{NP}$-cpl.	$\mathcal{NP}$-cpl.
Global	App.	n^k-App. $\mathcal{NL}$-hard	n^ϵ-App. $\mathcal{NP}$-cpl.	n^ϵ-App. $\mathcal{NP}$-cpl.
pol.-weights	Opt.	$\mathcal{RTC}^1$	$\mathcal{P}$-cpl.	$\mathcal{P}$-cpl.
Local	App.	n^k-App. $\mathcal{L}$-hard	?	n^k-App. $\mathcal{P}$-cpl.
exp.-weights	Opt.	$\mathcal{P}$-cpl.	$\mathcal{NP}$-cpl.	$\mathcal{NP}$-cpl.
Global	App.	2^n-App. $\mathcal{NL}$-hard	n^ϵ-App. $\mathcal{NP}$-cpl.	2^{n^ϵ}-App. $\mathcal{NP}$-cpl.
exp.-weights	Opt.	$\mathcal{P}$-cpl.	$\mathcal{PLS}$-cpl.	$\mathcal{PLS}$-cpl.
Local	App.	2^n-App. $\mathcal{NL}$-hard	?	2^{n^ϵ}-App. $\mathcal{PLS}$-cpl.

n denotes the number of vertices of the Hopfield net, ϵ a positive constant, and k an arbitrarily large positive constant depending on the polynomial bound attached to the weights. "Opt." denotes the complexity of computing optima, "App." of approximation. $\mathcal{L}$ and $\mathcal{NL}$ abbreviate $\mathcal{LOGSPACE}$ resp. $\mathcal{NLOGSPACE}$. The name of a complexity class stands for an algorithm solving a problem with the complexity of the functional equivalent of this class, "cpl." means that an additional hardness result exists, "hard" indicates a hardness result.

The most interesting open problems concerning this table correspond to the question marks. We are not aware of lower or upper bounds for the approximation of local optima for the negative weight Hopfield function, although we conjecture that it can be approximated in functional $\mathcal{NC}$ resp. in polynomial time because it seems to be very hard to code even trivial decision problems using N-HOPFIELD.

The following subsections prove the propositions in the table. Note that the third column of the table "inherits" lower bounds from the other two. Moreover it is clear that local optimization is at most as hard as global optimization.

3.1 The Positive Weight Hopfield Function

In this subsection the Hopfield energy function restricted to positive weights is investigated. This restriction implies that every neuron computes a monotone threshold function. Local search has a monotone property, too: it is possible to perform local search by starting from the vector of zeroes and changing states of neurons (correctly) from '0' to '1' until a local optimum is reached. Note that positive contributions to the energy function come from edge weights and negative thresholds, whereas positive thresholds contribute negatively to the energy function.

Units with negative thresholds are in state '1' in any local optimum. A net with only nonnegative thresholds has a trivial local optimum: the vector of zeroes. Thus units with negative thresholds are necessary for nontrivial local optimization. The size of the negative threshold on the other hand is not so important: it only adds a constant to the energy of all local optima. We will call units with threshold -1 "starting buttons", because they force the net to start a desired computation. We say that a unit is "turned on" when it changes its state from '0' to '1', in the opposite case it is "turned off".

Now we establish a strong connection of the positive weight Hopfield energy function to the s,t-MIN CUT problem and deduce the complexity of global optimization from this connection (note that s,t-MIN CUT is defined on directed graphs and minimizes the weights of edges leading from the side of a vertex subset s (here defined as '1'), to the side of a vertex subset t (here defined as '0')).

Theorem 3.2 *1. The s,t-MIN CUT problem with positive weights can be solved by a positively weighted Hopfield net, i.e., s,t-MIN CUT$\leq_{\mathcal{LCS}}$ P-HOPFIELD.*

2. P-HOPFIELD$\leq_{\mathcal{LCS}}$ *s,t-MIN CUT.*

PROOF:

1. Let G be any directed graph with weighted edges, and let s,t be two vertex subsets that must be separated by a feasible cut. First neglect s and t and consider the following expression for the minimization of the cut (y denotes the sides of the cut):

$$\begin{aligned}
\min CUT(G) &= \min \sum_{(i,j)} w_{(i,j)} y_i (1 - y_j) \\
&= \max \sum_{(i,j)} w_{(i,j)} (y_i y_j - y_i) \\
&= \max \sum_{(i,j)} (w_{(i,j)} y_i y_j - w_{(i,j)} y_i) \\
&= \max \sum_{i<j} (w_{(i,j)} + w_{(j,i)}) y_i y_j - \sum_i \left(\sum_{j:(i,j)\in E} w_{(i,j)} \right) y_i .
\end{aligned}$$

Finding a minimum cut is equivalent to maximizing the Hopfield energy function on a net that has the same graph structure (though undirected edges), as edge weights $w_{i,j} = w_{(i,j)} + w_{(j,i)}$, and as thresholds the summed weights on edges leaving a vertex (observe that computing the sum of weights requires $\mathcal{LOGSPACE}$ reductions). But since all the thresholds are positive the vector of zeroes is locally optimal and we have neglected s and t so far. Assign threshold -1 to all vertices in s and threshold $\sum_j w_{(i,j)} + 1$ to vertex v_i in t. Now in every local optimum of the Hopfield net all vertices in s have state '1', all vertices in t have state '0'. All other vertices behave like the MIN CUT problem demands. The Hopfield net and the s,t-MIN CUT instance have the same locally (and globally) optimal solutions. Unfortunately the reduction does not preserve approximability.

2. Take any positive weight Hopfield energy function H. We will implement the corresponding net on an undirected s,t-MIN CUT instance. Note that a Hopfield net with threshold $\sum_{j\neq i} w_{i,j}/2$ for unit v_i corresponds to undirected MIN CUT: undirected graphs are equivalent to directed graphs with edges from v_i to v_j and from v_j to v_i instead of an undirected edge between v_i and v_j. The noted connection between undirected MIN CUT and the positive weight Hopfield function follows from the equations in 1.

   We introduce two new vertices v^1 and v^0 into the net H that will be forced to belong to different sides of a feasible cut (by the s,t condition). The side of v^1 defines the state '1', the side of v^0 the state '0'. These vertices will be used to turn thresholds into edges resulting in a s,t-MIN CUT instance.

   If a vertex v_i of the Hopfield net H has threshold $t_i < \sum_{j\neq i} w_{i,j}/2$, then connect v_i to v^1 with weight $\sum_{j\neq i} w_{i,j} - 2t_i$ and change the threshold to $\sum_{j\neq i} w_{i,j} - t_i$. If v^1 is in state '1' then this does not change the computation of v_i, because

$$input^{new} - t_i^{new} = input + (\sum_{j\neq i} w_{i,j} - 2t_i) - (\sum_{j\neq i} w_{i,j} - t_i) = input - t_i.$$

   Now the threshold t_i^{new} is exactly one half of the sum of edges incident to v_i.

   If a vertex v_i of the Hopfield net H has threshold $t_i > \sum_{j\neq i} w_{i,j}/2$, then connect v_i to v^0 with weight $2t_i - \sum_{j\neq i} w_{i,j}$ and leave the threshold unchanged. If v^0 is in state '0', then this connection adds nothing to the input of v_i, but makes the threshold t_i one half of the sum of edges incident to v_i.

   The computation of H is unaltered, but all units act as in MIN CUT, except of v^0 and v^1. Now remove the thresholds, take the weighted graph, use $s = \{v^1\}$ and $t = \{v^0\}$. These two vertices are on different sides in any feasible cut. Now finding a locally (or globally) optimal s,t-MIN CUT determines a locally (resp. globally) optimal solution to the P-HOPFIELD instance H, where the side of $s = \{v^1\}$ defines the units in state '1', the side of $t = \{v^0\}$ defines the units in state '0'. □

Now we know that local (or global) optimization of P-HOPFIELD is as hard as optimization of s,t-MIN CUT. This establishes the following:

Theorem 3.3 *1. Finding a global optimum of* P-HOPFIELD$^{|1|}$ *is possible in functional* $\mathcal{RTC}^1$.

2. Finding a global optimum of P-HOPFIELD^{pol} *is possible in functional* $\mathcal{RTC}^1$.

3. Finding a global optimum of P-HOPFIELD^{exp} *is possible in polynomial time, and solves a* $\mathcal{P}$*-complete problem.*

PROOF: In the proof of theorem 3.2 it was shown that finding global optima for P-HOPFIELD is possible by finding global optima for s,t-MIN CUT (using a $\mathcal{LOGSPACE}$-reduction). There is a well-known connection between s,t-MIN CUT and s,t-MAX FLOW: both have the same instances, and global optima of the two problems have the same value on the same instance . Moreover, given a globally optimal solution to a s,t-MAX FLOW instance, one can obtain a globally optimal feasible minimum cut by breadth first search (see [VL90]), and thus in $\mathcal{AC}^1$ (see [KarRa90]).

It is possible to find a globally optimal s,t-MAX FLOW in functional $\mathcal{RTC}^1$ (see section 4.1) in the case of polynomial weights. Thus the first two statements of the theorem follow. In the case of exponential weights global optima of s,t-MAX FLOW (and thus of P-HOPFIELD) can be found in polynomial time (see [VL90]).

Determining the value of the global optimum for unbounded weight s,t-MAX FLOW is known to solve a $\mathcal{P}$-complete problem (see [KarRa90]). The second part of the third statement holds, because this value can be found by global optimization of P-HOPFIELD^{exp} (and with this of s,t-MIN CUT). □

It is unknown whether locally optimal solutions may be found faster or without the randomization in the case of polynomial or unit size weights. In the case of exponential weights this task is not easier.

Theorem 3.4 *Finding a local optimum of the Hopfield function with positive exponential weights solves a* $\mathcal{P}$*-complete problem.*

PROOF: The MONOTONE CIRCUIT VALUE PROBLEM (see [J90]) is $\mathcal{P}$-complete. Given a monotone circuit C of size m with fan-in 2, fan-out 2, and an input s, build a Hopfield net H consisting of m units simulating the m gates. The internal gates of the circuit are assumed to be given in topological order.

Let $W = 4^m$. The units for the input gates of the circuit have threshold -1 if their input bit is 1, W otherwise. The unit for internal gate i has threshold $2W/4^i - 1$ for an AND-gate and $W/4^i - 1$ for an OR-gate. The graph structure is the same as the graph of the circuit (directed edges are replaced by undirected edges). Edges leading into unit i are weighted with $W/4^i$.

Now local search is forced to follow the computation from the inputs to the outputs: changing unit i to the correct value of gate i gains more than it can lose on the units with higher numbers.

Consider any situation supposed to be a local optimum. Units simulating inputs clearly have the correct state, because a -1 threshold allows to turn a unit on, whereas a W threshold cannot be exceeded and units with this threshold are off.

Now to the units simulating the internal gates of the circuit. A "predecessor" of unit i is a unit simulating a predecessor of gate i in C, a "successor" of unit i is a unit simulating a successor of gate i in C. A wrong state of a unit simulating an OR-gate can be corrected by local search, because with one predecessor in state '1' the threshold of the unit is exceeded while with two successors in state '1' the threshold is not exceeded. A wrong state of a unit simulating an AND-gate can be corrected by local search, because with two predecessors in state '1' the threshold is exceeded and with one predecessor together with two successors in state '1' the threshold is not exceeded. Thus in a local optimum all units have the correct values of the corresponding gates and the $\mathcal{P}$-complete MONOTONE CIRCUIT VALUE PROBLEM is solved, i.e., the circuit output can be read off from the locally optimal state of the Hopfield net. □

It is possible that global optimization has the same complexity as local optimization in the case of polynomial weights, too. This is not immediately clear since it is easy to construct nets where local optima do not even approximate global optima. Thus local optimization could be easier. But the following lower bounds for local approximation are rather tight in the case of polynomial weights (theorem 3.7), so that one can say that local optimization (which is at least as hard as local approximation) is practically as hard as global optimization for P-HOPFIELD.

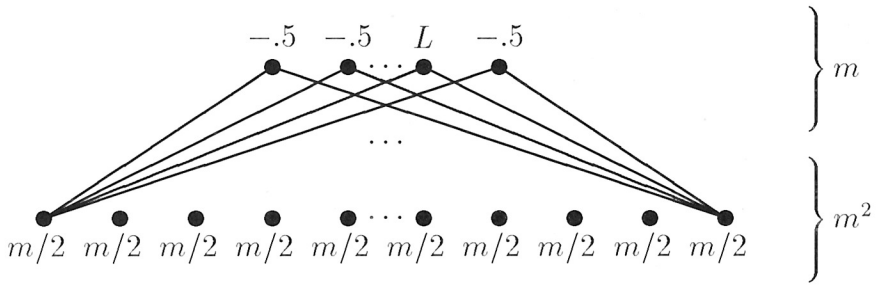
Now to lower bounds for local approximation on the three sizes of weights.

Theorem 3.5 *The Hopfield function with positive weights of unit size cannot be approximated locally within $\sqrt{n}/5$ in functional $\mathcal{AC}^0$.*

PROOF: The MAJORITY function (which is true on m bits iff at least $m/2$ of them are 1) cannot be computed by an unbounded fan-in circuit family of polynomial size and constant depth made of AND, OR, and NOT gates (see [We87], p.333), even if this family is nonuniform. We will show how to construct positive weight Hopfield nets with a large gap in the energy function between local optima that "accept" a string for MAJORITY, and those that "reject". This gap stays quite large after approximation. It is impossible, however, to evaluate the energy function in $\mathcal{AC}^0$ to decide MAJORITY.

A probabilistic constant depth circuit family can separate, given approximating states of the Hopfield nets, "accepting" from "rejecting". The probabilistic circuit family leads to a deterministic nonuniform circuit family for MAJORITY, if a $\mathcal{AC}^0$ family, or more generally, a family of polynomial size, constant depth AND/OR/NOT circuits, is able to approximate P-HOPFIELD$^{[1]}$ within $\sqrt{n}/5$. This leads to a contradiction with the lower bound for MAJORITY.

Let x be a string of length m. The Hopfield net will be called H_m. For each appearance of a 1 in x there is a unit with threshold -0.5, for each appearance of a 0 a unit with threshold $L = m^4$. These units are not interconnected, but they are completely bipartite connected to m^2 (not interconnected) units with threshold $m/2$ if m is not divisible by



two, else with threshold $(m-1)/2$. These units will be referred to as "counters". Edge weights are 1.

Now in a local optimum clearly all units of the first class (representing the input x) that correspond to ones in x are on, those that correspond to zeroes are off. If at least $m/2$ units of the first class are on, then the counters are on, too. If less than $m/2$ units of the first class are on, then the counters are off. Note that the local optimum is unique in both cases and thus also globally optimal.

A solution approximating the unique local optimum within performance ratio $m/3$ clearly never contains a unit with threshold L in state '1'. If $x \notin$MAJORITY, then such a solution can contain at most m counters in state '1', because each counter that is on loses at least .5, and more than m of them in state '1' would imply a larger performance ratio than $m/3$.

If $x \in$MAJORITY, then turning on a counter gains at least .5, so all the (edges to) counters gain at least $.5m^2$, while the other units gain at most $.5m$. Thus at most a fraction of $1/m$ of the overall energy is gained by the thresholds of the units that code the string. A $m/3$-approximating solution must have at least $3/m$ of the optimal energy, at least $2/m$ of this energy must be gained by (edges to) counters. This is at least the gain of (edges to) $m^2 \cdot 2/m = 2m$ counters. At least $2m$ counters must be on.

Consider the following probabilistic algorithm $\mathcal{A}$ for MAJORITY:

- Construct the Hopfield net H_m.
- Find a $m/3$-approximating solution to H_m.
- Choose $m/2$ counter values $c_1, \ldots, c_{m/2}$ independently.
- Output $\mathrm{OR}(c_1, \ldots, c_{m/2})$.

This probabilistic algorithm can be performed by a probabilistic constant depth, polynomial size circuit family made of AND, OR, and NOT gates if a circuit family with the same limitations exists that performs step 2. The probabilistic circuit family can be turned into a deterministic circuit family with the following fact (from [We87], p.356).

Fact 3.6 *Let C be a probabilistic circuit with arbitrary fan-in, size r, input size n, depth d, and the following connection to a decision f on inputs x:*

$$prob[C(x) = 1 | f(x) = 1] \geq \frac{1}{2} + \frac{1}{\log(n)} \quad \wedge \quad prob[C(x) = 1 | f(x) = 0] \leq \frac{1}{2}.$$

Then a deterministic circuit of size $O(n^6 \log^2(n) r)$ and depth $O(d)$ exists that computes f. The construction does not preserve uniformity.

Consider a circuit of the family $\mathcal{A}$. Let c_i be the random variable that is 1 when the ith chosen counter is on, and 0 otherwise. Then the probability that the circuit outputs 1 instead of 0 ($\leq m$ counters are on) is

$$\text{prob}[\bigvee_i c_i = 1] \leq \sum_i \text{prob}[c_i = 1] = m/2 \cdot \frac{m}{m^2} = \frac{1}{2}$$

The probability that the circuit computes 0 instead of 1 ($\geq 2m$ counters are on) is

$$\begin{aligned} \text{prob}[\textstyle\bigvee_i c_i = 0] &= \prod_i \text{prob}[c_i = 0] \leq \left(\frac{m^2 - 2m}{m^2}\right)^{m/2} \\ &= \left(\frac{m-2}{m}\right)^{m/2} < \left(\frac{m-1}{m}\right)^{m} \\ &< \frac{1}{e} < 0.37 \end{aligned}$$

Thus the probability of correct acceptance is larger than 0.63. The conditions to fact 3.6 are satisfied, and thus a polynomial size, constant depth, unbounded fan-in circuit family made of AND, OR, NOT gates exists that decides MAJORITY if such a circuit family is able to approximate P-HOPFIELD. This contradicts the lower bound we referred to. The net H_m has size $n = m^2 + m$, a local $\sqrt{n}/5 \leq \sqrt{2m^2}/5 < m/3$-aproximation suffices. □

Theorem 3.7 *A local n^k-approximation of the Hopfield function with positive weights of polynomial size solves a $\mathcal{LOGSPACE}$-hard problem, when weights of size n^{k+1} are allowed.*

PROOF: Take any language L in $\mathcal{LOGSPACE}$. Then a Turingmachine M deciding L exists that has the following properties:

1. M uses $O(\log n)$ workspace.
2. M has one unique accepting configuration acc.
3. Every configuration of M has exactly one successor, acc is its own successor.

The second property may easily be fulfilled by forcing the machine to clear its tape and move to a distinguished tape position before halting, resp. starting the accepting loop.

In the following a Hopfield net with polynomially bounded size and weights is constructed that simulates the computation of M on an input string x. The first step is the construction of a computation graph of M on x.

Lemma 3.8 *A word x can be mapped to a directed graph $G_{M,x}$ of polynomial size with outdegree 1 and two distinguished vertices* start *and* acc*, such that a computation of M on x corresponds to a path beginning at* start*. This path reaches* acc *iff $x \in L$. The mapping is computable in functional $\mathcal{AC}^0$.*

Take a vertex for every configuration of M, i.e., every state of M and its workspace. The set of vertices has size m polynomial in $|x|$, because the workspace is logarithmically bounded (in $|x|$), and the tape-alphabet as well as the set of states of M has constant size. Now connect every ordered pair of vertices if the configuration corresponding to the second vertex is the successor of the configuration corresponding to the first vertex. This can be done in parallel and in constant time. start is the vertex of the start configuration (on x), acc of the unique accepting configuration. Every vertex has outdegree 1, because M is deterministic.

Clearly the path in $G_{M,x}$ beginning at start corresponds to the computation of M on x. If acc is reached, then it is reached within m steps, because otherwise the machine gets into an infinite loop earlier and cannot reach acc anyway.

Now the graph $G_{M,x}$ is mapped to a Hopfield net that is able to find paths in directed graphs with the property that every vertex has outdegree ≤ 1.

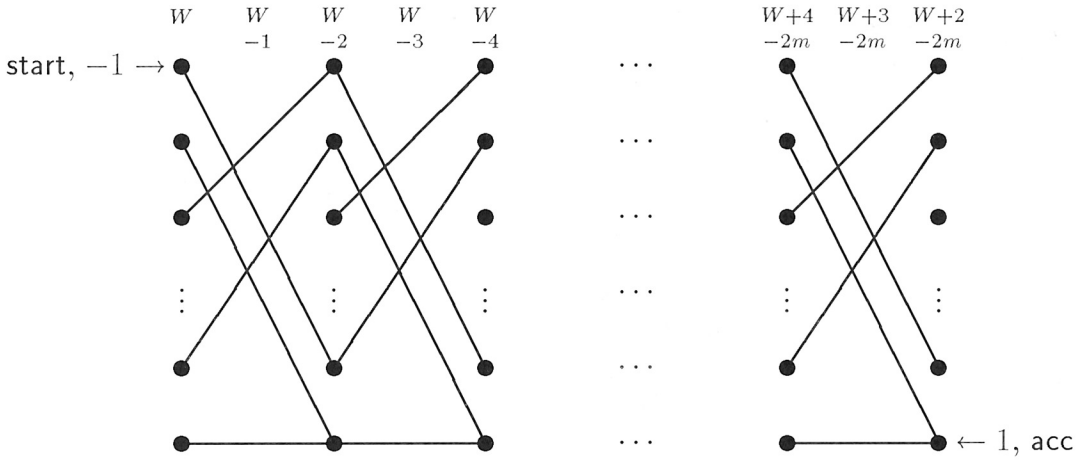
Lemma 3.9 *$G_{M,x}$ can be mapped to a Hopfield net $H_{M,x}$ of polynomial size with positive polynomial weights, that possesses only one local optimum. This optimum corresponds to the path in $G_{M,x}$ beginning at* start*. The mapping is computable in functional $\mathcal{AC}^0$.*

Let $W = (m^2)^{k+1}$ ($k > 0$ is a constant). $H_{M,x}$ is described by a matrix of m rows and m columns. Each column is a copy of the vertices of $G_{M,x}$. The directed edges of $G_{M,x}$ are replaced by undirected edges between neighboring columns. Formally the set of units is $V = \{v_{i,j} | 0 \leq i,j \leq m-1\}$, the edge set is $E = \{(v_{i,j}, v_{i+1,k}) | (v_j, v_k)$ is an edge in $G_{M,x}\}$.

Each column is going to represent one time step in the computation of M on x. In the 0th column we want the copy of start to be on, in the ith column the copy of the ith successor configuration of start.

All units in column i have threshold $W - 2i$, all edges between column i and $i+1$ have weight $W - 2i - 1$. A first exception is the unit $v_{0,\mathsf{start}}$ corresponding to the start configuration in column 0, which has threshold -1. This unit will be the starting button of the net. The second exception is the unit $v_{m-1,\mathsf{acc}}$, the mth copy of the accepting configuration, which has threshold 1.

Now consider the local optima of $H_{M,x}$. Obviously the set of units in state '1' in any configuration can be partitioned into a set of trees (since the whole graph has this property, too). The weight of any edge $(v_{i,j}, v_{i+1,k})$ is larger than the threshold of $v_{i+1,k}$, but smaller than the threshold of $v_{i,j}$. Thus if $v_{i,j}$ has no neighbor in column $i-1$ which is on, then turning $v_{i,j}$ off gains the threshold of $v_{i,j}$ and loses the weight of the single edge from $v_{i,j}$ to column $i+1$. This operation clearly gains. Turning on $v_{i+1,k}$ if $v_{i,j}$ is on clearly gains, too, because the gained edge has larger weight than the threshold of $v_{i+1,k}$.



In this way a path of vertices in state '1' in $H_{M,x}$ can be "extended" to the right by turning right neighbors on, and can be "deleted" from the left by turning its leftmost unit off. This holds for every path except the one starting at $v_{0,\mathsf{start}}$, which is the only unit that does not cause costs (the starting button).

To conclude: In column 0 only $v_{0,\mathsf{start}}$ is on, and only the units in the unique path beginning at $v_{0,\mathsf{start}}$ and leading to the right through all columns are on.

Lemma 3.10 *$H_{M,x}$ has a unique local optimum that corresponds to the path in $G_{M,x}$ beginning at* start. *If this path reaches* acc, *then the optimum has energy larger than m^{2k+1}, else it has energy m. Any state of $H_{M,x}$ that is not a path from* start *to* acc *has energy at most m.*

Only states of the Hopfield net, where all units in state '1' form a path beginning at $v_{0,\mathsf{start}}$ and leading to the right, have positive energy. Every other state loses more with thresholds than it gains with edges and has thus negative energy. A state of $H_{m,x}$ corresponding to a path that begins at $v_{0,\mathsf{start}}$ has energy equal to the path's *length* + 1. Only if the path reaches $v_{m-1,\mathsf{acc}}$ an additional gain is made: The edge gains $W - 2m + 3$, the unit costs 1. Thus an "accepting path"-state has energy

$$(m-1) + (W - 2m + 3) - 1 = W - m + 1 > m^{2k+1}.$$

Any other state has energy at most m, because it loses at least as many thresholds as it gains edges.

Thus a local optimum of $H_{M,x}$ has energy larger than m^{2k+1} if $x \in L$, and energy at most m if $x \notin L$. A $n^k = m^{2k}$-approximation ($n = m^2$ is the size of the Hopfield net) of a local optimum of the Hopfield energy function with positive polynomial weights is sufficient to find out whether $x \in L$ or not. In the case of acceptance the approximating

state must already be the local optimum, because this is the only state with enough energy to approximate within the demanded local performance ratio. Such an approximation solves a $\mathcal{LOGSPACE}$-hard problem, because given an approximating state-vector one can easily test whether it represents a path from $v_{0,\mathsf{start}}$ to $v_{m-1,\mathsf{acc}}$. □

The previous result shows that local approximation cannot be done below $\mathcal{AC}^1$ (assuming the hierarchy of complexity classes inside $\mathcal{NC}$ shown in fact 2.4 is strict), whereas a global optimum can be constructed in $\mathcal{RTC}^1$. These bounds are very close. An interesting open question is whether a $\mathcal{NC}$ algorithm may find or approximate local optima.

If the weights are unbounded the lower bound can be improved slightly.

Theorem 3.11 *A local 2^n-approximation of the Hopfield energy function with positive weights of exponential size solves a $\mathcal{NLOGSPACE}$-hard problem.*

PROOF: Let L be any language in $\mathcal{NLOGSPACE}$. Then a nondeterministic Turing-machine accepting L exists such that the following holds:

1. M uses logarithmically bounded workspace.
2. M has a unique accepting configuration.

Lemma 3.12 *Given a string x a digraph $G_{M,x}$ can be constructed in $\mathcal{AC}^0$ where $G_{M,x}$ has polynomial size, every vertex has indegree at most 2 and outdegree at most 2, and there are two distinguished vertices* start *and* acc*, such that $x \in L$ iff a path from* start *to* acc *exists.*

First take the configuration graph of M on input x. Every configuration has at most a constant number of predecessors and successors since the numbers of changes a Turing-machine can produce in one step is bounded. For the graph $G_{M,x}$ indegree and outdegree are reduced to 2 by inserting trees. The size of $G_{M,x}$ is polynomial in $|x|$ since the workspace of M is logarithmic in $|x|$. start is the vertex of the start configuration (on x), acc of the unique accepting configuration. If $x \in L$, then a path exists that begins at start and reaches acc within m steps. The problem to find such a path is called the GRAPH ACCESSIBILITY PROBLEM (see [J90]).

Lemma 3.13 *$G_{M,x}$ can be mapped to a Hopfield net $H_{M,x}$ of polynomial size that has positive weights and only one local optimum. This local optimum corresponds to all paths in $G_{M,x}$ that begin at* start*. Finding this optimum simulates the nondeterministic computation of M on x. The mapping is computable in functional $\mathcal{AC}^0$.*

Let $W = 4^{m^2}$, where m is the size of $G_{M,x}$. The Hopfield net is described by a matrix of m rows and m columns plus one additional unit. It has basically the same connections as the net of theorem 3.7. Every column is a copy of $G_{M,x}$. Edges connect neighboring columns only. Formally the set of units is $V = \{v_{i,j} | 0 \leq i,j \leq m-1\}$ and the edge set is $E = \{(v_{i,j}, v_{i+1,k}) | (v_j, v_k)$ is an edge in $G_{M,x}\}$. This time however we need one additional unit that is only connected to $v_{m-1,\mathsf{acc}}$.

The thresholds and weights are chosen somewhat differently than in the construction of theorem 3.7, because the net will compute in another way. A local optimum will not represent the successors of start, but the vertices that are *not* reachable from start. The columns will again correspond to the time steps of the computation of M on x. The unit $v_{i,j}$ will be on in a local optimum iff v_j in $G_{M,x}$ is not reachable from start in at most i steps. A vertex (except of start) is not reachable in at most i steps iff all of its predecessors are not reachable in at most $i-1$ steps. Thus $v_{i,j}$ must compute a logical AND on its neighbors in column $i-1$: if these are all on, then $v_{i,j}$ can be turned on.

The edges between column $i-1$ and i have weight $W/4^i$. The thresholds of the units in the first column are all -1 except of unit $v_{0,\mathsf{start}}$ that has threshold W. All other $v_{i,\mathsf{start}}$ have threshold W, too. All $v_{i,j}$, where v_j is different from start and $i > 0$ have the following thresholds: if v_j has indegree 0 in $G_{M,x}$, then $v_{i,j}$ has threshold -1. If v_j has indegree 1 in $G_{M,x}$, then $v_{i,j}$ has threshold $W/4^i - 1$. If v_j has indegree 2 in $G_{M,x}$, then $v_{i,j}$ has threshold $2W/4^i - 1$. The additional unit that is connected to $v_{m-1,\mathsf{acc}}$ will create the approximation gap. It has threshold 1 and its connection has weight $W/(4^m)$.

A local optimum of $H_{M,x}$ has the following properties: All units in column 0 are on except the one corresponding to start. Every "indegree 0-unit" in every column is on, every copy of start is off. Every unit in column i with all neighbors in the left neighbor column in state '1' is on. No unit with at least one of these neighbors in state '0' is on.

This holds since any unit may change its state independent of its right neighbors. If a indegree 2 unit in column i is on, but one of its left neighbors is not, then turning it off gains the threshold of $2W/4^i - 1$ and loses at most one edge to column $i-1$ and two edges to column $i+1$. The edges have weight

$$W/4^i + 2 \cdot W/4^{i+1} = (1 + 1/2) \cdot W/4^i < 2W/4^i - 1.$$

If a indegree 1 unit in column i is on, but its left neighbor is not, then turning it off gains the threshold of $W/4^i - 1$ and loses at most two edges to column $i+1$. The edges have weight

$$2W/4^{i+1} = 1/2 \cdot W/4^i < W/4^i - 1.$$

On the other hand turning on a unit with all left neighbors in state '1' gains 1.

So in a local optimum state '1' is assigned to all units $v_{i,j}$ such that configuration v_j is not reachable in $k \leq i$ steps, '0' to all others. The statement clearly holds for column 0 and all copies of start. We can assume it true for all columns up to $i-1$ and conclude for column i that all "indegree 0 units" are not reachable ('1'), all other units are not reachable in $\leq i$ steps iff their neighbors in column $i-1$ are all not reachable in $\leq i-1$ steps. This also implies that the local optimum is unique.

Lemma 3.14 *$H_{M,x}$ has a unique local optimum of energy $\Omega(2^{(2m^2-2m)})$ if $x \notin L$ and energy $O(2m^2)$ if $x \in L$.*

If $x \in L$ then a path exists from start to acc. Then the local optimum marks this path (and $v_{m-1,\mathsf{acc}}$) with zeroes. If $x \notin L$ then no such path exists and $v_{m-1,\mathsf{acc}}$ is not reachable and thus set to '1' as well as its neighbor in column m. Thus if $x \in L$ then the

energy is less than m^2 (a unit and its incident edges leading to the left contribute at most 1), otherwise at least (from the rightmost edge)

$$W/4^m = 4^{m^2-m} = 2^{2m^2-2m}.$$

Again a very poor local approximation of the Hopfield energy function solves the language problem: if the approximation has ratio $2^n = 2^{m^2+1}$ then $x \notin L$ implies energy larger than $2^{(m^2-2m-1)}$, $x \in L$ implies energy at most m^2. Any state that has exponentially large energy however must represent a correct computation in so far that no unit in the matrix is on that should not be on. To see this assume some unit would be on illegally. In the case of a unit with threshold W this would lose at least $W/2$. In the case of a unit with threshold $W/4^i$ this would lose at least $1/2 \cdot W/4^i - 1$. The unit in column m costs only 1, but gaining its edge requires to turn $v_{m-1,\mathsf{acc}}$ on and doing this wrongly is expensive.

Every state that has exponential energy contains $v_{m-1,\mathsf{acc}}$ and its right neighbor in state '1' (correctly), because the edge between them offers the only possibility to achieve that high energy. Every state that has no exponential energy does not contain $v_{m-1,\mathsf{acc}}$ in state '1'. This implies that testing a single bit finds whether the energy is exponential or not and thus decides the language problem. □

This result is interesting since $\mathcal{NLOGSPACE}$-hardness excludes a $\mathcal{NC}^1$ algorithm more securely than $\mathcal{LOGSPACE}$-hardness. Another aspect is that the nondeterminism of $\mathcal{NLOGSPACE}$ allows to exclude a very fast randomized algorithm, i.e., a $\mathcal{RNC}^1$ algorithm, because $\mathcal{RNC}^1$ is probably smaller then $\mathcal{NLOGSPACE}$.

An interesting observation is that the implicitly used $\mathcal{NLOGSPACE}$-complete problem is evaluation of a polynomial size circuit consisting of AND-gates only (which is the same as finding out whether an input 0 is connected to the output of a circuit). This problem can be implemented on a positive weight Hopfield net in a way such that the possible gain from turning on a unit is bounded and a gap can be produced that exceeds the gain of the whole computation, but is itself exceeded by the loss of any severe mistake. Conversely it seems to be difficult to express OR-Gates using only positive weights in a way such that the gain at every gate is bounded. The reason for the construction using a matrix is that it is impossible to find out the depth of a gate very fast.

To complete the picture global approximation is considered, too.

Theorem 3.15 *1. A global $n^{1/3}/2$-approximation of the Hopfield energy function with positive weights of unit size solves a $\mathcal{NLOGSPACE}$-hard problem.*

2. A global n^k-approximation of the Hopfield energy function with positive weights of polynomial size (bounded by n^{2k}) solves a $\mathcal{NLOGSPACE}$-hard problem.

3. A global 2^n-approximation of the Hopfield energy function with positive weights of exponential size solves a $\mathcal{NLOGSPACE}$-hard problem.

PROOF: It was shown in lemma 3.12 that the GRAPH ACCESSIBILITY PROBLEM is $\mathcal{NLOGSPACE}$-complete: given a directed acyclic graph G with n vertices, m edges, out-

and indegree at most 2, and given two vertices s, t, find out if a path from s to t exists. To solve this problem by global approximation of a positive weight Hopfield function we use a simulation of s,t-MIN CUT. Note that the globally optimal minimum cut has a value of 0 if no path from s to t exists, and a value larger than the least edge weight otherwise.

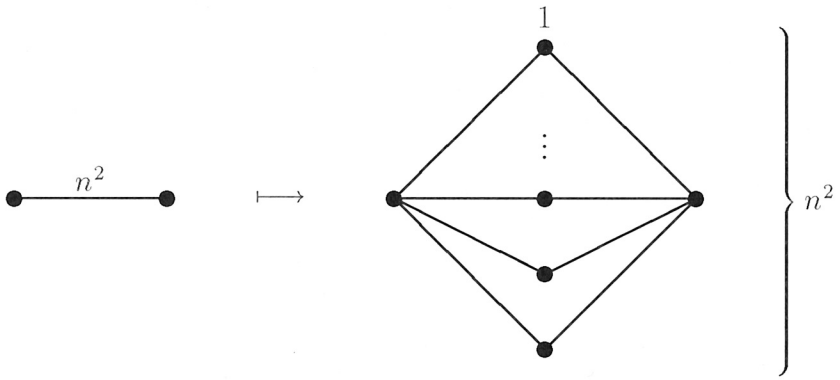
Let $W = n^k$ (for some constant $k \geq 1$) in the case of polynomial weights and $W = 2^n$ in the case of exponential weights. For the Hopfield net H take the graph G, change directed into undirected edges, weight all edges with W^2, assign threshold $outdegree_G(v_i) \cdot W^2$ to every vertex v_i, threshold $outdegree_G(s) \cdot W^2 - W$ to s, threshold W^3 to t. $outdegree_G(v)$ denotes the outdegree of v in G.

Now the vector of zeroes is clearly a local optimum. The thresholds are chosen almost as in the simulation of directed s,t-MIN CUT, where '0' and '1' define the sides of a cut. It is easy to see from the proof of theorem 3.2 that the energy of a state is the negative sum of those external edges that replace directed edges in G leading from the side of '1' to the side of '0'. An exception is made at s and t.

Turning s on does not cost the weights of all edges starting at s, but W less than that. Turning on t is very expensive, thus t is in state '0' in any optimum of H. The only possibility to achieve positive energy is to turn s on and gain W from its incident edges, since all other units have thresholds as large as the edges starting at them, and thus every gain from an edge is lost by thresholds.

If a path from s to t exists in G, then every state y of H has energy at most 0, because every cut in G separating s from t costs at least one edge, so that y has energy less than 0, and every other cut puts s on the side of t (in state '0'), so that no positive energy is possible. If no path from s to t exists in G, then a cut in G exists that separates the vertices in the transitive closure of s from t, and that costs no edge. This induces an optimum of H with energy W (gained at the edges incident to s).

In the case of unit weights take the previous construction for polynomial weights ($k = 1$) and replace edges as follows:



The intermediate units have threshold 1, all other units keep their threshold. The new Hopfield net H' has the same global optimum as H, since all intermediate units can be set to '1' iff both of their neighbors are in state '1' (this does unfortunately not work for local optimization). The size is increased to $N = n + mn^2 \leq n + 2nn^2 \leq 3n^3$.

In the weighted case a W-approximating solution is already optimal, because the least nonzero energy difference between two solutions is W and the global optimum is W. In the unweighted case observe that the graph which replaces a weighted edge may be viewed as one edge that is external iff both "original" vertices are in different states. This edge "weights" $N^{2/3}$ for the new net size N (the optimum is $N^{1/3}$). So a $N^{1/3}/2 < n$-approximating solution must equal the optimum on the "original" vertices. □

As shown above finding local optima of the positive weight Hopfield function is practically as hard as finding global optima. In addition to that even local and global approximation have almost the same complexity as global optimization in the case of polynomial weights. Thus local approximation may be not too interesting for this easy variant of the Hopfield function. It remains as open problem whether the positive weight Hopfield function with exponential weights can be approximated locally or globally in $\mathcal{NC}$. Answering this question would be very interesting since this problem is not parallelizable for exact local or global optimization.

3.2 The Negative Weight Hopfield Function

When the Hopfield energy function is restricted to negative weights then negative thresholds contribute positively, edges contribute negatively to the value of the energy function. Positive thresholds are useless because they cannot be exceeded. We show how to encode two interesting optimization problems into Hopfield nets with negative weights. First consider the MAX CUT problem (defined on undirected graphs).

Theorem 3.16 *1. The* MAX CUT *problem with positive weights can be solved by a negatively weighted Hopfield net, i.e.,* MAX CUT $\leq_{\mathcal{LCS}}$ N-HOPFIELD.

2. N-HOPFIELD $\leq_{\mathcal{LCS}}$ MAX CUT.

PROOF:

1. Let G be any graph with weighted edges.

$$\begin{aligned}
\max CUT(G) &= \max \sum_{i<j} w_{i,j}[(1-s_i)s_j + (1-s_j)s_i] \\
&= \max \frac{1}{2} \sum_i \sum_{j \neq i} (-2w_{i,j}s_is_j + w_{i,j}s_i + w_{i,j}s_j) \\
&= \max \sum_i \sum_{j \neq i} (-w_{i,j})s_is_j - \sum_i \sum_{j \neq i} \frac{-w_{i,j}}{2} s_i - \sum_i \sum_{j \neq i} \frac{-w_{i,j}}{2} s_j \\
&= \max \sum_{i<j} (-2w_{i,j})s_is_j - \sum_i \left(\sum_{j \neq i} -w_{i,j} \right) s_i.
\end{aligned}$$

   Thus a Hopfield net with negative weights exists that has the same optima as the MAX CUT instance. This reduction preserves the cost of any cut exactly.

2. Given a negatively weighted Hopfield net H add two new vertices v^1 and v^0 which will be forced to be in state '1' resp. '0' in a local optimum. If a unit v_i has threshold $t_i < \sum_{j \neq i} w_{i,j}/2$, then add an edge from v_i to v^0 with weight $2t_i - \sum_{j \neq i} w_{i,j}$ and leave the threshold unchanged. If a unit v_i has threshold $t_i > \sum_{j \neq i} w_{i,j}/2$, then add an edge from v_i to v^1 with weight $\sum_{j \neq i} w_{i,j} - 2t_i$ and change the threshold of v_i to $\sum_{j \neq i} w_{i,j} - t_i$. This does not alter the computation of the net (as shown in theorem 3.2). But now the threshold of every vertex (except v^1 and v^0) is one half of the sum of incident edges and thus we search for a locally maximal cut among these vertices.

   To achieve that v^0 and v^1 also behave like MAX CUT demands simply insert a very large negative edge between them and choose their threshold half the sum of their incident edges. Now all thresholds can be removed. Negate all edge weights (so that they are positive). Call this graph G.

   A locally maximal cut in G clearly separates v^1 and v^0. Define the side of v^1 as '1' and the side of v^0 as '0'. This induces a local optimum for N-HOPFIELD. The reduction does not preserve approximability. □

The proof of the previous theorem implies that finding a global optimum for the negative weight Hopfield function solves a $\mathcal{NP}$-hard problem, since MAX CUT is known to be $\mathcal{NP}$-hard (see [GJ79]). Local search is also hard for MAX CUT: in [SchY91] it was shown that finding a local optimum solves a $\mathcal{P}$-hard problem in the unweighted or polynomially weighted case, and is $\mathcal{PLS}$-complete in the case of unbounded weights.

Now another famous problem is considered: the INDEPENDENT SET problem. Given an undirected graph construct a Hopfield net by taking the same graph, assigning weight -1 to every edge and threshold $-.5$ to every vertex. Clearly in any local optimum no neighboring units are both in state '1', since then turning one of them off gains 1 for the edge and loses only .5 for the threshold. Local and global optima of the net correspond to local and global optima of the INDEPENDENT SET instance. The energy of an optimal state is half the size of an optimal INDEPENDENT SET—hence the embedding preserves approximability (this is a S-reduction).

Theorem 3.17 *1. Finding global optima for N-Hopfield solves a $\mathcal{NP}$-hard problem even in the case of unit weights.*

2. Finding local optima for N-Hopfield solves a $\mathcal{P}$-hard problem in the case of unit size and polynomial size weights, and is $\mathcal{PLS}$-complete in the case of unbounded weights.

3. Approximating global optima for N-Hopfield within n^ϵ for unit weights, polynomial weights, exponential weights solves a $\mathcal{NP}$-hard problem (for some $\epsilon > 0$.

PROOF: The first two statements have already been examined earlier on this page. The third follows from a hardness result for INDEPENDENT SET: [BeSc92] showed that approximating INDEPENDENT SET within n^ϵ solves a $\mathcal{NP}$-hard problem (for some constant $\epsilon > 0$) if MAX 2-SAT has an approximation threshold, i.e., approximating MAX 2-SAT better than some constant is $\mathcal{NP}$-hard. This premiss was shown with methods from the area of probabilistically checkable proof systems (see [AS92] and [ALMSS92]). Since we have just S-reduced INDEPENDENT SET to N-HOPFIELD (with unit weights), proposition 3. is valid. □

The very interesting open question is whether local optima can be approximated easier than they can be found. Consider the following scale of problems: the Hopfield function with negative weights of unit size and with the ratio $|t_i|/degree_i = p$ fixed throughout the net. If $p = 1/2$ then this equals the MAX CUT problem (and can thus be approximated globally, but is hard for local search), if $p = 2/degree_i$ and $degree_i$ is fixed throughout the net, then this is the INDEPENDENT SET problem on graphs of fixed degree (which cannot be approximated globally, but local optima can be found in $\mathcal{NC}$). Even this restricted form of the Hopfield function has the property of being not approximable globally and being hard for local search (the special property of INDEPENDENT SET of being easy for local search does probably not generalize to an interesting fraction of the whole scale of problems since the problems are in general nonmonotone). This problem would be a very interesting candidate for finding a local approximation algorithm, but unfortunately we do not know one.

Another interesting feature of this scale of problems is the way they can be expressed with the Hopfield function on domain $\{-1, 1\}$. The MAX CUT problem is expressed by edge weights -1 and thresholds 0, INDEPENDENT SET by edge weights -1 and thresholds $degree(v) - 1$. Though it is easy to approximate MAX CUT, this seems to be difficult for the negative Hopfield function on domain $\{-1, 1\}$ with thresholds 0. The known approximation algorithms for MAX CUT do not generalize to a good approximation for this function. On the other hand the function that expresses INDEPENDENT SET can easily be approximated: simply set every unit to '1'. Now all thresholds are gained and their sum is twice as large as the sum of edges. But of course this does not yield an efficient approximation for INDEPENDENT SET (which does not exist). This is an example of the way the domain $\{-1, 1\}$ can be rather uncomfortable for an approximation preserving expression of optimization problems.

3.3 The Positive and Negative Weight Hopfield Function

We turn to the consideration of the Hopfield function with unrestricted signs on the weights (note that maximization of PN-HOPFIELD is equivalent to maximization of the sum of weights of satisfied conjunctions in a 2-DNF formula with signed weights and only unnegated variables). Of course hardness results that are valid for a restriction on the signs keep valid.

Theorem 3.18 *1. Finding a globally optimal solution to the* PN-HOPFIELD *function solves a $\mathcal{NP}$-hard problem in the case of unit size weights.*

2. Finding a locally optimal solution to the PN-HOPFIELD *function solves a $\mathcal{P}$-hard problem in the case of unit size and polynomial size weights, and is $\mathcal{PLS}$-complete in the case of unbounded weights.*

3. A global n^ϵ-approximation of the PN-HOPFIELD *function with unit weights solves a $\mathcal{NP}$-hard problem (for some $\epsilon > 0$).*

PROOF: Follows from the same results in the previous section. □

But there is still a possibility to strengthen the above result on global approximability since it is derived from the INDEPENDENT SET problem, which is believed to be incomplete (see [Ka92]) for the class of $\mathcal{NP}$-maximization problems via S-reductions. We conjecture that the N-HOPFIELD function is incomplete, too. The PN-HOPFIELD function on the other hand is complete. First we need two complete problems.

Theorem 3.19 *1.* LONGEST PATH WITH FORBIDDEN PAIRS *is complete for the class of $\mathcal{NP}$-maximization problems with polynomially bounded optima via S-reductions.*

2. MAX CIRCUIT OUTPUT *is complete for the class of $\mathcal{NP}$-maximization problems via S-reductions.*

PROOF:

1. See [BeSc92].

2. Clearly MAX CIRCUIT OUTPUT is a $\mathcal{NP}$-maximization problem. For the reduction take any $\mathcal{NP}$-maximization problem $L = (P, C)$. An instance x of L is mapped by the instance mapping f to the following instance of MAX CIRCUIT OUTPUT: a circuit that, given an input s, outputs 0 if $P(s, x) = 0$, and outputs $C(s, x)$ otherwise.

   A solution s to $f(x)$ with $P(s, x) = 0$ is mapped by the solution mapping g to a standard solution of nonnegative cost to x. A standard solution of nonnegative cost must be computable in polynomial time due to definition 2.9. A solution s with $P(s, x) = 1$ is mapped by g to s.

   Now clearly x and $f(x)$ have the same global optima, and any solution to $f(x)$ is mapped to a solution to x of at most the same performance ratio. Thus (f, g) is a S-reduction (with, of course, polynomial size amplification). □

Theorem 3.20 *1. The* PN-HOPFIELD *function with unit weights is complete via S-reductions for the class of $\mathcal{NP}$-maximization problems with polynomial optima.*

2. The PN-HOPFIELD *function with unbounded weights is complete via S-reductions for the class of $\mathcal{NP}$-maximization problems.*

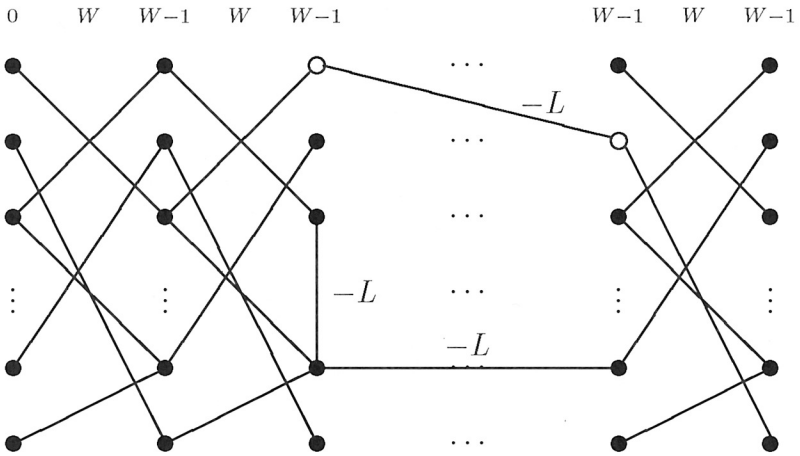
3. Approximating the PN-HOPFIELD *function with unbounded weights globally within 2^{n^ϵ} solves a $\mathcal{NP}$-hard problem (for some $\epsilon > 0$).*

PROOF:

1. By S-reduction from LONGEST PATH WITH FORBIDDEN PAIRS. We first construct a Hopfield net with polynomial weights for this problem, and then show how to reduce the weights to unit size. Let (G, P) be an instance of LONGEST PATH WITH FORBIDDEN PAIRS, where $G = (V, E)$ is a directed graph, P a collection of forbidden pairs. Let m denote the number of vertices of G. The Hopfield net H that is going to find optima of (G, P) is described by a matrix of m^2 units $\{v_{i,j} | i, j = 0 \ldots m-1\}$. The columns represent m copies of the graph. Let $L = 2m^3 + 1$ and $W = m^2$. The connections of H are as follows:

   - Positive connections:
     - between $v_{i,j}$ and $v_{i+1,k}$ if $(v_j, v_k) \in E$ and $0 \leq i \leq m-2$: weight W
   - Negative connections:
     - between $v_{i,j}$ and $v_{i,k}$ for all $j \neq k$ and $0 \leq i \leq m-1$: weight $-L$
     - between $v_{i,k}$ and $v_{j,k}$ for all $i \neq j$ and $0 \leq k \leq m-1$: weight $-L$
     - between $v_{i,j}$ and $v_{k,l}$ if $(v_j, v_l) \in P$, for all $i \neq k$: weight $-L$

   The thresholds of all units in column 0 are 0, all other units have threshold $W - 1$.

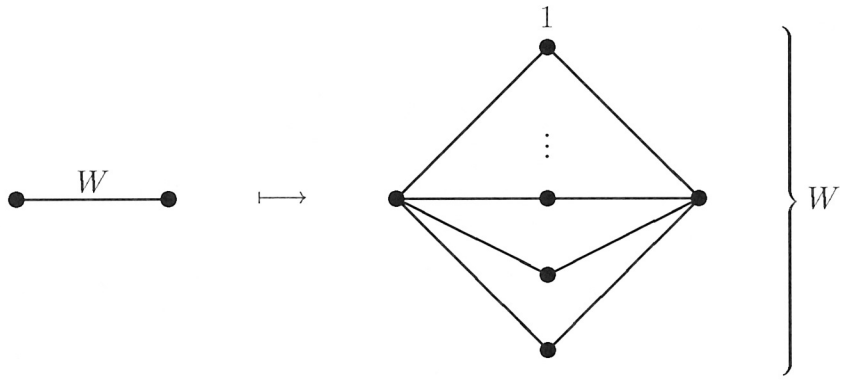


A global optimum of H represents a longest simple path (of length k) without forbidden pairs in the following way: one unit per column is on in the columns 0–k. The unit that is on in column i corresponds to the ith vertex of the path. These units form a path in H, too. All other units are off.

To see this consider a globally optimal state of the net. First assume that two units in the same column are on (i.e., the state does not represent one or more paths in G). Then the loss of this mistake is L and thus larger than the sum of all positively weighted edges incident to a single unit (which is at most $2m \cdot W = 2m^3$). So at least one of the units can be turned off. The same holds if two units in the same row are on (i.e., a representation of paths that traverse the same vertex twice), and if two units that are on correspond to two vertices of a forbidden pair. Thus an optimum corresponds to one or more simple paths without forbidden pairs.

Only a representation of a single path that starts in the first column can have energy more than 0, because any path traverses one more vertex than edge. A path of length k loses $k+1$ thresholds and gains k edges. Only if one of the thresholds is 0 (the path starts in column 0) positive energy is possible, because the threshold $W-1$ exceeds the possible gain of any path. Thus a global optimum represents one simple path beginning in column 0 and respecting all forbidden pairs. The energy of the optimum equals the length of the represented path. Thus a longest path yields a globally optimal state of H.

Now the weights will be reduced. First any positively weighted edge is replaced:



We will refer to the new units as "edge units", to the original units as "vertex units". The resulting Hopfield net has the same global optimum as before (see the proof of theorem 3.15). Now take a negative edge. Instead of connecting $v_{i,j}$ to $v_{k,l}$ with weight $-L$ use a complete bipartite graph of unit weight negative edges between the vertex unit $v_{i,j}$ and all adjacent edge units on one side, and $v_{k,l}$ and all adjacent edge units on the other side.

The new construction expresses the same constraints as before. A state of the Hopfield net (restricted to the vertex units) that does not represent a simple path

without forbidden pairs has energy less than 0: if some vertex unit that is on is neighbor of less than $W/2$ edge units that are on, then there is a loss of at least $W/2 - 1$ (caused by the threshold), and thus the state has energy less than 0. If at least $W/2$ edge units are on in the neighborhood of every vertex unit that is on, then at least $(W/2)^2$ negative edges contribute to the energy for every violated constraint.

Now to the formal definition of the reduction. The instance mapping f maps (G, P) to H. The solution mapping g maps a state of H which represents a simple path without forbidden pairs (on the vertex units) to this path. Any other state is mapped to a trivial solution to (G, P).

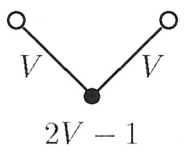
H and (G, P) have global optima of the same cost. A state s of H either represents a simple path without forbidden pairs or has energy at most 0. In the former case the path has at least the same cost for the instance (G, P) as s has for H. In the latter case the trivial solution to (G, P) has larger cost than the state of the net. So $g(s, (G, P))$ has at most the same performance ratio as s.

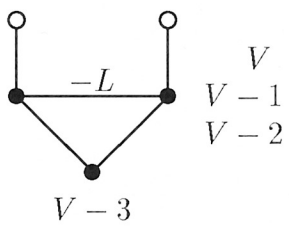
2. By S-reduction from MAX CIRCUIT OUTPUT. Without loss of generality an instance of MAX CIRCUIT OUTPUT is a circuit C made of AND and OR Gates with fan-in 2 and fan-out 2, and of NOT-gates that are only connected to inputs. Note that monotone circuits trivialize MAX CIRCUIT OUTPUT. The circuit is given to the reduction as a list of gates in topological order.

   The Hopfield net that is going to simulate the circuit is called H and consists of three types of units: units for the input gates, units for the internal gates, and a unit that "decodes" the output of the circuit, i.e., represents its value as a contribution to the energy function.

   Let m denote the size of the circuit C, let $W = 4^{m+1}2^m m^2$ and $L = W^2$. There are two units for every input gate i of the circuit: v_i^+ and v_i^-. They are connected by an edge with weight $-L$ (which ensures that they cannot be on both in an optimum). These "input units" have threshold -1 so that one of them can be turned on with a positive gain if both are off (all edges incident to the input units will have positive weight).

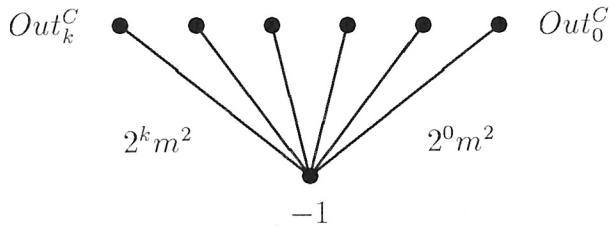
   An AND-gate is simulated by a single unit, an OR-gate by three units. These units are connected to each other and to the units simulating the predecessors of the gate. The weights and thresholds are determined by $V = W/4^i$ for gate i.





The single unit of an AND-graph and the lowest unit of an OR-graph will be referred to as "gate units". The predecessors of the depicted gate simulators are other gate units or input units v_i^+ resp. v_i^-. v_i^+ represents a unnegated input, v_i^- a negated input.

The exponentially decreasing weights ensure that every gate unit computes its gate function on the states of its predecessors, independent of the states of its successors. W is chosen such that even the lowest of all gate units has a threshold of at least $4 \cdot 2^m m^2 - 3$ and is independent of the following representation of the circuit's output. The $k+1$ units simulating the output gates Out_i^C (where $k < m$) are connected to a unit *dec* that "decodes their value".



Now consider a globally optimal state of the net. We can assume that all pairs of input units have complementary states. It is possible to correct the states of gate units in topological order (by local search): if the two predecessors of an AND-unit are on, then the unit can be turned on; if one of them is off, then it can be turned off. If at least one predecessor of an OR-graph is on, then one intermediate unit of the OR-graph can be turned on and afterwards the gate unit, else they can be turned off.

Note that the energy of a correct AND-graph is at most 1, the energy of a correct OR-graph at most 2. Thus in the whole net, except of the edges to *dec*, the energy cannot exceed $2m$. A state that does not represent a correct computation of every gate can be improved by local search. The input units have the value that maximizes the energy. This is the value that maximizes the value encoded by the outputs of the circuit, because this value (multiplied by m^2) is added to the energy function by the edges to *dec* (*dec* is on in any optimum). The energy of a globally optimal state of H is $O(m) + T^{opt} \cdot m^2$ when the circuit C computes T^{opt} (binary encoded) on the states of the v^+. The states of the v^+ induce a globally optimal input for C.

Now to the definition of the reduction. The circuit C is mapped by the instance mapping f to H. Any state of the Hopfield net is mapped by the solution mapping g to the states of the input units v^+.

If a state s of H has energy less than 0 then $g(s, C)$ has a better performance ratio (for C) than s (for H), because the circuit outputs encode nonnegative numbers only. If s has energy at least 0, then it represents a correct computation or a computation where additional gate units can be turned on correctly, *never* a computation where

a gate unit is illegally on (this would lose at least $(4 \cdot 2^m m^2 - 3) - (2^k m^2)$ on the threshold of a gate unit and this loss exceeds the maximal positive energy). The states of the v^+ induce a solution to C of cost T, where T is larger than the output of the circuit-simulation in H. This holds because the circuit-simulation in H is monotone (NOT-gates are implemented as negations of inputs) and no "allowed" error of the circuit-simulation increases its output.

The energy of a state s of H is $O(m) + T'm^2$ for the circuit-simulation output T'. Let T be the output of circuit C on the input $g(s, C)$. Then $T' \leq T$. The globally optimal output of C and of the circuit-simulation is the same value T^{opt}. The performance ratio of $g(s, C)$ is (with some constants c, d)

$$R(g(s,C),C) = \frac{T^{opt}}{T} = O\left(\frac{dm + T^{opt}m^2}{cm + T'm^2}\right) = O(R(s,H))$$

if $T > 0$ and $cm + T'm^2 > 0$. Otherwise it is easy to see that $R(g(s, C), C) < R(s, H)$.

3. Follows from fact 2.7. and statement 2. □

Global optimization of PN-HOPFIELD is a complete problem under an approximability preserving reduction. Thus approximation cannot be done considerably more efficient than optimization for PN-HOPFIELD.

PN-HOPFIELD corresponds to a generalized graph cut problem.

Theorem 3.21 *1. The* MAX CUT *problem with positive and negative weights can be solved by a Hopfield net, i.e.,* PN-MAX CUT$\leq_{\mathcal{LCS}}$ PN-HOPFIELD.

2. PN-HOPFIELD$\leq_{\mathcal{LCS}}$ PN-MAX CUT.

PROOF:

1. Let G be any graph with weighted edges. Now as in theorem 3.16 the PN-MAX CUT problem can be implemented, this time using positive and negative weights.

2. Given a Hopfield net H transform to PN-MAX CUT like in theorem 3.16 by using a vertex clamped to '1' and a vertex clamped to '0'. This "clamping" can be achieved in PN-MAX CUT by a very large negative edge between the two vertices. All thresholds t_i can be transformed (like in theorem 3.16) so that they are half of the sum of weights on edges incident to v_i. This yields a PN-MAX CUT instance. □

Now we return to the consideration of local approximation. First note that the lower bounds for the positively weighted Hopfield function are valid for PN-HOPFIELD, too. In the case of unit weights this is all we know.

Theorem 3.22 *The* PN-HOPFIELD *function with unit weights cannot be approximated locally within* $\sqrt{n}/5$ *in functional* $\mathcal{AC}^0$.

The above result is very weak since obtaining an exact solution solves a $\mathcal{P}$-hard problem. If the weights are polynomial or unbounded the result can be improved.

Theorem 3.23 *1.* PN-HOPFIELD^{pol} *is* $\mathcal{NC}^1\mathcal{LS}^{pol}$*-complete via* $\mathcal{NC}$*-LPR-reductions.*

2. PN-HOPFIELD^{exp} *is* $\mathcal{PLS}$*-complete via* $\mathcal{P}$*-LPR-reductions.*

3. A local n^k*-approximation of* PN-HOPFIELD^{pol} *solves a* $\mathcal{P}$*-hard problem.*

4. A local $2^{\epsilon\cdot\sqrt{n}}$*-approximation of* PN-HOPFIELD^{exp} *solves a* $\mathcal{PLS}$*-hard problem (for some* $\epsilon > 0$*).*

PROOF: We will show a $\mathcal{NC}$-LPR-reduction (with size amplification $O(n^2)$) from $\mathcal{NC}^1$-FLIP$_{k\cdot\log}$ (for arbitrary k) to PN-HOPFIELD^{pol} (with weights bounded by $n^{O(k)}$). Afterwards we show that this construction also $\mathcal{NC}$-LPR-reduces FLIP to PN-HOPFIELD^{exp}. Theorem 2.15 leads to statements 1. and 2. (the implicit inclusion statements are trivial).

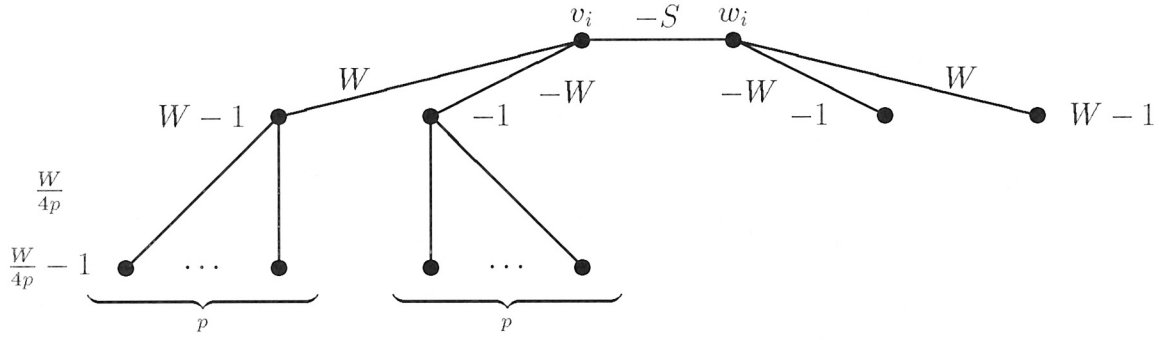
Theorem 2.17 implies that a n^{2k}-approximation for $\mathcal{NC}^1$-FLIP$_{4k\cdot log}$ solves a $\mathcal{P}$-hard problem, and that a $2^{\epsilon\cdot n}$-approximation for FLIP solves a $\mathcal{PLS}$-hard problem for some $\epsilon > 0$. The reductions of 1. and 2. imply statements 3. and 4. via theorem 2.14, because their size amplification is $O(n^2)$.

For the reduction first a given circuit has to be mapped to a Hopfield net with polynomial weights. We can assume that this circuit has p inputs, q outputs, size m, and depth d with $q, d = O(\log m)$, and that the circuit consists of fan-in/fan-out 2 AND/OR gates and of NOT gates that are connected directly to the inputs. The Hopfield net H uses representations of the p inputs of the circuit and works out $p+1$ simulations of the circuit, one for the "normal" input assignment (called the "original" circuit C) and one for each possibility to flip one input variable. The idea to use such "test circuits" was introduced (while $\mathcal{PLS}$-reducing FLIP to MAX SAT) in [Kr90]. There is one test circuit T_i for every variable. The net will allow to flip a variable in the normal input assignment iff the corresponding test circuit has a larger output than the original circuit. This will be done in a complicated sequence of local optimization steps. H has only local optima in which the original circuit and all test circuits compute correctly, and the original circuit has a larger output than any test circuit. In this situation the variables induce a locally optimal solution to the $\mathcal{NC}^1$-FLIP$_{log}$ instance.

H consists of units representing the input variables, of units representing the circuit gates, of units decoding the outputs of the circuits, and of control units. There are $2p$ "variable units" v_i and w_i for $1 \leq i \leq p$. The v_i represent the "normal" input, whereas the w_i correspond to the complements of the v_i. v_i is read by C and by the T_j for $j \neq i$, w_i is read by T_i. Other units form the circuits C and T_i in a similar way as in the proof of theorem 3.20. Note that we are allowed to use weights that are exponential in the depth of the circuits. The outputs of every circuit are connected to a vertex that decodes them using weights exponentially large in q and represents the binary encoded output as a contribution to the energy function. There are 6 more control units for every test circuit and variable. The net has size $O(pm) = O(m^2)$.

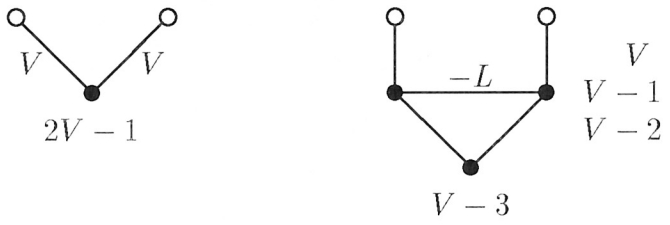
Before describing the whole construction in detail we give a short explanation of the way local search will change the state of the net towards a local optimum. It will be possible to get into a state where every circuit computes correctly and the states of the variable units represent some input assignment. If a circuit T_i has larger output than C, then turning on a single "decoding" unit (called max_i) gains the value of this output while losing the value of the output of C. max_i activates some control units, these "disconnect" the circuits C and T_j for $j \neq i$, i.e., turn every unit in these circuits off, and flip the variable v_i. Then circuit C will be "connected" again, C recomputes its output and is now maximal, max_i can be turned off. This allows to disconnect T_i and flip w_i in turn. Now the net can return to a standard situation, and the whole process can start again—until the inputs of C produce a locally optimal output.

Let $R = 4pm$, $S = R^2$, $W = 4p4^d \cdot S^7 2^q$, $L = W^2$. Note that all these numbers are polynomially bounded in m. v_i and w_i are connected by an edge of weight $-S$, they both have threshold $-3R$. Every variable unit is connected to a unit duplicating its state and to a unit computing its negation. The first of these units is connected with weight W and has threshold $W-1$. The second of these units is connected with weight $-W$ and has threshold -1. In the case of v_i these two units are each connected to p units duplicating their states. This makes the state of v_i and its negation available p times (for the p circuits that may want to read it). w_i is read by only one circuit and thus does not need this duplication. The $2 \cdot p$ duplicating units have threshold $W/(4p)-1$. Their connections have weight $W/(4p)$. The construction is depicted below. The two successors of the variable units will be referred to as "variable duplicators", their successors as "input units" of the circuits.



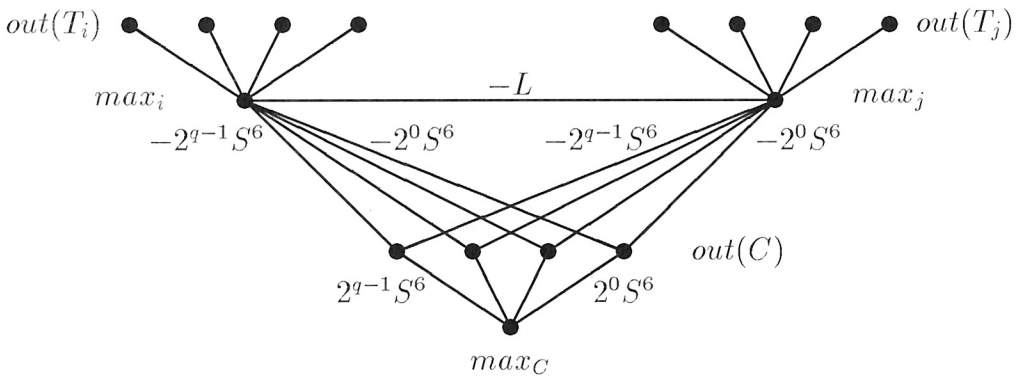
Suppose that v_i (or w_i) has some fixed state. Then local search can proceed as follows: if v_i is on, then its successor with threshold -1 can be turned off, its other successor can be turned on. If v_i is off, then its successor with threshold -1 can be turned on, its other successor can be turned off. These two units are independent of their p successors, which are connected to them with weight $W/(4p)$. The states of the units under the two direct successors of v_i can be changed by local search so that they duplicate the states of their predecessors. Note the following: if all these units are off and v_i has some state (but is not fixed), then local search leads to the situation, where the units in the left branch under v_i duplicate the state of v_i and the units in the right branch under v_i duplicate the negation of the state of v_i. This does not change the state of v_i.

The circuits C and T_i are placed below this construction and are connected to the (negated or unnegated) input units, where C is connected to inputs that follow the v_i only, while T_i is connected to inputs that follow w_i and to those that follow v_j for $j \neq i$. The circuits contain no further negations (NOT-gates). The gates of the circuits are simulated by one unit for an AND gate and three units for an OR gate, where an edge leading into a gate in depth i has weight $V = W/(4p \cdot 4^i)$. Note that on the output level the value V is still at least $W/(4p \cdot 4^d) = S^7 2^q$. This ensures that the output gate units are independent of the deeper decoding units. The gate simulators are:



These gate simulation graphs can be corrected by local search from the top to the bottom due to the exponentially decreasing weights (like in the proof of theorem 3.20). Note that turning on any unit in these circuit simulations or in the higher variable duplication units can contribute at most 1 to the energy function, i.e., the gain from edges to higher units minus the threshold is always at most 1. The gain from edges to lower units is counted as the gain of these lower units.

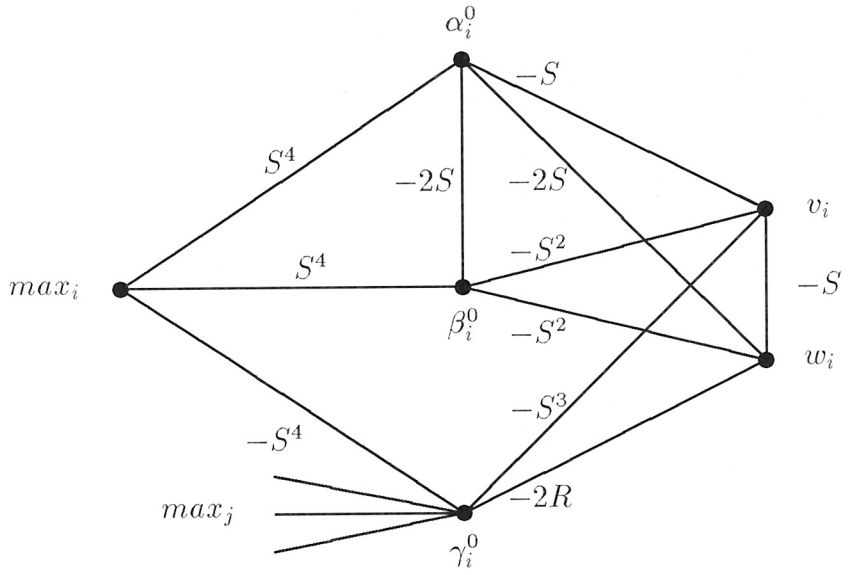
The q outputs of circuit C are connected to a vertex max_C (which has threshold -1) with weight $2^j S^6$ for the jth output bit (max_C is on in every local optimum). The outputs of the circuit T_i are connected to a unit max_i with weight $2^j S^6$ for the jth output bit. max_i has threshold S^5. max_i is also connected to the output gate units of C with weight $-2^j S^6$ for output j. Additionally all the max_i (not max_C) are fully interconnected with weight $-L$ on all edges. This construction allows to turn on one max_i if at least one T_i has a larger output than C. The $-L$ connections ensure that at most one max_i can be turned on with a positive gain.



The crucial idea of this construction is that turning on a single max_i makes T_i "carry" the new maximal output (i.e., loses the output of C and gains the output of T_i) as long as C has to be corrected and the variable v_i has to be flipped. Afterwards C "carries" the increased output again. To implement the correction process there are $6p$ control variables $\alpha_i^0, \beta_i^0, \gamma_i^0$ and $\alpha_i^1, \beta_i^1, \gamma_i^1$ for $1 \leq i \leq p$.

Local search will proceed roughly in the following way: if the states of the outputs of T_i encode a larger value than those of the outputs of C, then max_i can be turned on and will start a control process that "disconnects" C and T_j ($j \neq i$) and the variable duplicators of v_i by edges of weight -2 leading into every unit of these. The thresholds of the output gate units of C and of the T_j are not longer exceeded (max_i is on and all max_j are off). These units can all be turned off. Now the thresholds of their predecessors are no more exceeded and these can be turned off. In this way all units in the circuits C and T_j can be turned off from the outputs upward. When all units in the disconnected circuits and variable duplicators are off, then the variable v_i is connected to no unit that is on, except possibly w_i. The control units allow to flip v_i and connect the circuit C and the variable duplicators of v_i again (i.e., the control unit with the weight -2 edges to C and these duplicators is turned off). C is recomputed starting from the vector of zeroes and thus leaving the variable unit v_i unchanged. C computes the same output as T_i and max_i can be turned off. At this time the T_j are connected again and can be recomputed starting from the vector of zeroes. The circuit T_i can be disconnected in order to flip the variable w_i. After the recomputation of T_i a new cycle can start. In this way the Hopfield net moves to larger and larger energy, where the current maximal output dominates the energy function and is always preserved.

The control units $\alpha_i^0, \beta_i^0, \gamma_i^0$ act in the situation when $v_i, w_i = 1, 0$ or $v_i, w_i = 0, 0$ (in the following we will identify units with their states). α_i^0 is connected to max_i with weight S^4, to v_i with weight $-S$, and to w_i with weight $-2S$. α_i^0 has threshold $S^4 - S - R$. Additionally α_i^0 is connected to all units in C and T_j for $j \neq i$ and to the variable duplicators of v_i with weight -2. These connections weight together some value between 0 and $-R$ and are irrelevant for the decision whether α_i^0 can be turned on. β_i^0 is connected to max_i with weight S^4, to v_i with weight $-S^2$, to w_i with weight $-S^2$, to α_i^0 with weight $-2S$, and to all units in $T_{j \neq i}$ with weight -2. β_i^0 has threshold $S^4 - 2S - R$. γ_i^0 is connected to all max_i with weight $-S^4$, to v_i with weight $-S^3$, to w_i with weight $-2R$, and to all units in T_i including the variable duplicators of w_i with weight -2. γ_i^0 has threshold $-R$.



The "equivalence" stated the following propositions means that a situation contradicting the statements can be improved by one or two local search steps.

Lemma 3.24 *1.* $\alpha_i^0 = 1$ *iff* $max_i = 1$ *and* $v_i = 1$ *and* $w_i = 0$.

2. $\beta_i^0 = 1$ *iff* $max_i = 1$ *and* $v_i = w_i = 0$.

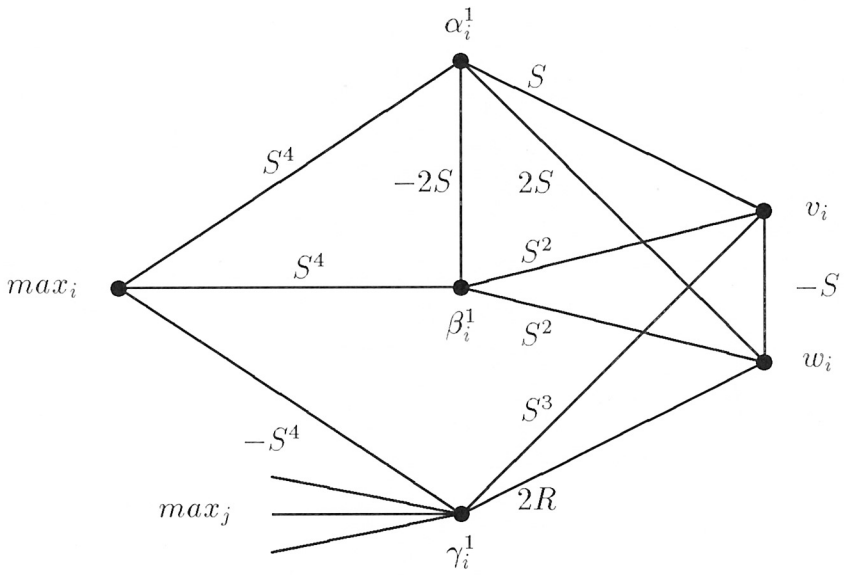
3. $\gamma_i^0 = 1$ *iff* $max_j = 0$ *for all* j *and* $v_i = w_i = 0$.

If $\gamma_i^0 = 1$ then all max_j must be off and $v_i = w_i = 0$, because else its threshold of $-R$ is not exceeded. Conversely, if max_j is off and $v_i = w_i = 0$, then γ_i^0 can be set to 1, because its threshold is exceeded even if all other negatively connected neighbors are on.

$\beta_i^0 = 1$ iff $max_i = 1$ and $v_i = w_i = 0$ is clear from the threshold $S^4 - 2S - R$. The connections to α_i^0 and to the T_j do not matter.

If $\alpha_i^0 = 1$ then max_i must be on and w_i must be off as well as β_i^0, otherwise its threshold $S^4 - S - R$ is not exceeded. If $v_i = w_i = 0$ then β_i^0 can be turned on, and then the threshold of α_i^0 is not exceeded, too.

The control units $\alpha_i^1, \beta_i^1, \gamma_i^1$ act in the situation when $v_i, w_i = 0, 1$ or $v_i, w_i = 1, 1$. α_i^1 is connected to max_i with weight S^4, to v_i with weight S, and to w_i with weight $2S$. α_i^1 has threshold $S^4 + 2S - R$. Additionally α_i^1 is connected to all units in C and T_j for $j \neq i$ and to the variable duplicators of v_i with weight -2. β_i^1 is connected to max_i with weight S^4, to v_i with weight S^2, to w_i with weight S^2, to α_i^1 with weight $-2S$, and to all units in $T_{j \neq i}$ with weight -2. β_i^1 has threshold $S^4 + 2S^2 - 2S - R$. γ_i^1 is connected to all max_i with weight $-S^4$, to v_i with weight S^3, to w_i with weight $2R$, and to all units in T_i including the variable duplicators of w_i with weight -2. γ_i^1 has threshold $S^3 + R$.



Lemma 3.25 *1.* $\alpha_i^1 = 1$ *iff* $max_i = 1$ *and* $v_i = 0$ *and* $w_i = 1$.

2. $\beta_i^1 = 1$ *iff* $max_i = 1$ *and* $v_i = w_i = 1$.

3. $\gamma_i^1 = 1$ *iff* $max_j = 0$ *for all* j *and* $v_i = w_i = 1$.

If $\gamma_i^1 = 1$ then all max_j must be off and $v_i = w_i = 1$, because else its threshold of $S^3 + R$ is not exceeded. Conversely, if $max_j = 0$ and $v_i = w_i = 1$, then γ_i^1 can be set to 1, because its threshold is exceeded even if all other negatively connected neighbors are on.

$\beta_i^1 = 1$ iff $max_i = 1$ and $v_i = w_i = 1$ is clear from the threshold $S^4 + 2S^2 - 2S - R$.

If $\alpha_i^1 = 1$ then max_i must be on and w_i must be on as well. β_i^1 must be off, otherwise its threshold $S^4 + 2S - R$ is not exceeded. If $v_i = w_i = 1$ then β_i^1 can be turned on and the threshold of α_i^1 is not exceeded, too.

Now the net H is completely defined.

The dynamic of local search is as follows: if the output of C is not maximal, then some test circuit T_i has a larger output. Turning on max_i loses the value of the output of C on the negative edges to the output units of C, but gains the value of the output of T_i (the difference is larger than the threshold S^5 of max_i because of the multiplication of the output value by S^6). Then one α_i can be turned on, the circuits C and $T_{j \neq i}$ can be turned off from the outputs upwards to the variable duplicators of v_i (every unit in the circuits receives an edge of weight -2 from α_i). The variable v_i flips, β_i can be turned on, α_i can be turned off, the circuit C (and the variable duplicators of v_i) can be recomputed, the output of C is equal to the output of T_i, and max_i is turned off (max_i has input $O(S^4)$ and threshold S^5). Then β_i can be turned off. Now all test circuits except T_i can recompute their output (and the process up to here can take place for another variable until eventually for every variable $w_i = v_i$). Also γ_i can be turned on, then T_i is disconnected, w_i can be flipped, and γ_i is turned off reconnecting T_i.

Lemma 3.26 *In any local optimum of H all circuits compute correctly, all variable units satisfy $v_i \neq w_i$, and the value encoded by the outputs of C is at least as large as the value encoded by the outputs of any other test circuit, i.e., the variable units induce a local optimum for $\mathcal{NC}^1$-FLIP$_{log}$.*

Consider any situation of the net supposed to be a local optimum. First assume some connected (i.e., all control units disconnecting it are off) circuit would not compute correctly. In this case either some unit of a variable duplicator or some gate unit would produce this error. If a unit in a variable duplicator is wrong then either the variable unit or its successors can be changed by local search. If one of the input units or of the gate units in the circuits is wrong, then it can be corrected by local search. All other gate units can be corrected from the top to the bottom. So in a local optimum the variable duplicators have the right states and the circuits compute correctly.

Now we show that the proposition of the lemma holds. Consider a state supposed to be locally optimal. In this situation the statements of lemma 3.24 and lemma 3.25 can be used as real equivalences.

1. For the first case assume that some $max_i = 1$. Clearly no other max_i is on because of the very large negative connections between them. Now the circuit T_i must be connected (because γ_i cannot be on) and therefore T_i computes correctly. It is also true that the output of T_i is larger than the output of C.

   (a) If $v_i, w_i = 0, 1$ and $max_i = 1$ then α_i^1 can be turned on. Now all gates of C and $T_{j \neq i}$ can be turned off from the outputs up to the variable duplicators of v_i (the thresholds of the outputs of C are not exceeded), leaving v_i only connected to the units α_i^1 and w_i in state '1'. Thus v_i can be turned on (v_i has input $S - S$ and threshold $-3R$). Then β_i^1 can be turned on, α_i^1 can be turned off, and C can be recomputed from the vector of zeroes. This leaves v_i unchanged. The new output of C is the same as the output of T_i and thus max_i can be turned off. Hence the considered state is no local optimum. Contradiction.

   (b) If $v_i, w_i = 1, 0$ and $max_i = 1$ then α_i^0 can be turned on. Now all gates of C and $T_{j \neq i}$ can be turned off up to the variable duplicators leaving v_i only connected to α_i^0 in state '1'. Then v_i can be turned off (v_i has input $-S$ and threshold $-3R$), β_i^0 can be turned on, α_i^0 can be turned off, and C can be recomputed from the vector of zeroes. After this it is possible to turn max_i off. Contradiction.

   (c) If $v_i = w_i = e$ and $max_i = 1$ then β_i^e can be turned on. Then α_i^e can be turned off and C is connected. This implies that C computes correctly. If C is recomputed from the vector of zeroes then a correction does not affect v_i. Otherwise this may be the case leading to one of the cases a) and b). So both circuits C and T_i compute correctly and on the same inputs. They have the same output and max_i can be turned off. Contradiction.

2. Now assume for the second case that all $\max_i$ are 0. Then for all variables with $v_i = w_i$ the second variable can be flipped.

   (a) If $v_i = w_i = 1$ then it is possible to turn on γ_i^1, disconnect T_i, flip w_i to 0 (w_i has input $-S + 2R$ and threshold $-3R$), turn off γ_i^1 again, and recompute T_i from the vector of zeroes (without changing w_i).

   (b) If $v_i = w_i = 0$ then it is possible to turn on γ_i^0, disconnect T_i, flip w_i to 1 (w_i has input $-2R$, threshold $-3R$), turn off γ_i^0 again, and recompute T_i from the vector of zeroes (without changing w_i).

Thus $max_i = 0$ for all i, all $v_i \neq w_i$, and all circuits are connected and compute correctly. The output of C is at least as large as the outputs of the T_i, and these compute the 1-flip neighbors. So the variables induce a locally optimal solution to the $\mathcal{NC}^1$-FLIP$_{log}$ instance that was mapped to H.

Lemma 3.27 *A local optimum of H has energy $\Theta(S^6 \cdot output(C))$.*

Since all gate graphs gain at most R, every variable unit gains at most R, and everything else at most $O(pS^4)$ the edges at the output units of C dominate the energy.

The reduction from $\mathcal{NC}^1$-FLIP$_{log}$ consists of the described mapping f from a circuit T to the net H. The solution mapping g maps a state s of the net H to the states of those variables v_i and w_i that are read by the circuit with the largest output, i.e., w_i is used when T_i has the largest output.

A locally optimal state of H is mapped to the v_i, and these induce a solution to the $\mathcal{NC}^1$-FLIP$_{log}$ instance T of almost the same local performance due to lemma 3.26/3.27.

If a state s of H has energy below zero then any string is a better solution to T and the local performance ratio of $g(s, T)$ for T is larger than the local performance ratio of s for H.

If the energy of s is larger than zero then no severe error happens in a computation: no $-L$ connection is active and no unit in a circuit simulation is on which should not. Units may be off which should be on, but this decreases the value of the circuit outputs and hence the energy of the state, because the circuit simulation uses monotone gates only. So the states of the variable units induce a solution to T with larger cost than the value of the outputs of the circuits in the net. The inputs read by the largest circuit induce a solution with almost the same or better performance ratio compared to the state of the net, because the outputs dominate the energy function. Thus this is a LPR-reduction. The reduction is computable in functional $\mathcal{LOGSPACE}$.

Now observe that exactly the same construction works for FLIP when exponential weights are allowed. Exponential values of the parameter W are used due to the possibly linear number of outputs and the linear depth of the given circuit. □

We have examined the general Hopfield function. It is a "very complete" problem—possibly one of the hardest $\mathcal{NP}$ optimization problems. The most interesting remaining open question is whether this function can be approximated locally in the case of unit weights.

4 Graph Cut problems

We will now investigate the three graph cut problems that are exactly as hard to optimize as the three versions of the Hopfield energy function. The reductions to the cut problems however did not preserve approximability. The situation is therefore that hardness results for MAX CUT with nonnegative or with signed weights are valid for the N- resp. PN-HOPFIELD function, too, whereas hardness results for the Hopfield energy function do not generalize to the MAX CUT problems. The reductions between s,t-MIN CUT and P-HOPFIELD did not preserve approximability either.

4.1 The MIN CUT/MAX FLOW Problem

The positive weight Hopfield energy function is related to the s,t-MIN CUT problem (as shown in theorem 3.2). The reason why s,t-MIN CUT is famous is its connection to s,t-MAX FLOW: both problems have the same set of instances, and global optima of s,t-MIN CUT and of s,t-MAX FLOW on the same instance have the same value (see [VL90]).

Theorem 4.1 *1. Global optima of s,t-MAX FLOW can be constructed in functional $\mathcal{RTC}^1$ if all edge weights are polynomially bounded.*

2. Global optima of s,t-MAX FLOW can be constructed in polynomial time in the case of unbounded weights. Finding the value of a global optimum solves a $\mathcal{P}$-complete problem.

3. Global optima of s,t-MIN CUT can be constructed in functional $\mathcal{RTC}^1$ if all edge weights are polynomially bounded.

4. Global optima of s,t-MIN CUT can be constructed in polynomial time in the case of unbounded weights. Finding the value of a global optimum solves a $\mathcal{P}$-complete problem.

5. Finding local optima of s,t-MIN CUT solves a $\mathcal{P}$-complete problem in the case of unbounded weights, solves a $\mathcal{LOGSPACE}$-complete problem in the case of polynomial weights, and is impossible in functional $\mathcal{AC}^0$ in the case of unit weights.

PROOF:

1. In [KarRa90] an algorithm is described that constructs a global optimum of s,t-MAX FLOW in the case of polynomial weights by computing the determinant of a polynomial size integer matrix (which contains some random numbers). It is shown in [Pan85] that $O(\log n)$ integer matrix multiplications suffice to compute the determinant of an $n \times n$ integer matrix. An integer matrix multiplication can be computed in functional $\mathcal{TC}^0$ due to fact 2.5, thus the determinant can be computed in functional $\mathcal{TC}^1$. This algorithm for s,t-MAX FLOW implies statement 1) of the theorem.

2. See [VL90] and [KarRa90].

3. Given a global optimum of s,t-MAX FLOW one can can construct a global optimum of s,t-MIN CUT by breadth first search (see [VL90]). This is possible in $\mathcal{AC}^1$ (see [KarRa90]).

4. An algorithm follows from 2) and the note to 3). The hardness result follows from statement 2) and the fact that global s,t-MAX FLOW and s,t-MIN CUT optima have the same value.

5. Local optima of s,t-MIN CUT are as hard to find as local optima of P-HOPFIELD (see theorem 3.2). Theorems 3.4, 3.7, 3.5 imply the statement. □

The classical algorithms for MAX FLOW build on finding augmenting paths and increasing the flow along these until no more augmenting paths exist and the obtained flow is optimal. This can be viewed as a local search process: two flows are neighboring if they differ by a single augmenting path. A somewhat easier neighborhood is defined by augmenting paths that are directed forward on every edge (general augmenting paths include backward edges where the flow is reduced). An optimal flow with respect to this neighborhood is used as a building stone in Dinic's MAX FLOW algorithm as well as in other related work (see e.g. [Co92]).

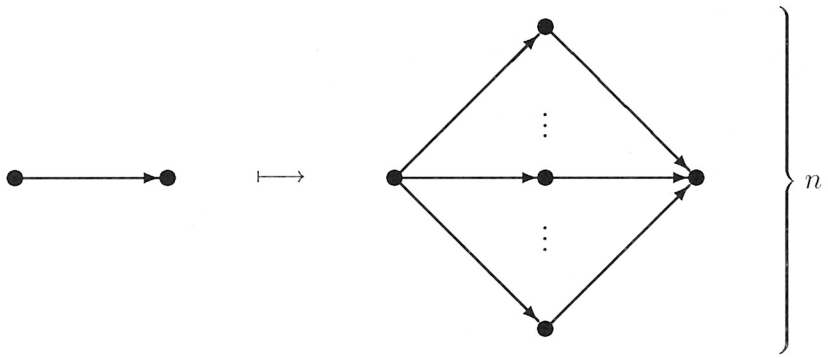
Definition 4.1 *A blocking flow is a flow such that every path from s to t contains at least one saturated edge.*

Obviously a blocking flow is a locally optimal flow with respect to the "forward path" neighborhood. It is unknown whether blocking flows are easier to compute than maximum flows. The following shows that both tasks are at least not very easy.

Theorem 4.2 *Approximating a global optimum of s,t-MIN CUT, s,t-MAX FLOW, and approximating a s,t-BLOCKING FLOW in directed acyclic graphs within 2^n (exponential weights), n^k (polynomial weights), $\sqrt{n}/2$ (unit weights) solves a $\mathcal{NLOGSPACE}$-hard problem.*

PROOF: Approximation solves the GRAPH ACCESSIBILITY PROBLEM (lemma 3.12). A legal s,t-blocking flow on a given directed graph of size n, indegree 2, outdegree 2 with two special vertices s and t has either value 0 (if no path from s to t exists) or at least the value of the smallest edge weight on a path from s to t. Edge capacities can be used to make an arbitrarily bad approximation (with respect to the maximal allowed size of weights) solve the accessibility problem. Choose the weights of all edges as $W = n^k$ in the case of polynomial weights and as $W = 2^n$ in the case of exponential weights. Now clearly a W-approximation solves the accessibility problem.

In the case of unit weights replace every edge by the following graph:



Clearly if a path from s to t exists, then any blocking flow has a value of at least n, otherwise exactly 0. The size of the graph is increased to at most $N = n + 2n \cdot n$. A $\sqrt{N}/2 \leq n$-approximation solves the accessibility problem.

The same result is valid for the stronger s,t-MAX FLOW and also for s,t-MIN CUT: if a path from s to t exists, then the globally minimal cut has at least value n^k resp. 2^n resp. n, otherwise 0. □

Global optimization of s,t-MAX FLOW and s,t-MIN CUT solves a $\mathcal{P}$-complete problem in the case of unbounded weights, so a large gap lies between the above result and the hardness of optimization. It is unknown whether these problems can be approximated in $\mathcal{NC}$. They are pseudo-$\mathcal{RNC}$ problems since they can be solved in $\mathcal{RNC}$ when the weights are polynomial. But a technique similar to that applied to the KNAPSACK problem to achieve an approximation-scheme (by dividing the weights down to polynomiality, see [PapSt82]) does not work. The best approximations that are known work only in $\mathcal{NC}$ if the depth of the input graph, i.e., the maximal length of a path from s to t, is polylogarithmic (see [Co92], where approximations of blocking flows are used, these approximations depend on the depth, too). However, we can present an amplification result (similar to one known for INDEPENDENT SET, see [PapSt82]) which says that approximation is either hard or very easy.

Theorem 4.3 *1. If s,t-MAX FLOW with unbounded weights can be approximated globally in $(\mathcal{R})\mathcal{NC}$ within a constant, then it has a (randomized) NCAS.*

2. If s,t-MIN CUT with unbounded weights can be approximated globally in $(\mathcal{R})\mathcal{NC}$ within a constant, then it has a (randomized) NCAS.

3. If s,t-BLOCKING FLOW with unbounded weights can be approximated in $(\mathcal{R})\mathcal{NC}$ within a constant, then it has a (randomized) NCAS.

PROOF: Assume that a $\mathcal{NC}$ algorithm $\mathcal{A}$ exists which produces (given a graph G) a legal flow of at least $(1-\epsilon)$ the value of the maximum flow for some constant ϵ. Let G be a directed graph with weighted edges and two special vertices s and t. We can assume that s has indegree 0 and that t has outdegree 0, because otherwise we can insert new

vertices s' and t' without changing the value of a maximum flow, that are predecessor resp. successor of s resp. t and have the demanded property. It is also allowed to restrict the choice of s and t to single vertices instead of subsets of the vertex set, because such vertex subsets can be merged into a single vertex.

We will insert G into itself, run $\mathcal{A}$ on this new graph and read off a better solution than $\mathcal{A}$ normally guarantees thus yielding an approximation algorithm of improved quality. This construction can be iterated constantly often producing approximations of every desired constant quality, i.e., an approximation scheme.

We will identify flows F with their value. Let G' be as follows: first take G. Then for every edge $e = (u, v)$ with weight w insert the graph G, i.e., replace e by G such that u coincides with s and v with t. All other vertices of G are inserted as new vertices, as well as all edges. Now the graph replacing the edge can carry exactly the maximum flow of G. To express the weight multiply all edges in the small copy of G with w. The new graph G' (after replacing all edges by copies of G) has size $m(n-2)+n$ for n being the number of vertices of G and m the number of edges. The maximum flow of G' is F_{max}^2 for the maximum flow F_{max} of G, because every graph replacing an edge can carry wF_{max}. The s vertex and the t vertex of G' (called s_l and t_l) are the s and t vertex of the "large copy" of G that connects the "small copies" which replace the edges.

Now run $\mathcal{A}$ on G'. The algorithm produces a solution with value at least $(1-\epsilon)F_{max}^2$. At least one of the small copies of G or the large copy of G that connects these will induce a better flow than $(1-\epsilon)$ of the optimum. This is the new algorithm:

1. Compute G' from G.
2. Run $\mathcal{A}$ on G', receive a flow F'.
3. Find the small copy of G in G' that carries the largest flow of all small copies after division by its corresponding edge weight w. Call this flow F_a.
4. Compute a flow in G by replacing the flow through every small copy of G by an edge flow again and by dividing the whole flow by F_a. Call this flow F_b.
5. Output the maximum of $\{F_a, F_b\}$.

Observe that $F_b = F'/F_a$. F_a is clearly a feasible flow in G. F_b is a feasible flow in G, too. This holds because replacing a flow in a small copy by an edge flow induces a feasible flow, and dividing all edge flows by the same value, too. Additionally every edge in G carries at most its weight w and at least 0: since s has indegree 0 in G it is impossible that a small copy carries a flow from its t to its s, thus the flow of any small copy (from its s to its t) is at least 0 and at most wF_a (for the corresponding edge weight w).

The maximum of $\{F_a, F_b\}$ is an improved approximation: If $F_a \leq \sqrt{1-\epsilon}F_{max}$ then

$$F_b = \frac{F'}{F_a} \geq \frac{(1-\epsilon)F_{max}^2}{\sqrt{1-\epsilon}F_{max}} = \sqrt{1-\epsilon}F_{max},$$

and the flow F_b is an improved approximation.

If $F_a \geq \sqrt{1-\epsilon} F_{max}$ then the flow F_a is an improved approximation.

Thus in every case the algorithm gives an improved approximation. Iterating the construction a constant number of times yields a NCAS: to find a solution that is $1-\delta$ as large as the maximum k iterations suffice, where

$$1-\delta = (1-\epsilon)^{(1/2)^k} \iff k = \log\log\left(\frac{1}{1-\epsilon}\right) - \log\log\left(\frac{1}{1-\delta}\right).$$

Observe that the same technique works for s,t-MIN CUT. In the case of s,t-BLOCKING FLOW observe that the same as above holds for F_{max} being the least blocking flow (approximation in the sense of local approximation of section 2.4). □

Theorem 4.4 *Approximating s,t-MIN CUT with unbounded weights locally within 2^n solves a $\mathcal{LOGSPACE}$-hard problem.*

PROOF: Approximation of undirected s,t-MIN CUT solves the GRAPH ACCESSIBILITY PROBLEM on directed acyclic graphs restricted to outdegree 1 (see theorem 3.7). Given a directed acyclic graph G of size m with indegree at most 2 and outdegree at most 1 and two special vertices start and acc first turn every edge around. Every vertex of the new graph G' has outdegree at most 2 and indegree at most 1.

The input graph for s,t-MIN CUT is described by a matrix of m^2 vertices $v_{i,j}$ where each column is a copy of the vertices of G'. A directed edge (v_j, v_k) of G' is replaced by undirected edges $(v_{i,j}, v_{i+1,k})$ for every $0 \leq i \leq m-1$. The edges between column i and $i+1$ have weight 4^{m^2-i}. Note that every vertex of G'' has degree at most 1 to the left neighbor column and degree at most 2 to the right neighbor column.

The set s consists of all copies of acc, t consists of all copies of start and all copies of vertices with indegree 0 in G' except of acc and all vertices different from acc in column 0. Call this instance of s,t-MIN CUT G''.

Now consider any feasible s,t-cut in G''. If a vertex $v_{i,j}$ in column i is connected to a vertex in column $i-1$ that is on the side of s, then $v_{i,j}$ can be moved to that side due to the exponentially decreasing weights. The same holds for the side of t, because the edges are undirected and all edges between the sides s and t count.

So in a local optimum for a vertex that is no copy of start and not in column 0 the following holds: a vertex that is reachable by a path from the left starting at some vertex in s is on the side of s. All other vertices are on the side of t, because they are reachable by a path from the left starting at some vertex in t.

If no path from start to acc exists in G, then in G'' all copies of start are not reachable from paths beginning at some copy of acc to the left. If this holds then in a locally optimal cut the sides of s and t are not connected to each other and the cut values 0. If on the other hand a path from start to acc exists in G, then at least one edge crosses every s,t-cut with weight at least $4^{m^2-m} > 2^n$. □

It seems as if finding a local optimum and a global optimum of s,t-MIN CUT are almost equally hard tasks. Global approximation seems to be not much easier. It is an open question whether local approximation is easier.

4.2 MAX CUT and MAX (S)NP

It is shown in theorem 3.16 that MAX CUT is exactly as hard to optimize as N-HOPFIELD (see [GJ79] and [SchY91] for complexity results on MAX CUT). On the other hand it is known that MAX CUT can be approximated very well. State of the art is a polynomial time approximation algorithm that produces a solution .878 as good as the global optimum ([GoWi94], which results in a performance ratio of 1/0.878). This improves the 2-approximation (.5 of the optimum) known long before. On the other hand it was shown that there is no polynomial time approximation scheme for MAX CUT and some other problems. These problems are hard for the class MAX SNP or the class MAX NP defined by Papadimitriou and Yannakakis ([PapY91]) with syntactical means related to a syntactical characterization of $\mathcal{NP}$ given by Fagin ([Fa74]).

Fact 4.5 *$\mathcal{NP}$ consists of all predicates on structures G which can be expressed in the form $\exists S : \phi(G, S)$, where S is a structure (and thus the first quantifier is second order) and ϕ is a first order formula. ϕ can be assumed to be of the form $\forall x \exists y : \psi(x, y, G, S)$, where ψ is quantifier-free.*

The class of those predicates which can be expressed without the second quantifier is called $\mathcal{SNP}$ or strict $\mathcal{NP}$.

G is the "input", S corresponds to the guess-string of a nondeterministic machine. Let us express SAT:

$$\exists T : \forall c \exists x : [(P(c, x) \wedge x \in T) \vee (N(c, x) \wedge x \notin T)],$$

where c denotes a clause, x a variable, T the set of true variables. P and N encode the instance: $P(c, x) = 1$ iff variable x appears positively in clause c, $N(c, x) = 1$ iff variable x appears negatively in clause c. Now to the class of optimization problems derived from this characterization.

Definition 4.2 *For each predicate $\Pi \in \mathcal{NP}$ of the form $\exists S \forall x \exists y : \psi(x, y, G, S)$ the maximization version of Π is defined as:*

$$\max_S |\{x | \exists y : \psi(x, y, G, S)\}|.$$

The class of such optimization problems is called MAX NP. *The corresponding class for the Π in $\mathcal{SNP}$ is called* MAX SNP.

In the case of MAX CUT the definition is very sensible. Consider "G is bipartite":

$$\exists C : \forall e = (u, v) : [(u \in C \wedge v \notin C) \vee (v \in C \wedge u \notin C)]$$

The maximization version is: maximize the number of external edges, i.e., maximize the cut.

Fact 4.6 *The following problems are* MAX SNP*-complete with respect to L-reductions (which are defined in [PapY91]):*

MAX k-SAT *for every* $k \geq 2$, INDEPENDENT SET *with constant degree,* MAX CUT *(and some interesting others).*

It was shown in [PapY91] that all problems in MAX NP can be approximated within a constant in polynomial time. We will later show how to do this in functional $\mathcal{TC}^0$. But first to the existence of approximation schemes.

Fact 4.7 *All problems that are* MAX SNP- *or* MAX NP*-hard via L-reductions have no PTAS unless* $\mathcal{P} = \mathcal{NP}$.

This was among an astonishing wave of results on interactive proof systems in 1989-1992 (for an overview see [J92]). It is proved in [AS92] and [ALMSS92]. The strategy is very interesting. First a model of probabilistically checkable proof systems is defined, then $\mathcal{NP}$ is characterized in terms of this model and afterwards related to approximation. A probabilistically checkable proof system consists of two parties: a prover with unbounded computational capabilities who provides a proof (e.g. that a graph is Hamiltonian) on a tape. This tape is accessible for the second party, the verifier, who has to work in polynomial time, accesses some of the proof (using randomness) and accepts or rejects it. The prover always tries to create convincing proofs, but has to work off-line, i.e., cannot be adaptive to questions of the verifier. The verifier accepts correct proofs always and rejects wrong ones with some constant probability.

A crucial idea is to bound the number of queries and the amount of randomness the verifier is allowed to use. The breakthrough was a characterization of $\mathcal{NP}$ as the set of languages that are verifiable with $O(\log n)$ random bits and $O(1)$ query bits to the proof. The verifier of the proof system for a $\mathcal{NP}$ language accesses only a constant number of bits drawn randomly from a polynomial length proof and is able to reject wrong proofs in polynomial time with constant probability, while accepting every correct proof.

The connection to approximation algorithms is easy to see: there is a proof system for any language in $\mathcal{NP}$. It is possible to decide the language (on some input) by finding out whether a proof exists such that all of the verifiers (distinguished by their choice of random bits) accept, or if a constant fraction of them rejects for every proof. To find this out one can form a Boolean formula of verifyers operating on the fixed input instance. The remaining input of every verifier is the portion of the proof it accesses. Since this portion contains only constantly many bits every verifier can be written as a constant size circuit. The verifiers use $O(\log n)$ random bits and so there are polynomially many of them. This yields a polynomial length formula of constant size circuits that is satisfiable fully (by some proof) if the input instance is in the $\mathcal{NP}$ language, and satisfiable only up to a constant fraction otherwise. Thus a PTAS for the problem to maximize the number of satisfied circuits in a formula of constant size circuits solves a $\mathcal{NP}$-complete problem. This generalizes to all MAX SNP-hard problems for the L-reductions of [PapY91].

Theorem 4.8 *All problems in* MAX NP *can be approximated within a constant in functional* $\mathcal{TC}^0$.

PROOF: We first show how to do this for MAX SNP problems, and then how to extend the algorithm to MAX NP. Let $\exists S \forall x : \varphi(x, G, S)$ be some predicate in $\mathcal{NP}$ and M be the maximization form of it. Then φ is a constant size formula on Boolean variables $S(z)$ and $G(w)$ where z and w are projections of the vector x, and G is the input structure. When G is fixed, then for every x the formula φ is a constant size Boolean formula on variables $S(z)$.

There are polynomially (in the length of G) many possible values of x, and thus the first order part of the predicate can be written as a polynomial size formula of conjuncts with each at most k variables (for some constant k): $\varphi_1 \wedge \cdots \wedge \varphi_m$. The fraction of assignments that satisfy φ_i is called $f_i \geq 2^{-k}$ (unsatisfiable φ_i are left out, so at least one assignment to the k variables satisfies φ_i). A random assignment to the variables $S(z)$ satisfies each φ_i with probability f_i, and thus satisfies expected $\sum f_i \geq m2^{-k}$ conjuncts, m is trivially an upper bound on the global optimum. This random assignment yields a constant ratio approximation algorithm (all up to here is from [PapY91]).

It is clear that such a random assignment can be produced in functional $\mathcal{RAC}^0$. We show now that k-wise independent randomness is sufficient for such an algorithm and afterwards how to derandomize this leading to a functional $\mathcal{TC}^0$ algorithm. To see that k-wise independence suffices consider the probability that a conjunct φ_i on $l \leq k$ literals $S_1, \ldots, S_l$ is satisfied (which is satisfiable by setting $S = v$ for some v) by a k-wise independent random assignment:

$$\begin{aligned} \text{prob}(\varphi_i(x) = 1) &\geq \text{prob}\left(\bigwedge_{j=1}^{l} S_j = v_j\right) \\ &= \prod_{j=1}^{l} \text{prob}(S_j = v_j) \\ &= 1/2^l \geq 1/2^k. \end{aligned}$$

Hence an algorithm that uses k-wise independent random variables is sufficient for a performance ratio $1/2^k = O(1)$-approximation. To derandomize the algorithm we show how to create n k-wise independent random variables from a polynomial size probability space that can efficiently be searched in parallel. The following lemma is from [ABI86].

Lemma 4.9 *Suppose* $n = 2^l - 1$ *and* $k = 2t + 1 \leq n$. *Then there exists a uniform probability space* Ω *of size* $2(n+1)^t$ *and* k*-wise independent random variables* $\xi_1, \ldots, \xi_n$ *over* Ω *each of which takes the values 0 and 1 with probability* $\frac{1}{2}$.

Let F be the field $GF(n+1)$ and denote the nonzero elements of F by column-vectors of length l: $x_1, \ldots x_n$. Consider the following $1 + lt$ by n matrix over $GF(2)$:

$$H = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ x_1 & x_2 & x_3 & & x_n \\ x_1^3 & x_2^3 & x_3^3 & \vdots & x_n^3 \\ \vdots & \vdots & \vdots & & \vdots \\ x_1^{2t-1} & x_2^{2t-1} & x_3^{2t-1} & \ldots & x_n^{2t-1} \end{pmatrix}.$$

This is the parity check matrix of the BCH code of length n and distance $2t + 2$ (see [Ad91]). It is well known that any $k = 2t+1$ columns of H are linearly independent. Now let $\Omega = \{1, 2, \ldots, 2(n+1)^t\}$ and A be the matrix where the rows are all the $2(n+1)^t = 2^{lt+1}$ linear combinations of the rows of H. Then the random variables $\xi_1 \ldots, \xi_n$ over Ω are defined by the 1 to nth entry of row i in A after choosing i from Ω. These random variables are k-wise independent (see [ABI86]).

To implement this on a $\mathcal{TC}^0$ circuit increase the number of variables $S(z)$ to the next power of 2, compute the value of the $S(z)$ for every possible choice from Ω and evaluate the number of satisfied φ_i for this choice. This can be done in $\mathcal{TC}^0$ due to fact 2.5. Then determine the choice/solution s that satisfies the most conjuncts of φ. Output this solution. Since this solution is the best of all it is clearly better than the expectation, and thus:

$$C(s, x) \geq E[C(s, x)] \geq 1/2^k m$$

Since finding a maximum is in $\mathcal{TC}^0$, the whole circuit is. The circuit can be constructed in functional $\mathcal{LOGSPACE}$ because H is computable in functional $\mathcal{LOGSPACE}$ given n, and the circuits from fact 2.5 can also be constructed in functional $\mathcal{LOGSPACE}$.

Now to the problems in MAX NP. Let $\exists S \forall x \exists y : \varphi(x, y, G, S)$ be some predicate in $\mathcal{NP}$ and M be its maximization form. For a fixed input G and some x_i the formula φ_i can either be satisfied by some y or not. Choose any y for which it is satisfiable if one exists. Now again at least one assignment to the at most k variables $S(z)$ satisfies φ_i, and the same algorithm works as above.

Note also that the x may carry arbitrarily large positive weights (as the weights in MAX CUT, and that the algorithm still works (producing an output of at least $1/2^k \sum w_i$ instead of $1/2^k m$). □

We have seen that MAX CUT can be approximated globally within a constant, and this very fast. Thus the problem is also easy for local approximation. An open question is whether MAX CUT has a NCLAS, i.e., an approximation scheme for local optima. The existence of a PTAS was excluded by the use of methods from the area of interactive proof systems. A generalization of these methods to local optimization seems to be difficult. Thus it remains as interesting open problem whether MAX CUT or other problems in MAX SNP that are hard for local search have fast parallel local approximation schemes.

4.3 The MAX CUT Problem with signed weights

When the MAX CUT problem is generalized to signed weights the difficulty appears what to do when all weights are negative and the problem collapses to MIN CUT. To keep the correspondence to PN-HOPFIELD as close as possible we do not define special vertex sets s and t as for MIN CUT, but allow a cut to be an arbitrary partition of the graph into two (even empty) sets. This implies that the global optimum is always nonnegative. The problem is equivalent to PN-HOPFIELD regarding (global and local) optimization complexity (see theorem 3.21).

The problem has not been investigated deeply. It is unknown if the problem can be approximated globally in polynomial time or locally in functional $\mathcal{NC}$. Very recently the algorithm of Goemans and Williamson [GoWi95] for MAX CUT has been applied to graphs with signed weights, yielding the following:

Fact 4.10 *Let* $W_- = \sum_{i<j} w_{ij}^-$, *where* $x^- = \min(0, x)$. *Then*

$$(E[W] - W_-) \geq \alpha \left(\frac{1}{2} \sum_{i<j} w_{ij}(1 - v_i \cdot v_j) - W_- \right),$$

where $E[W]$ *denotes the expected value of the cut obtained by the algorithm, the* v_i *are an optimal solution of the problem relaxed to the reals between* -1 *and* 1 *(and thus the sum* $\frac{1}{2}\sum_{i<j} w_{ij}(1 - v_i \cdot v_j)$ *is an upper bound on the globally optimal cut), and* $\alpha = .878$.

Of course this is not an approximation algorithm in the usual sense. We pose the question whether PN-MAX CUT can be approximated, and more generally, whether simple problems that are easy to approximate have efficient approximations when weights are allowed to be signed.

5 Conclusion

We have defined a new approach to cope with the hardness of optimization problems: approximation of local optima. This approach could be useful when a problem cannot be approximated globally and local search is hard, too. In the case of polynomially bounded optima local search is clearly possible in polynomial time, but one may ask for fast parallel algorithms. In the case of unbounded optima local search may be harder ($\mathcal{PLS}$-complete, though these problems tend to behave rather well in most practical situations), and one could look for polynomial time algorithms. Since the structural property of local optimality is seldom of interest local approximations might have some practical value. Most local search approaches to optimization, like simulated annealing, are heuristics to find good solutions, and do not search local optima for their own sake.

The notion of local approximability is also of some theoretical interest. The approximations are very weak (we demand from a local approximation algorithm to produce a solution almost as good as the worst nonnegative cost local optimum, see definition 2.16) and should allow very fast parallel algorithms in some cases. It would be interesting to find out whether the relation between local search and local approximation is analogous to that between global optimization and global approximation. A question related to this is whether MAX CUT has a local approximation scheme. It should be possible to create a theory parallel to the theory of global approximation investigating such questions in order to understand the nature of optimization problems more deeply.

One first step towards this is the introduction of a reduction. We proposed a special kind of "local performance ratio preserving reduction" (definition 2.17) and showed that complete local search problems under this reduction exist (theorem 2.15). These problems cannot be approximated locally more efficient than optimized locally (theorem 2.17). This was a non-surprising correspondence to the theory of global approximation. Intuitively there should be the same approximation degrees as in the case of global approximation: approximation schemes, constant approximations, etc. Unfortunately we were not yet able to find a result strengthening the belief in a concrete practical value of local approximation. There is a candidate for such a problem: the N-HOPFIELD function. This is a difficult problem for global optimization, global approximation, and local optimization (see section 3.2). It would be very nice to come up with an efficient local approximation algorithm for this problem.

After introducing the notion of local approximation we investigated the complexity of the Hopfield energy function under the four approaches global optimization, global approximation, local optimization, and local approximation. The consideration of this function was well motivated because local optima are of special interest here: the Hopfield energy function defines a neural network that determines local optima automatically and has widely been applied to optimization problems. We showed that the Hopfield energy function on the domain $\{0,1\}$ is very hard—it is complete for the class of all global maximization problems (in the case of unit weights for all global maximization problems with polynomially bounded optima) and complete for the $\mathcal{PLS}$ resp. $\mathcal{NC}^1\mathcal{LS}^{pol}$ maximization problems (unbounded resp. polynomially bounded optima) via approximability preserving reduc-

tions (theorem 3.20/23). This implies that the general Hopfield energy function is able to express all $\mathcal{NP}$ maximization problems such that local and global approximability is preserved. Additionally the Hopfield energy function defines neural networks performing local search—thus one can say that it is an interesting programming language of "full" complexity.

We also considered restricted versions of the general Hopfield energy function. The version with positive weights only is an easy to solve variant having a fast parallel global optimization algorithm (polynomial weights) and a polynomial time global optimization algorithm (unbounded weights). An interesting structural property of this function is that it is almost equally hard to optimize locally as to optimize globally. The version of the Hopfield energy function with negative weights only is more mysterious: it is very hard for all the three approaches we considered different from local approximation. A most interesting question is whether the problem can be approximated locally in $\mathcal{NC}$. Both restricted versions can be viewed as variants which allow a programming style adopted to the complexity of the problem one wants to model. The Hopfield energy function can be seen as an interesting flexible unification of optimization problems.

In section 4 we examined graph cut problems that are exactly as hard to optimize as the three versions of the Hopfield energy function. While s,t-MIN CUT and P-HOPFIELD possibly have the same approximation complexity this is wrong for MAX CUT and N-HOPFIELD: MAX CUT can be approximated globally within a constant in functional $\mathcal{TC}^0$, N-HOPFIELD cannot be approximated globally in polynomial time within n^ϵ for some constant $\epsilon > 0$ (theorems 4.8 and 3.17). A question arising from the consideration of a generalized MAX CUT problem is how well problems with signed weights can be approximated. This has not been discussed deeply (see [Ka92] for a catalogue of results on approximation complexity). One example of dramatically increased hardness is PN-HOPFIELD. This is equivalent to MAX 2-DNF with signed weights on its conjunctive clauses and only positive occurrences of variables. The normal MAX 2-DNF is a member of MAX SNP and can thus be approximated globally within a constant in functional $\mathcal{TC}^0$ due to theorem 4.8, while the problem with signed weights cannot be approximated efficiently at all (theorem 3.23).

There are many open questions worthy of further consideration:

- Has MAX CUT a $\mathcal{NC}$ local approximation scheme (NCLAS)?
- How well can PN-MAX CUT be approximated?
- Can s,t-MAX FLOW with exponential weights be approximated in $\mathcal{NC}$?
- Can the results in the table on page 25 be tightened?
  - Try to find a local approximation for N-HOPFIELD$^{|1|}$ or a hardness result for PN-HOPFIELD$^{|1|}$
  - Can P-HOPFIELD^{exp} be approximated (locally or globally) in $\mathcal{NC}$?
- Are there hard problems (for local search and global approximation) with efficient local approximations? We still have to come up with the answer.

A Definitions of Optimization Problems

Here we provide exact definitions of the optimization problems considered in this paper. $\mathcal{I}$ denotes the set of instances, S the set of feasible solutions, N the considered neighborhood, C the cost function, *opt* indicates whether C has to be minimized or maximized. The feasibility predicate P can easily be derived from S.

[1] P-HOPFIELD
$\mathcal{I} = \{H | H = \langle (w_{i,j})_{1 \leq i,j \leq n}, (t_i)_{1 \leq i \leq n} \rangle; w_{i,j} = w_{j,i} \geq 0\}$
$S(H) = \{0,1\}^n$
N =1-flip
$C(s,H) = \sum_{i<j} w_{i,j} s_i s_j - \sum_i t_i s_i$
opt = max

[2] N-HOPFIELD
$\mathcal{I} = \{H | H = \langle (w_{i,j})_{1 \leq i,j \leq n}, (t_i)_{1 \leq i \leq n} \rangle; w_{i,j} = w_{j,i} \leq 0\}$
$S(H) = \{0,1\}^n$
N =1-flip
$C(s,H) = \sum_{i<j} w_{i,j} s_i s_j - \sum_i t_i s_i$
opt = max

[3] PN-HOPFIELD
$\mathcal{I} = \{H | H = \langle (w_{i,j})_{1 \leq i,j \leq n}, (t_i)_{1 \leq i \leq n} \rangle; w_{i,j} = w_{j,i}\}$
$S(H) = \{0,1\}^n$
N =1-flip
$C(s,H) = \sum_{i<j} w_{i,j} s_i s_j - \sum_i t_i s_i$
opt = max

[4] s,t-MIN CUT
$\mathcal{I} = \{\langle G,s,t \rangle | G = (w_{(i,j)})_{1 \leq i,j \leq n}; w_{(i,j)} \geq 0; s,t \subseteq \{1,\ldots,n\}; s,t \neq \emptyset\}$
$S(G,s,t) = \{y \in \{0,1\}^n | \forall i \in s : y_i = 1, \forall i \in t : y_i = 0\}$
N =1-flip
$C(y, \langle G,s,t \rangle) = \sum_{(i,j)} w_{(i,j)} y_i (1 - y_j)$
opt = min

[5] MAX CUT
$\mathcal{I} = \{G | G = (w_{i,j})_{1 \leq i,j \leq n}; w_{i,j} = w_{j,i} \geq 0\}$
$S(G) = \{0,1\}^n - \{0^n, 1^n\}$
N =1-flip
$C(s,G) = \sum_{i<j} w_{i,j} [s_i(1 - s_j) + (1 - s_i) s_j]$
opt = max

[6] PN-MAX CUT

$\mathcal{I} = \{G|G = (w_{i,j})_{1\leq i,j\leq n}; w_{i,j} = w_{j,i}\}$

$S(G) = \{0,1\}^n$

N =1-flip

$C(s,G) = \sum_{i<j} w_{i,j}[s_i(1-s_j) + (1-s_i)s_j]$

opt = max

[7] s,t-MAX FLOW

$\mathcal{I} = \{\langle G,s,t\rangle | G = (w_{i,j})_{1\leq i,j\leq n}; w_{i,j} = -w_{j,i}; s,t \subseteq \{1,\ldots,n\}; s,t \neq \emptyset\}$

$S(G,s,t) = \{x|x = (x_{i,j})_{1\leq i,j\leq n}; \forall i,j : |x_{i,j}| \leq |w_{i,j}|; x_{i,j} > 0 \Rightarrow w_{i,j} > 0; x_{i,j} < 0 \Rightarrow w_{i,j} < 0; x_{i,j} = -x_{j,i}; \forall i \notin s \cup t : \sum_j x_{j,i} = \sum_k x_{i,k}\}$

$N_1(x,\langle G,s,t\rangle) = \{y|y \in S(G,s,t); y_{i,j} - x_{i,j} = z_{i,j} \in \{0,c,-c\}$; the $z_{i,j} > 0$ form a simple path from s to t in the graph of nonzero weighted edges$\}$. For this neighborhood ("augmenting paths") local implies global optimality (see [VL90]).

$N_2(x,\langle G,s,t\rangle) = \{y|y \in S(G,s,t); y_{i,j} - x_{i,j} = z_{i,j} \in \{0,c,-c\}$; the $z_{i,j} > 0$ form a simple path from s to t in the graph of positively weighted edges$\}$. Local optima for N_2 are called "blocking flows".

$C(x,\langle G,s,t\rangle) = \sum_{i\in s}\sum_j x_{i,j}$

$opt = max$

[8] INDEPENDENT SET

$\mathcal{I} = \{G|G = (V,E)$ is an undirected graph$\}$

$S(G) = \{s \in \{0,1\}^n | s_i = s_j = 1 \Rightarrow \{s_i,s_j\} \notin E\}$

N =1-flip

$C(s,G) = \sum_i s_i$

opt = max

[9] LONGEST PATH WITH FORBIDDEN PAIRS

$\mathcal{I} = \{\langle G,P\rangle | G = (V,E)$ is a directed graph; $P \subseteq V^2\}$

$S(G,P) = \{(i_1,\ldots,i_k)|$ the vertices v_{i_j} form a simple path in G without traversing both vertices of a pair in $P\}$

$C((i_1,\ldots,i_k),\langle G,P\rangle) = k$

opt = max

[10] MAX SAT

$\mathcal{I} = \{F|F = \langle V,D\rangle; V$ is a set of variables, D a set of disjunctive clauses of negated and unnegated variables$\}$

$S(F) = \{D' \subseteq D|$ an assignment to V exists so that D' is satisfied $\}$

N =1-flip

$C(D',F) = |D'|$

opt = max

[11] MAX K-SAT

$\mathcal{I} = \{F | F = \langle V, D \rangle; V$ is a set of variables, D a set of disjunctive clauses of negated and unnegated variables having length at most k$\}$

$S(F) = \{D' \subseteq D |$ an assignment to V exists so that D' is satisfied $\}$

N =1-flip

$C(D', F) = |D'|$

opt = max

[12] MAX DNF

$\mathcal{I} = \{F | F = \langle V, D \rangle; V$ is a set of variables, D a set of conjunctive clauses of negated and unnegated variables$\}$

$S(F) = \{D' \subseteq D |$ an assignment to V exists so that D' is satisfied $\}$

N =1-flip

$C(D', F) = |D'|$

opt = max

[13] MAX K-DNF

$\mathcal{I} = \{F | F = \langle V, D \rangle; V$ is a set of variables, D a set of conjunctive clauses of negated and unnegated variables having length at most k$\}$

$S(F) = \{D' \subseteq D |$ an assignment to V exists so that D' is satisfied $\}$

N =1-flip

$C(D', F) = |D'|$

opt = max

[14] MAX CIRCUIT OUTPUT

$\mathcal{I} = \{T | T$ is circuit made of fan-in 2 and fan-out 2 AND/OR/NOT gates, T has p inputs and q outputs out_i^T.$\}$

$S(T) = \{0,1\}^p$

N =1-flip

$C(s, T) = \sum_{i=0}^{q-1} out_i^T(s) 2^i$

opt = max

If the depth is restricted to $\log^c |T|$ then the problem is called $\mathcal{NC}^c$-MAX CIRCUIT OUTPUT.

The local optimization version of these problems is called [$\mathcal{NC}^c$-] FLIP.

If $q \leq k \cdot \log |T|$ then the local optimization version is called [$\mathcal{NC}^c$-] $\text{FLIP}_{k \cdot \log}$.

References

[Ad91] J. Adámek. Foundations of Coding. Wiley-Interscience, 1991.

[ABI86] N. Alon, L. Babai, A. Itai. A fast and simple randomized algorithm for the maximal independent set problem. *J. of Algorithms*, vol.7, pages 567–583, 1986.

[AS92] S. Arora, S. Safra. Probabilistic Checking of Proofs; A New Characterization of NP. *Proc. 33rd IEEE Symp. on Found. of Comp. Science*, pages 2–13, 1992.

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy. Proof Verification and Hardness of Approximation Problems. *Proc. 33rd IEEE Symp. on Found. of Comp. Science*, pages 14–23, 1992.

[BeSc92] P. Berman, G. Schnitger. On the Complexity of Approximating the Independent Set Problem. *Information and Computation*, vol.96, pages 77–94, 1992.

[Co92] E. Cohen. Approximate Max Flow on Small Depth Networks. *Proc. 33rd IEEE Symp. on Found. of Comp. Science*, pages 648–658, 1992.

[Fa74] R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. *Complexity of Computations*, AMS 1974.

[GJ79] M.R. Garey, D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, 1979.

[GoWi94] M.X. Goemans, D.P. Williamson. .878-Approximation Algorithms for MAX CUT and MAX 2-SAT. *Proc. 26th ACM Symp. on Theory of Computing*, pages 422-431, 1994.

[GoWi95] M.X. Goemans, D.P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. Unpublished.

[Gol92] M. Goldmann. On Threshold Circuits and Related Models of Computation, Dissertation, NADA Stockholm, 1992.

[HKP91] J. Hertz, A. Krogh, R.G. Palmer. Introduction to the Theory of Neural Computation. Addison-Wesley, 1991.

[Ho82] J.J. Hopfield. Neural Networks and Physical Systems having Emergent Collective Computational Abilities, 1982. *Reprinted in* Anderson, Rosenfeld: *Neurocomputing: Foundations of Research*, Cambridge, MIT-Press, 1988.

[HoTa85] J.J. Hopfield, D.W. Tank. "Neural" Computation of Decisions in Optimization Problems. *Biological Cybernetics*, vol.52, pages 141–152, 1985.

[J90] D.S. Johnson. A Catalogue of Complexity Classes. *Handbook of Theoretical Computer Science A*, pages 67–161, Elsevier 1990.

[J92] D.S. Johnson. The NP-Completeness Column: An Ongoing Guide. *Journal of Algorithms*, vol.13, pages 502–524, 1992.

[JPaY88] D.S. Johnson, C.H. Papadimitriou, M. Yannakakis. How Easy is Local Search? *Journal of Computer and System Sciences*, vol.37, pages 79–100, 1988.

[Ka92] V. Kann. On the Approximability of NP-complete Optimization Problems. Dissertation, NADA Stockholm, 1992.

[KarRa90] R.M. Karp, V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. *Handbook of Theoretical Computer Science A*, pages 869–941, Elsevier 1990.

[KiGV83] S. Kirkpatrick, C. Gelat, M. Vecchi. Optimization by simulated annealing. *Science*, vol.220, pages 671–680, 1983.

[Kr90] M.W. Krentel. On Finding and Verifying Locally Optimal Solutions. *SIAM Journal Comput.*, vol.19, pages 742–749, 1990.

[LiKe73] S. Lin, B. Kernighan. An efficient heuristic for the traveling salesman problem. *Oper. Res.*, vol.21, pages 498–516, 1973.

[Lu86] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal Comput.*, vol.15, pages 1036–1053, 1986.

[McPi43] W. S. McCulloch, W. Pitts. A logical Calculus of Ideas immanent in Nervous Activity. *Bulletin of Mathematical Biophysics 5*, pages 115–133, 1943. *Reprinted in* Anderson, Rosenfeld: *Neurocomputing: Foundations of Research*, Cambridge, MIT-Press, 1988.

[Mu71] S. Muroga. Threshold Logic and its Applications. Wiley-Interscience, 1971.

[Pan85] V. Pan. Fast and Efficient Parallel Algorithms for the Exact Inversion of Integer Matrices. *Proc. 5th Ann. Conf. on Foundations of Software Technology and Theoretical Computer Science.* LNCS 206, 1985.

[Pap92] C.H. Papadimitriou. The Complexity of Lin-Kernighan. *SIAM Journal Comput.*, vol.21, pages 450–465, 1992.

[PapSt82] C.H. Papadimitriou, K.Steiglitz. Combinatorial Optimization, Algorithms and Complexity. Prentice Hall, 1982.

[PapSY90] C.H. Papadimitriou, A.A. Schäffer, M. Yannakakis. On the Complexity of Local Search. *Proc. 22th ACM Symp. on Theory of Computing*, pages 438-445, 1990.

[PapY91] C.H. Papadimitriou, M. Yannakakis. Optimization, Approximation, and Complexity Classes. *Journ. of Comp. and Syst. Sciences*, vol.43, pages 425–440, 1991.

[Par94] I. Parberry. Circuit complexity and Neural Networks. MIT-Press, 1994.

[SchY91] A.A. Schäffer, M. Yannakakis. Simple Local Search Problems that are Hard to Solve. *SIAM Journal Comput.*, vol.20, pages 56–87, 1991.

[Sim90] H.U. Simon. On approximate solutions for combinatorial optimization problems. *SIAM Journal Disc. Math.*, vol.3, pages 294–310, 1990.

[SBKH93] Siu, Bruck, Kailath, Hofmeister. Depth Efficient Neural Networks for Division and Related Problems. *IEEE Trans. Information Theory* 39, pages 946–956, 1993.

[SRo94] Siu, Roychowdhury. On Optimal Depth Threshold Circuits for Multiplication and Related Problems. *SIAM J. Discrete Math.*, vol. 7, pages 284–292, 1994.

[VL90] J. van Leeuwen. Graph Algorithms. *Handbook of Theoretical Computer Science A*, pages 525–631, Elsevier 1990.

[We87] I. Wegener. The Complexity of Boolean Functions. Wiley-Interscience, 1987.