A STRUCTURAL VIEW OF APPROXIMATION

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Sanjeev Khanna

August 1996

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Rajeev Motwani
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Serge Plotkin

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Madhu Sudan

Approved for the University Committee on Graduate Studies:

Abstract

The discovery of efficient probabilistically checkable proofs for NP, and their surprising connection to hardness of approximation, has resulted in a quantum leap in our understanding of approximability of many important optimization problems. Yet much of this research has stayed focused on problem-specific results. Building on many such results, this dissertation develops frameworks which unify a variety of seemingly different results and thereby highlights the intrinsic structural properties which govern the approximability of optimization problems.

Broadly speaking, our work comprises of three distinct parts. In the first part, we develop a structural basis for computationally-defined approximation classes such as APX (constant-factor approximable problems) and poly-APX (polynomial-factor approximable problems). We show that there exist canonical transformations whereby every problem in an approximation class can be "expressed" as a problem in a syntactically-defined optimization class. Thus, for example, we show that every problem in APX is expressible as a problem in the syntactic class MAX SNP. These results serve to reconcile two distinct views of approximability and provide a better understanding of the structure of the computational classes. An immediate consequence of these results is the discovery that a wide variety of natural optimization problems are complete for the computational classes. We also study the possibility that the core of PTAS (problems with polynomial-time approximation schemes) may be characterized via a combination of syntactic constraints and restrictions on the input instances.

In the second part, we study two natural classes of maximization problems defined via constraint satisfaction requirements. We discover classification theorems which, quite surprisingly, allow us to determine the approximability of every problem in both these classes merely from the description of the problem. Our results serve to formally affirm many trends concerning the approximability of

optimization problems which have emerged over the years. In particular, they provide insight into questions such as: Why are there no natural maximization problems that exhibit a $\log n$ threshold for approximability? Why do all the known MAX SNP problems that are NP-hard, turn out also to be MAX SNP-hard? and, so on.

Finally, in the last part of this work, we study the problem of efficiently coloring a 3-colorable graph. Since Karp's original NP-hardness result for this problem, no better lower bounds have been obtained towards the approximability of this fundamental optimization problem. We show that it is NP-hard to color a 3-colorable graph with 4 colors and thereby establish that a k-colorable graph cannot be colored with less than $k + 2\lfloor k/3 \rfloor$ colors, unless P $=$ NP.

Acknowledgements

First and foremost, I would like to express my sincere thanks to my advisor Rajeev Motwani for his continuous guidance and support. I have learned a lot from his creative style of research and his persistence in tackling difficult research problems. It has been a privilege to work under his close supervision.

Many thanks to Madhu Sudan who has been not only a partner in several of my research efforts but also a friend, mentor and a constant source of inspiration.

Thanks also to Muli Safra for being a very encouraging and patient collaborator in my starting days, and for teaching me many things in complexity theory.

I have had the great fortune of learning from many great teachers. In particular, I would like to thank Ed Reingold, Doug West, Rajeev Motwani, and Serge Plotkin, for all the fascinating algorithms, combinatorics, and graph theory courses they have taught me. I would also like to thank them all for generously giving me their valuable advice on many occasions. Thanks are also due to Serge for being on my thesis reading committee.

Very special thanks to Julien Basch and Chandra Chekuri who have been constant partners in my academic and non-academic pursuits over the last several years. My life has been deeply enriched by their wonderful friendship.

My stay at Stanford would not have been such a pleasant experience without the great companionship of my colleagues. I would like to thank Anand, Arjun, Ashish, Craig, David, Donald, Eric, Harish, Jeffrey, Luca, Michael, Moses, Omri, Piotr, Robert, Stevens, and Suresh, for being delightful colleagues and for keeping me stimulating company over the years.

I am truly indebted to my mother, father and brother, for their love, support and encouragement from thousands of miles away.

Finally, I would like to thank my wonderful wife Delphine for always being there for me.

Contents

Chapter 1

Introduction

Over the last three decades, a large number of naturally-arising optimization problems have been shown to be NP-hard. Some classic examples of such problems include multiprocessor scheduling problems arising in systems, constraint satisfaction problems arising in AI, graph coloring problems arising in compiler optimization, and the well-known traveling salesman problem. Unless P = NP, there do not exist polynomial-time algorithms to solve these problems optimally. Despite the intensive efforts of many researchers, there has been little progress towards solving the NP-hard problems in polynomial-time. In fact, in the early 70's itself, it was widely conjectured that P $\neq$ NP. Since polynomial-time computability is commonly regarded as a pre-requisite in considering an optimization problem as computationally tractable, researchers have considered relaxed versions of these problems so as to make them provably tractable. One such well-studied relaxation has been to give up the optimality condition, i.e., to allow approximate solutions. The standard measure used to assess the quality of approximation is called the *approximation ratio*. Informally, an approximation ratio measures the multiplicative factor which separates the optimal and the approximate solution. In general, the objective is to find polynomial-time algorithms with approximation ratio close to one.

Sustained research over the years has significantly enhanced our understanding of NP-hard optimization problems from an algorithmic as well as a complexity-theoretic point of view. On the one hand, we have polynomial-time algorithms for a variety of problems with *provably good* approximation ratios. On the other hand, we now also have matching bounds, commonly referred to as *hardness of approximation* results, which show that these ratios cannot be further improved unless P = NP (in which case, they can be in fact solved optimally). This research has brought to the fore some surprising facts and intriguing trends. We mention some of the most important ones along with the issues surrounding them. First, even though the decision problems underlying all known NP-hard optimization problems are polynomially isomorphic, these optimization problems themselves exhibit a wide spectrum of behavior with respect to polynomial-time approximability. On the one end of this spectrum, there are problems such as multiprocessor scheduling which can be approximated to within any constant strictly greater than one. On the other end of the spectrum, we have problems such as graph coloring where we cannot approximate the optimal solution value to within even certain polynomial factors. Is there an underlying structure which unifies all problems

with similar approximability? Second, "natural" optimization problems seem to exhibit only few distinct types of approximability. For instance, we do not know of any natural maximization problem which is approximable to within logarithmic factors and yet is not approximable to within some constant factor. In fact, it seems consistent with our current understanding of approximability to conjecture the non-existence of such problems. While there are several impediments to a formal investigation of such a thesis in the general setting, can its truth be validated for any class of natural problems which is well-structured and yet general enough to capture representative problems from the entire spectrum of NP optimization problems? Third, the emergence of several widely applicable algorithmic paradigms such as the primal-dual method [44] and the randomized rounding technique [92] has highlighted that a suitable syntactic expressibility of a general class of problems can be translated into a canonical positive approximability result. However, with the exception of just one result[1], there are no known canonical hardness results which apply to an entire class of problems. Such results are critical to our understanding of intrinsic structure governing the approximability of seemingly different problems. Are there other large classes of naturally-arising optimization problems where syntactic expressibility can be used to uniformly distinguish "hard" problems from "easy" ones? These are the issues which motivate the work of this dissertation.

Our work makes progress towards addressing each of the questions raised above. The next two sections provide the necessary background and a formal overview of our results. However, a brief informal description of some representative results follows. We show that all problems with similar approximability can be expressed in a canonical manner. This canonical representation is the intrinsic structural property shared by all of them. For example, we show that every problem with a constant-factor approximation can be "expressed" as the MAX 3-SAT problem. We study the approximability of maximization problems built out of constraint satisfaction requirements. Our results yield a classification theorem whereby the precise asymptotic approximability of every problem defined in this manner, can be determined merely from the problem description. The classification theorem gives a finite approximation hierarchy for the problems in these classes and, moreover, the levels of these hierarchies correspond precisely to those observed for naturally-arising

[1]This is the well-known result of Lund and Yannakakis where they studied hardness of approximating maximum subgraph for hereditary graph properties [81].

optimization problems. This theorem formally affirms the trends which seem to emerge from the study of optimization problems over the last two decades.

Much of the work in this dissertation builds upon the recent results which established a remarkable connection between the existence of efficient probabilistically checkable proofs and the approximability of optimization problems [37, 8, 7]. These results have led to a quantum leap in our understanding of the hardness of approximation. Based on these largely problem-specific results, our work has developed general results which apply to entire classes of optimization problems, thereby underscoring the inherent structure governing the approximability of these optimization problems.

1.1 The Historical Development

This section reviews the basic concepts through a historical overview of the results most intimately tied to our work. The historical development presented here serves to set up the conceptual foundations and the context for the work described in this dissertation.

1.1.1 The Theory of NP-Completeness

We start with a formal definition of the class NP; some preliminaries follow. An alphabet is a finite set of symbols, commonly denoted by Σ. A *string* is a finite sequence of symbols drawn from an alphabet. The length of a string w is denoted by $|w|$. A language L is a set of strings drawn over a fixed alphabet, i.e., $L \subseteq \Sigma^*$.

Definition 1 (NP) *A language $L \subseteq \Sigma^*$ belongs to* NP *if there exists a polynomial-time deterministic Turing machine M and a constant c such that for all $x \in \Sigma^*$, we have*

- $x \in L \Rightarrow \exists y \in \Sigma^*$ *with* $|y| = O(|x|^c)$ *such that M accepts the input pair (x, y).*
- $x \notin L \Rightarrow \forall y \in \Sigma^*$, *$M$ rejects the input pair (x, y).*

A natural interpretation of the above definition is as follows: the string y is the *proof* to the assertion that $x \in L$ and the machine M is the *verifier* of this proof.

The theory of NP-completeness was originally developed to study decision problems. Cook [25] and Levin [76] independently established that all NP-languages are "reducible" to the 3-SAT problem[2] — the problem of deciding if a given 3-SAT formula is satisfiable or not. Soon thereafter, Karp [57] showed that the 3-SAT problem in turn, is "equivalent" to a whole variety of natural NP decision problems. The notion of reduction and equivalence are formalized in the definitions below.

Definition 2 (Polynomial-Time Reduction) *A language $L_1 \subseteq \Sigma_1^*$ is* polynomial-time reducible *to another language $L_2 \subseteq \Sigma_2^*$ (denoted $L_1 \propto_P L_2$) if there exists a polynomial-time computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that for all $x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$.*

This notion of reduction is called *polynomial-time many-one reduction* or *Karp reduction* in the literature. A language L_1 is *polynomial-time equivalent* to another language L_2 if $L_1 \propto_P L_2$ and $L_2 \propto_P L_1$.

Definition 3 (NP-Hard) *A language L is* NP-hard *if for any language $L' \in$ NP, $L' \propto_P L$.*

Definition 4 (NP-Complete) *A language L is* NP-complete *if $L \in$ NP and L is* NP-*hard*.

The Cook-Levin result established that the 3-SAT problem is NP-complete. Karp showed that many important NP decision problems such as determining whether a graph is k-colorable, verifying the hamiltonicity of a graph, or testing if a graph has a clique of size at least k, are all reducible to the 3-SAT problem and hence NP-hard. Thus, either all these problems are *hard* (i.e., P $\neq$ NP), or they are all *easy* (i.e., P $=$ NP). Simultaneously, it was also realized that NP-completeness theory provides a framework to study the complexity of many naturally-arising optimization problems. These two developments had a remarkable impact in stimulating the theory of NP-completeness and establishing the importance of the P $=$ NP question. The class of optimization problems studied via the NP decision problems is called NPO, an acronym for NP optimization problems.

Definition 5 (NPO) *An* NPO *problem Π is a four-tuple $\langle \mathcal{D}, \mathcal{S}, V, goal \rangle$ such that:*

- *$\mathcal{D}$ is the set of input instances and it is recognizable in polynomial-time,*

[2] See Appendix A for a formal definition of various well-known problems that we will refer to in this dissertation.

- *Given any input $x \in \mathcal{D}$, $\mathcal{S}(x)$ denotes the set of feasible solutions such that there exists a polynomial p with $|y| \leq p(|x|)$ for all $y \in \mathcal{S}(x)$. Moreover, there exists a polynomial-time computable predicate π such that $\pi(x, y)$ holds if and only if $y \in \mathcal{S}(x)$.*
- *Given $y \in \mathcal{S}(x)$, $V(x, y)$ denotes the objective function value and is polynomial-time computable.*
- *Finally,* goal $\in \{\max, \min\}$ *denotes whether* Π *is a maximization or a minimization problem.*

The NPO problems are reduced to decision problems by considering the feasibility of solutions which satisfy a given bound on the objective function value. The resulting decision problem is at least as hard as the original optimization problem, and in fact often the complexity of the two problems is related by only a polynomial factor. Over the years, more and more optimization problems have been shown to have an associated NP-complete decision problem. Thus, NP-completeness theory became a vehicle for establishing the intractability of NPO problems, modulo the assumption P $\neq$ NP. But of course, this methodology required that the optimization problems first give up their essential character of being optimization problems!

1.1.2 The Advent of Approximation Algorithms

All attempts to resolve the P $=$ NP question thus far have been in vain. In the early 70's itself, it began to be widely conjectured that these two classes are distinct and that it is unlikely that we have polynomial-time *exact* algorithms for the NPO problems. Since many of these problems are well-rooted in practice, computational tractability is an important objective. It was only natural that researchers begin considering relaxation of these problems to obtain provably tractable versions of these problems. The primary such relaxation considered was that of relaxing the optimality condition. In particular, the focus shifted to polynomial time algorithms to compute *near-optimal* solutions, i.e., solutions which are guaranteed to be within a certain multiplicative factor of the optimal solution.

Definition 6 (Approximation Algorithm) *An algorithm A is an approximation algorithm for a* NPO *problem* Π *if given an input instance $\mathcal{I}$, it computes a feasible solution S for the input $\mathcal{I}$.*

Of course, the value of an arbitrary feasible solution can be far from the optimal value. Our interest is in algorithms which return solutions with guaranteed near-optimal values. The following definitions help us characterize the performance of approximation algorithms. We use the notation OPT($\mathcal{I}$) to denote the optimum objective function value on instance $\mathcal{I}$ of a problem. Throughout this dissertation, we study only polynomial-time approximation algorithms.

Definition 7 (Performance Ratio) *An approximation algorithm A for an optimization problem* Π *has* performance ratio $\mathcal{R}(n)$ *if, given any instance $\mathcal{I}$ of* Π *with $|\mathcal{I}| = n$, the solution $A(\mathcal{I})$ satisfies*

$$\max\left\{\frac{V(\mathcal{I}, A(\mathcal{I}))}{\text{OPT}(\mathcal{I})}, \frac{\text{OPT}(\mathcal{I})}{V(\mathcal{I}, A(\mathcal{I}))}\right\} \leq \mathcal{R}(n).$$

A solution of value within a multiplicative factor $\mathcal{R}(n)$ of the optimal value is referred to as an $\mathcal{R}(n)$-approximation. *We say that a* NPO *problem is approximable to within a factor $\mathcal{R}(n)$ if it has a polynomial-time approximation algorithm with performance ratio $\mathcal{R}(n)$.*

In a seminal paper, Johnson [58] studied the polynomial-time approximability of many NPO problems. He discovered that while good polynomial-time approximations are possible for several NP-hard optimization problems, there are many others which seem to resist efforts towards such algorithms. The subsequent work of several other researchers has only served to confirm the trend observed by Johnson. We now have a whole variety of results which show that many NP-hard optimization problems cannot be well-approximated unless P = NP. More importantly, these results have served to formally confirm the *diverse approximability* of the NP-hard optimization problems. This diverse behavior might appear somewhat surprising at first glance because the decision problems underlying these optimization problems were not only polynomially reducible to each other but were also *polynomially isomorphic*[3]. But clearly, the reductions among the decision problems were not strong enough to preserve the quality of approximate solutions. There was a distinct need for developing a framework which allowed us to study optimization problems *without having to deprive them of their essential character* by disguising them as decision problems.

[3]Two languages $L_1, L_2 \subseteq \Sigma^*$ are said to be polynomially isomorphic if there is a polynomial-time computable *bijection* h from Σ^* to itself such that $x \in L_1$ if and only if $h(x) \in L_2$. Hartmanis and Berman [51] showed in 1976 that all known NP-complete problems are polynomially isomorphic.

Meanwhile, a *behavioral* classification scheme emerged for the NPO problems — one which recognized their differences in terms of approximability.

1.1.3 The Computational view of Approximation

This classification scheme was implicit in Johnson's work — NPO problems are assigned to various classes based on their polynomial-time approximability. Thus, we have classes such as APX and PTAS.

Definition 8 (APX) *An* NPO *problem* Π *is in the class* APX *if there exists a polynomial-time algorithm for* Π *whose performance ratio is bounded by a constant.*

Definition 9 (PTAS) *An* NPO *problem* Π *is in the class* PTAS *if for any rational* $\epsilon > 0$*, there exists a polynomial-time approximation algorithm for* Π *whose performance ratio is bounded by* $(1+\epsilon)$.

If we let F-APX denote the class of NPO problems that are approximable to within a factor F, then we obtain a *hierarchy* of approximation classes. For instance, poly-APX and log-APX are the classes of NPO problems which have polynomial-time algorithms with performance ratio bounded polynomially and logarithmically, respectively, in the input length.

The strength of this classification scheme lies in its ability to capture all problems with a specified threshold of approximability. But this view of approximation is an "abstract view"; it does not highlight the inherent structural properties which determine the approximability of the NPO problems.

We will often be interested in restricting our attention to polynomially bounded subsets of these classes.

Definition 10 (Polynomially Bounded Subset) *The* polynomially bounded subset *of class* $\mathcal{C}$ *of problems, denoted* $\mathcal{C}$*-PB, is the set of all problems* $\Pi \in \mathcal{C}$ *for which there exists a polynomial* $p(n)$ *such that for all instances* $\mathcal{I} \in \Pi$, $\mathrm{OPT}(\mathcal{I}) \leq p(|\mathcal{I}|)$.

Thus, for example, APX-PB denotes the class of NPO problems that are constant-factor approximable and have polynomially-bounded optimum values.

1.1.4 Syntactic Characterizations of Decision Problems

The notion of complexity seems inherently linked to computational models and resource bounds on space and time. But surprisingly, almost parallel to the development of the computation-oriented view of NP-completeness theory, logicians were successfully developing a *descriptive* or *syntactic* view of complexity classes. Such efforts are rooted in a very simple intuition, namely, a "hard to decide" problem is probably also "hard to describe", and vice versa. A classical result in this direction was established by Fagin [34].

Theorem 1 ([34]) NP *is precisely the set of decision problems that can be expressed in the existential second order logic, that is as* $\exists\, S, \Phi(\mathcal{I}, S)$ *where* Φ *is a first-order formula,* S *is a structure (i.e. it is a second-order variable which ranges over relations of a fixed arity over the input universe) whose existence is to be decided, and* $\mathcal{I} = (U; \mathcal{P})$ *is the input structure comprising of a universe* U *and a finite set of constant arity predicates* $\mathcal{P}$.

This is a remarkable theorem. A complexity class which was defined in terms of Turing machine computations performed in a certain amount of time can also be described with no reference to either machines or resources. This characterization clearly tells us the structure of the NP languages. Using this characterization of NP, it is a relatively intuitive task to establish 3-SAT as NP-complete. The ease of discovering natural complete problems is a very desirable attribute which seems readily available with syntactically-defined classes, and becomes a difficult task (if not infeasible), when a class is behaviorally-defined.

1.1.5 Syntactic Optimization Classes

Over the last two decades, pioneering work due to Immerman and others has led to such syntactic characterizations of many other classes, e.g., PSPACE and NSPACE (see the introductory survey [55]). But this perspective stayed essentially in the realm of decision problems for a long time. It was as recently as 1988 that Papadimitriou and Yannakakis in a seminal paper [87] brought this view to the study of optimization problems. They were motivated by a collection of NPO problems which were known to be in APX, but whose membership in PTAS remained unresolved. It was

natural to ask if the question of their membership in PTAS could be reduced to a single question just as the P = NP question could be reduced to the question of the membership of 3-SAT in P.

There were two fundamental obstacles in achieving such a goal. First, how do we define a class which naturally contains all these problems and yet is restrictive enough to essentially capture only the relevant problems? Secondly, what notion of reductions would allow us to establish an equivalence between these problems in terms of their approximability? Papadimitriou and Yannakakis observed that efforts to computationally define such a class are not likely to succeed. In their own words: "*The intuitive reason is that computation is an inherently unstable, non-robust mathematical object, in the sense that it can be turned from non-accepting by changes that would be insignificant in any reasonable metric — say, by flipping a single state to accepting ..... There seems to be no clear notion of "approximately correct computaion". If we move the focus of approximability to, for example, the number of correct bits in the input (so that the machine accepts), then there seems to be no generic reduction that preserves approximability.* " Thus intuitively it seemed clear that one needed to take a different approach towards defining such a class. The only other such approach known was the syntactic approach hitherto applied to characterizing classes of decision problems. Indeed, they discovered an elegant answer using the syntactic approach.

Papadimitriou and Yannakakis began with Fagin's syntactic characterization of NP. In a standardized form, Fagin's result may be stated as follows: *A decision problem belongs to the class* NP *if and only if it can be expressed as* $\exists S, \forall \vec{x}, \exists \vec{y}, \phi(\mathcal{I}, S, \vec{x}, \vec{y})$ *where* ϕ *is a quantifier-free first order formula,* S *is the desired structure,* $\mathcal{I} = (U; \mathcal{P})$ *is the input structure, and* $\vec{x}$ *and* $\vec{y}$ *are finite vectors (i.e. they range over tuples in the universe* U *of the input structure).* Let us better understand this definition through a concrete example.

Example 1 (SAT) *The structure that we seek here is a truth assignment and we would like to verify that it satisfies all the given clauses. The input consists of a universe* U *whose elements are the clauses and the variables, and a set of three predicates* C, P, *and* N. *The unary predicate* C *is true if and only if a given element is a clause. The other two predicates* P *and* N *are binary, where* $P(c, v)$ *is true if and only if the clause* c *contains the variable* v *as a positive literal, and* $N(c, v)$ *is true if and only if the clause* c *contains the variable* v *as a negative literal. Now, the satisfiability*

problem may be expressed as $\exists S \subseteq X, \forall x, \exists\, y, \phi\left((U; \{C, X, P, N\}), S, x, y\right)$ *where*

$$\phi\left((U; \{C, X, P, N\}), S, x, y\right) \;=\; \overline{C(x)} \vee \Big((P(x, y) \wedge S(y)) \vee (N(x, y) \wedge \overline{S(y)})\Big)\,.$$

A natural subclass of NP, called *strict* NP or SNP, can be defined from the above characterization simply by dropping the existential quantifier, that is, SNP is the class of properties expressible as $\exists S, \forall \vec{x}, \phi(I, S, \vec{x})$. A canonical example of a problem in this subclass is 3-SAT.

Example 2 (3-SAT) *As before, the structure that we seek here is a truth assignment and we would like to verify that it satisfies all the given clauses. The role of the existential quantifier in the previous example is simply to verify that at least one literal in each clause is set to true. But in* 3-SAT*, the clauses have at most three literals each and hence the role of the existential quantifier can be pushed into the fixed-size formula* ϕ*. More formally, we encode the input instance via four predicates* $C_0, C_1, C_2,$ *and* C_3*, where* C_i *evaluates to true on a clause if and only if it has exactly* i *negated literals. Thus, if the clause* $c = (x_1, x_2, x_3) \in C_i$*, then the first* i *variables in* c *appear negated and the remaining variables appear unnegated. The* 3-SAT *problem may now be expressed as* $\exists S \subseteq X, \forall (x_1, x_2, x_3), \phi\left((U; \{C_0, C_1, C_2, C_3\}), S, (x_1, x_2, x_3)\right)$*, where*

$$\begin{aligned}
\phi\left((U; \{C_0, C_1, C_2, C_3\}), S, (x_1, x_2, x_3)\right) \;=\;& \big((x_1, x_2, x_3) \in C_0 \wedge (S(x_1) \vee S(x_2) \vee S(x_3))\\
\bigvee\;& \Big((x_1, x_2, x_3) \in C_1 \wedge (\overline{S(x_1)} \vee S(x_2) \vee S(x_3))\Big)\\
\bigvee\;& \Big((x_1, x_2, x_3) \in C_2 \wedge (\overline{S(x_1)} \vee \overline{S(x_2)} \vee S(x_3))\Big)\\
\bigvee\;& \Big((x_1, x_2, x_3) \in C_3 \wedge (\overline{S(x_1)} \vee \overline{S(x_2)} \vee \overline{S(x_3)})\Big)\,.
\end{aligned}$$

Now, what if we relax the condition that the structure must satisfy the formula ϕ for all choices of $\vec{x}$? That is, we look for structures which maximize the number of choices of $\vec{x}$ for which the formula is satisfied. This gives us an optimization analog of the classes SNP and NP.

Definition 11 (MAX SNP) *An optimization problem* Π *is in* MAX SNP *if it can be expressed as*

the problem of finding a structure S which maximizes the objective function

$$V(\mathcal{I}, S) = |\{\vec{x} \ : \ \phi(\mathcal{I}, S, \vec{x})\}|,$$

where $\mathcal{I}$ is the input and ϕ is a quantifier-free first order formula.

Example 3 (MAX 3-SAT) *The formula $\phi()$ is the same as in Example 2. But the goal is now to find S such that $|\{(x_1, x_2, x_3) \ : \ \phi\left((U; \{C_0, C_1, C_2, C_3\}), S, (x_1, x_2, x_3)\right)|$ is maximized.*

A natural extension is to associate a weight with every tuple in the range of the universal quantifier; the modified objective is to find an S which maximizes $V(\mathcal{I}, S) = \sum_{\vec{x}} w(\vec{x})\mathbf{\Phi}(\mathcal{I}, S, \vec{x})$, where $w(\vec{x})$ denotes the weight associated with the tuple $\vec{x}$.

The optimization analog of the class NP is the class MAX NP as defined below.

Definition 12 (MAX NP) *An optimization problem Π is in* MAX NP *if it can be expressed as the problem of finding a structure S which maximizes the objective function*

$$V(\mathcal{I}, S) = |\{\vec{x} \ : \ \exists\, \vec{y}\, \phi(\mathcal{I}, S, \vec{x}, \vec{y})\}|,$$

where $\mathcal{I}$ is the input and ϕ is a quantifier-free first order formula.

A typical example of a problem that belongs to the class MAX NP but not to MAX SNP, is the problem MAX SAT.

An immediate consequence of the syntactic definitions of these classes is a canonical approximation algorithm yielding a constant-factor approximation to every problem in both these classes [87]. Thus, both these classes are contained in the approximation class APX. However, it is not difficult to show (via syntactic considerations) that many problems in APX are not expressible as a problem in either of these classes, and hence that the containment is strict (see [74], for example).

Soon after the work of Papadimitriou and Yannakakis, a variety of other syntactic optimization classes were introduced and well-studied in the literature. We formally introduce some of these classes, such as RMAX(k) [85] and MIN F$^+\Pi_2(k)$ [73], in the next chapter. In the remainder of this

section, we focus on the class MAX SNP. We first describe the notion of approximation-preserving reduction used by Papadimitriou and Yannakakis; we begin with the concept of error in a solution.

Definition 13 (Error) *Given a solution S to an instance $\mathcal{I}$ of an* NPO *problem* Π*, we define its* error $\mathcal{E}(\mathcal{I}, S)$ *as*

$$\mathcal{E}(\mathcal{I}, S) = \max\left\{\frac{V(\mathcal{I}, S)}{\mathrm{OPT}(\mathcal{I})}, \frac{\mathrm{OPT}(\mathcal{I})}{V(\mathcal{I}, S)}\right\} - 1.$$

Notice that the above definition of error applies uniformly to the minimization and maximization problems at all levels of approximability.

Definition 14 (L-Reduction) *A problem* Π L-reduces *to another problem* Π' *(denoted* $\Pi \propto_L \Pi'$*) if there exist polynomial-time computable functions f and g, and two positive constants α and β, satisfying the following:*

- *f maps an instance $\mathcal{I}$ of* Π *to an instance $\mathcal{I}'$ of* Π' *such that* $\mathrm{OPT}(\mathcal{I}') \leq \alpha \mathrm{OPT}(\mathcal{I})$*,*
- *g maps solutions S' of $\mathcal{I}'$ to solutions S of $\mathcal{I}$ such that*

$$|\mathrm{OPT}(\mathcal{I}) - V(\mathcal{I}, S)| \leq \beta |\mathrm{OPT}(\mathcal{I}') - V(\mathcal{I}', S)|.$$

It is easy to verify the following two facts [87].

Fact 1 *L-reductions compose together, that is, if* $\Pi \propto_L \Pi'$ *and* $\Pi' \propto_L \Pi'''$*, then* $\Pi \propto_L \Pi''$*.*

Fact 2 *L-reductions preserve approximability. If* $\Pi \propto_L \Pi'$*, then given a solution S' to* Π' *with error $\mathcal{E}(\mathcal{I}', S')$, we can obtain a solution S to* Π *with error at most $\alpha\beta\mathcal{E}(\mathcal{I}, S)$.*

This linear relation between the two errors gives the L-reductions their name — *linear reductions*.

Akin to the notion of NP-hardness, Papadimitriou and Yannakakis defined the notion of MAX SNP-hardness using L-reductions[4].

[4]Later, we slightly modify the notion of MAX SNP-hardness by introducing somewhat more general reductions.

Definition 15 (MAX SNP-Hard) *An optimization problem* Π *is said to be* MAX SNP-hard *if for any* $\Pi' \in$ MAX SNP, $\Pi' \propto_L \Pi$.

Definition 16 (MAX SNP-Complete) *An optimization problem* Π *is* MAX SNP-complete *if* $\Pi \in$ MAX SNP *and* Π *is* MAX SNP-hard.

The approximation-preserving nature of L-reductions ensures that if any MAX SNP-hard problem has a PTAS, then so does every problem in the entire class MAX SNP. A wide variety of important optimization problems has been shown to be MAX SNP-hard. Some examples include MAX 3-SAT, MIN VERTEX COVER, TSP(1,2), MAX INDEPENDENT SET in bounded degree graphs (MIS-B), and so on. A positive or a negative approximability for any of these problems would imply the same for the rest of the class — a development very similar to the unification facilitated by the theory of NP-completeness.

1.1.6 Probabilistically Checkable Proofs (PCP)

As we saw earlier, there is a fundamental connection between verification of proofs and the definition of the complexity class NP. We essentially defined NP as the class of languages whose membership proofs are *efficiently verifiable*. The notion of efficient verification is based on polynomial-time verification in the classical definition where the verifier examines the complete proof and the verification process is assumed to be deterministic. In the mid-80's, however, a new notion of proof verification emerged. In this notion, the verifier is allowed an access to a random source and it verifies the proof by examining only some of its bits. Thus, in the new framework, a verifier need not read the complete proof. Of course, a consequence is that the verifier can no longer be certain of having made a correct decision. The efficiency of verification is characterized by the number of random bits needed and the query bits read by the verifier. This gave rise to a new family of complexity classes as defined below.

Definition 17 *(*PCP*) Given two functions* $r, q : \mathcal{Z}^+ \rightarrow \mathcal{Z}^+$*, a language* L *is said to be in* PCP(r, q) *if there exists a probabilistic polynomial-time oracle machine* M *such that for all* $x \in \Sigma^*$*, we have*

- $x \in L \Rightarrow \exists\, \pi$ *such that on all choices of random strings,* M *on oracle* π *accepts* x.

- $x \notin L \Rightarrow \forall\, \pi$, *$M$ on oracle π accepts with probability at most 1/2.*

Moreover, M uses $O(r(n))$ coin flips and queries the oracle π at most $O(q(n))$ times.

The above definition was formalized by Arora and Safra [8], and has its origins in the work of Babai [11] and Goldwasser, Micali and Rackoff ([46]). The classical definition of NP can be restated in terms of the above definition: NP $=$ PCP$(0, poly(n))$ where $poly(n)$ denotes the class of univariate polynomials. However, an alternate surprising characterization of NP was provided by Arora, Lund, Motwani, Sudan and Szegedy [7] (a detailed exposition of this result can be found in [2, 97]).

Theorem 2 ([7]) NP $=$ PCP$(\log n, 1)$.

This result says that if there exists a polynomial size membership proof, then there also exists another polynomial size membership proof which can be verified with high probability by just examining a constant number of its bits! As we are going to see in the next section, PCP based characterizations of NP has far-reaching ramifications for the hardness of approximation.

1.1.7 PCP and the Hardness of Approximation

The class MAX SNP, as defined by Papadimitriou and Yannakakis, gave the right *equivalence class* to address the question of whether a PTAS indeed existed for many constant-factor approximable optimization problems. This unification brought by this class is analogous to Karp's work showing equivalence between a whole variety of NP decision problems. But we still needed an analog of Cook's theorem for the class MAX SNP, that is, a result which would associate the question of PTAS for MAX SNP-hard problem with the collapse of two complexity classes into one. This critical final result came from PCP, a culmination of a long series of remarkable results.

In a pioneering paper, Feige *et al* [37] established a very surprising connection between efficiently checkable proofs and hardness of approximation. Specifically, they showed that if NP $\subseteq$ PCP(q, r) then there does not exist a constant-factor approximation algorithm for clique unless NP $\subseteq$ DTIME(2^{q+r}). Soon thereafter, a connection to MAX 3-SAT was also realized,

namely, MAX 3-SAT does not have a PTAS unless P = NP [7]. In one swoop, this final result excludes the possibility of PTAS for any MAX SNP-hard problem unless P = NP.

The sequence of events highlighting the deep connection between probabilistically checkable proofs and the hardness of approximation may appear surprising at first. But, in hindsight, one sees the intuition behind such a connection — the notion of probabilistic verification overcomes the bottleneck mentioned by Papadimitriou and Yannakakis ([87]), namely, the inherently unstable and non-robust nature of computation. Thus, just as the notion of exact verification in the classical definition of NP provided a means to study the intractability of computing exact solutions to the NP-hard optimization problems, the notion of probabilistic checking provided a means to study the intractability of computing approximate solutions to the NP-hard optimization problems.

1.2 Overview and Organization of Results

Broadly speaking, this dissertation can be viewed as being composed of three distinct parts.

1.2.1 Part I: A Structural Basis for Approximation Classes

As pointed out earlier, the classification of NPO problems into classes such as APX and poly-APX, does not yield insight into the structure of problems contained in these classes. We seek a characterization for the approximation classes akin to Fagin's syntactic characterization of NP which, in a similar manner, identifies the structural properties shared by all problems in each of these classes. A simple analogy from material sciences may help clearly illustrate the spirit of our quest. Materials can be classified as soft, hard, or brittle, based on their observed toughness. But alternately, one could associate the strength of the material with a certain underlying crystal structure, and thus obtain a structural characterization for all materials that are soft, hard, or brittle. Computationally-defined approximation classes are analogous to the classification of materials into soft, hard, or brittle, and what we seek is an understanding of the structure that underlies their observed approximability.

Our work in this part shows that indeed approximation classes have such syntactic characterizations. We show that with each of the computational classes, such as APX-PB, log-APX-PB and poly-APX-PB, we can associate a syntactic optimization class such that every problem in

the computational class is "expressible" via reductions as a problem in the associated syntactic class. For instance, we show that every problem in APX has an approximation-preserving reduction to a problem in MAX SNP. Our method introduces a general technique for creating approximation-preserving reductions which show that any well-approximable problem can be reduced in an approximation-preserving manner to a problem that is hard to approximate to corresponding factors. We demonstrate this technique by applying it to other syntactic classes. Thus we establish that the syntactic optimization classes provide a structural basis for the computationally-defined approximation classes.

This reconciliation of the computational and the syntactic views enhances our understanding of the approximation classes from both an algorithmic and a complexity-theoretic perspective. On the one hand, we can now identify natural complete problems for classes such as APX, while on the other hand, we can study algorithmic paradigms which uniformly apply to the set of core problems in the approximation classes.

Chapter 2 describes most of the aforementioned work. In the later half of this chapter, we study an algorithmic paradigm for MAX SNP. We use the syntactic nature of MAX SNP to define a general paradigm, *non-oblivious local search*, useful for developing simple yet efficient approximation algorithms. We show that such algorithms can find good approximations for all MAX SNP problems, yielding approximation ratios comparable to the best-known for a variety of specific MAX SNP-hard problems. Non-oblivious local search provably out-performs standard local search in both the degree of approximation achieved and the efficiency of the resulting algorithms.

While the techniques discussed in Chapter 2 lead to a syntactic characterization of several approximation classes, they do not extend to the class PTAS. In fact, we are not aware of any natural syntactic optimization classes which are subclasses of PTAS. Chapter 3 explores the possibility that a core of PTAS may be characterized through syntactic classes endowed with restrictions on the structure of the input instances.

We argue that while the core of APX is the purely syntactic class MAX SNP, in the case of PTAS we must identify the core in terms of syntactic prescriptions for the problem definition augmented with structural restrictions on the input instances. Specifically, we propose such a unified framework

based on syntactic classes restricted to instances exhibiting a planar structure. A variety of known results, and some new results, follow directly from this framework, thereby lending credence to our hypothesis that there exists some common structural underpinnings for problems in PTAS.

1.2.2 Part II: Approximability of the Constraint Satisfaction Problems

In this part of the dissertation, we initiate a systematic study of the maximization versions of constraint satisfaction problems. On the one hand, our investigation leads to some surprising classification theorems which allow us to uniformly identify hard problems in these classes. More precisely, we show that the syntactic description of these classes can be used for an *exact characterization* of the central elements that govern the approximability of a problem in the class. On the other hand, this study also serves to formally confirm some trends in approximability which have emerged over the past several years.

The starting point in our study is the remarkable paper of Schaefer [95] which considers a subclass of languages in NP and proves a "dichotomy theorem" for this class. The subclass considered was problems expressible as "constraint satisfaction problems" and the dichotomy theorem showed that every language in this class is either in P, or is NP-hard. This result is in sharp contrast to a result of Ladner [75], which shows that such a dichotomy does not hold for NP, unless P $=$ NP.

We consider two classes of maximization problems built on Schaefer's framework of constraint satisfaction problems: the constraints are boolean constraints as in Schaefer's case, but the objective in one class is to find a solution maximizing the weight of satisfied constraints, while in the other, we seek a feasible solution which maximizes the weight of variables set to true. Together, these classes capture a whole variety of natural optimization problems such as MAX CUT, s-t MIN CUT in directed graphs, and MAX CLIQUE. We determine the approximability of every problem in each of these classes.

For the first class of problems, studied in Chapter 4, we show that every problem is either solvable *exactly* in P or is APX-hard (hence not *approximable* to within arbitrary constant factors in polynomial-time, unless P $=$ NP). The class of problems captured here is a subclass of MAX SNP. We feel that this class forms a combinatorial core of MAX SNP and our results for this class bring

a new insight towards understanding the class MAX SNP.

The problems in the second class, on the other hand, are shown to belong to one of five classes: exactly solvable in polynomial-time, approximable to within constant factors in polynomial-time, approximable to within polynomial factors in polynomial-time, not approximable but decidable, and undecidable (unless P=NP). Thus, this class has a representative problem at each level of approximability threshold observed for maximization problems. Our results here give some evidence towards the truth of approximability patterns that have emerged from our study of maximization problems. This latter class of problems is studied in Chapter 5.

Our work here builds on the impressive collection of non-approximability results derived over the recent years (see the surveys [6, 12, 15, 59], for example). However, much of the past work has been *enumerative* in its approach, considering specific problems and then proving hardness results for such problems. Our approach adds to this body of work by extracting *exhaustive* results from these — that is, we consider large (infinite) collections of problems and then characterize precisely the asymptotic approximability of each one of these problems.

1.2.3 Part III: Hardness of Approximating Graph-Coloring

The last five years have witnessed an astounding progress in our understanding of the approximability of NPO problems. An extensive collection of open problems has been either solved completely or a significant progress has been made towards the final answer. The $(n^{1-\epsilon})$-hardness results for MAX CLIQUE [52] and COLORING [38], the $\log n$-threshold for approximating MIN SET COVER [36], the PTAS for the Euclidean TSP problem [3], the significantly improved approximation ratios for MAX CUT [45], MAX 2-SAT [45], and GRAPH k-COLORING [65], are only some among this impressive collection of results. Yet there are important questions concerning the approximability of fundamental optimization problems that remain unanswered. Probably the most striking of these open questions is the approximability of the GRAPH 3-COLORING. Since Karp's original NP-hardness result over two decades ago, nothing more has been established on the approximability of this problem. The best known approximation algorithms has a polynomial approximation ratio [65]. In Chapter 6, we study this problem and make a modest progress (but the only progress thus far), in improving the lower bound for this problem. Specifically, we show

that coloring a 3-colorable graph with 4 colors is NP-hard. This modest progress already requires somewhat involved ideas. An immediate corollary of this result is that a k-colorable graph cannot be colored with less than $k + 2\lfloor k/3 \rfloor$ colors unless P $=$ NP. We also present a simplified proof of the $\Omega(n^{\epsilon})$ lower bound due to Lund and Yannakakis [82] on the approximability of the general graph coloring problem.

Chapter 2

A Structure Theorem for Approximation Classes

2.1 Introduction *

We saw earlier that a variety of classes of NPO problems have been defined based on their polynomial-time approximability. Some examples of these classes are APX (the class of constant-factor approximable problems), PTAS (the class of problems with polynomial-time approximation schemes), and FPTAS (the class of problems with fully-polynomial-time approximation schemes). The advantage of working with classes defined using approximability as the criterion is that it allows us to work with problems whose approximability is well-understood. Crescenzi and Panconesi [28] have recently also been able to exhibit complete problems for such classes, particularly APX. Unfortunately such complete problems seem to be rare and artificial, and do not seem to provide insight into the more natural problems in the class. Research in this direction has to find approximation-preserving reductions from the known complete but artificial problems in such classes to the natural problems therein, with a view to understanding the approximability of the latter.

We also saw that over the last decade, a second family of classes of NPO problems has emerged.These are the classes defined via syntactic considerations, based on a syntactic characterization of NP due to Fagin [34]. Research in this direction, initiated by Papadimitriou and Yannakakis [87], and followed by Panconesi and Ranjan [85] and Kolaitis and Thakur [73], has led to the identification of classes such as MAX SNP, RMAX(2), and MIN $F^{+}\Pi_2(1)$. The syntactic prescription in the definition of these classes has proved very useful in the establishment of complete problems. Moreover, the recent results of Arora, Lund, Motwani, Sudan, and Szegedy [7] have established the hardness of approximating complete problems for MAX SNP to within (specific) constant factors unless P = NP. It is natural to wonder why the hardest problems in this syntactic sub-class of APX should bear any relation to all of NP.

Though the computational view allows us to precisely classify the problems based on their approximability, it does not yield structural insights into natural questions such as: Why certain problems are easier to approximate than some others? What is intrinsic to optimization problems with a similar approximability behaviour? What is the canonical structure of the hardest representative problems of a given approximation class? and, so on. Furthermore, intuitively speaking, this

* This chapter is based on joint work with Rajeev Motwani, Madhu Sudan and Umesh Vazirani [67].

view is too abstract to facilitate the identification of, and reductions to establish, natural complete problems for a class. Showing existence of natural complete problems is a critical step in establishing the importance of a complexity class. The importance of the class NP, for instance, is strongly linked to its abundance in naturally arising important problems. The syntactic view, on the other hand, is essentially a structural view. The syntactic prescription gives a natural way of identifying canonical hard problems in the class and performing approximation-preserving reductions to establish complete problems.

Attempts at trying to find a class with both the above mentioned properties, i.e., natural complete problems and capturing all problems of a specified approximability, have not been very successful. Typically the focus has been to relax the syntactic criteria to allow for a wider class of problems to be included in the class. However in all such cases it seems inevitable that these classes cannot be expressive enough to encompass all problems with a given approximability. This is because each of these syntactically defined approximation classes is strictly contained in the class NPO; the strict containment can be shown by syntactic considerations alone. As a result if we could show that any of these classes contains all of P, then we would have separated P from NP. We would expect that every class of this nature would be missing some problems from P, and this has indeed been the case with all current definitions.

We explore a different direction by studying the structure of the syntactically defined classes when we look at their closure under approximation-preserving reductions. The idea of looking at the closure of a class is implicit in the work of Papadimitriou and Yannakakis [87] who state that: *minimization problems will be "placed" in the classes through L-reductions to maximization problems*. The advantage of looking at the closure of a set is that it maintains the complete problems of the set, while managing to include all of P into the closure (for problems in P, the reduction is to simply use a polynomial time algorithm to compute an exact solution). It now becomes interesting, for example, to compare the closure of MAX SNP (denoted $\overline{\text{MAX SNP}}$) with APX. A positive resolution, i.e., $\overline{\text{MAX SNP}} = \text{APX}$, would immediately imply the non-existence of a PTAS for MAX SNP-hard problems, since it is known that PTAS is a strict subset of APX, if P $\neq$ NP. On the other hand, an unconditional negative result would be difficult to obtain, since it would imply P $\neq$ NP.

Here we resolve this question in the affirmative. The exact nature of the result obtained depends upon the precise notion of an approximation preserving reduction used to define the closure of the class MAX SNP. The strictest notion of such reductions available in the literature are the L-reductions due to Papadimitriou and Yannakakis [87]. We work with a slight extension of the reduction, which we call E-reductions. Using such reductions to define the class $\overline{\text{MAX SNP}}$ we show that this equals APX-PB, the class of all polynomially bounded NP optimization problems which are approximable to within constant factors. An interesting side-effect of our results is the positive answer to the question of Papadimitriou and Yannakakis [87] about whether MAX NP has any complete problems. We next extend the technique used in showing this result to establish a general *structure theorem* which shows that any "well" approximable problem can be reduced in an approximation-preserving manner to a problem which is hard to approximate to corresponding factors. The structure theorem provides a powerful tool for establishing that syntactic optimization classes provide a structural basis for the computationally defined approximation classes. We demonstrate this technique by applying it to other syntactic classes.

It is worthwhile to note that by using slightly looser definitions of approximation preserving reductions (and in particular the PTAS-reductions of Crescenzi and Trevisan [29]), our results can be extended to show that all of APX equals $\overline{\text{MAX SNP}}$.

The syntactic view seems useful not only in obtaining structural complexity results but also in developing paradigms for designing efficient approximation algorithms. This was demonstrated first by Papadimitriou and Yannakakis [87] who show approximation algorithms for every problem in MAX SNP. We further exploit the syntactic nature of MAX SNP to develop another paradigm for designing good approximation algorithms for problems in that class and thereby provide an alternate computational view of it. We refer to this paradigm as *non-oblivious local search*, and it is a modification of the standard local search technique [99]. We show that every MAX SNP problem can be approximated to within constant factors by such algorithms. It turns out that the performance of non-oblivious local search is comparable to that of the best-known approximation algorithms for several interesting and representative problems in MAX SNP. An intriguing possibility is that this is not a coincidence, but rather a hint at the universality of the paradigm or some variant thereof.

Our results are related to some extent to those of Ausiello and Protasi [10]. They define a class

GLO (for Guaranteed Local Optima) of NPO problems which have the property that for all locally optimum solutions, the ratio between the value of the global and the local optimum is bounded by a constant. It follows that GLO is a subset of APX, and it was shown that it is in fact a strict subset. We show that a MAX SNP problem is not contained in GLO, thereby establishing that MAX SNP is not contained in GLO. This contrasts with our notion of non-oblivious local search which is guaranteed to provide constant factor approximations for all problems in MAX SNP. In fact, our results indicate that non-oblivious local search is significantly more powerful than standard local search in that it delivers strictly better constant ratios, and also will provide constant factor approximations to problems not in GLO. Independently of our work, Alimonti [1] has used a similar local search technique for the approximation of a specific problem not contained in GLO or MAX SNP.

2.2 Organization of Results

In Section 2.3, we introduce E-reductions and the notion of closure of a class. In Section 2.4, we show that $\overline{\text{MAX SNP}} = \text{APX-PB}$. A generic theorem which allows to equate the closure of syntactic classes to appropriate computational classes is outlined in Section 2.5; we also develop an approximation hierarchy based on this result.

The notion of non-oblivious local search and NON-OBLIVIOUS GLO is developed in Section 2.6. In Section 2.7, we illustrate the power of non-obliviousness by first showing that oblivious local search can achieve at most the performance ratio $3/2$ for MAX 2-SAT, even if it is allowed to search *exponentially* large neighborhoods; in contrast, a very simple non-oblivious local search algorithm achieves a performance ratio of $4/3$. We then establish that this paradigm yields a $2^k/(2^k-1)$ approximation to MAX k-SAT. In Section 2.8, we provide an alternate characterization of MAX SNP via a class of problems called MAX k-CSP. It is shown that a simple non-oblivious algorithm achieves the best-known approximation for this problem, thereby providing a *uniform* approximation for all of MAX SNP. In Section 2.9, we further illustrate the power of this class of algorithms by showing that it can achieve the best-known ratio for a specific MAX SNP problem and for VERTEX COVER (which is not contained in GLO). This implies that MAX SNP is not

contained in GLO, and that GLO is strict subset of NON-OBLIVIOUS GLO. In Section 2.10, we apply it to approximating the traveling salesman problem. Finally, in Section 2.11, we apply this technique to improving a long-standing approximation bound for maximum independent sets in bounded-degree graphs.

2.3 E-Reductions

Given an NPO problem Π and an instance $\mathcal{I}$ of Π, we use $|\mathcal{I}|$ to denote the length of $\mathcal{I}$ and $\text{OPT}(\mathcal{I})$ to denote the optimum value for this instance. Also, recall that for any solution S to $\mathcal{I}$, the value of the solution is denoted by $V(\mathcal{I}, S)$ and is assumed to be a polynomial time computable function which takes positive integer values (see Chapter 1 for formal definitions).

We now describe precisely the approximation preserving reduction that we will use. This reduction, which we call the E-reduction, is essentially the same as the L-reduction of Papadimitriou and Yannakakis [87] and differs from it in only one relatively minor aspect.

Definition 18 (E-reduction) *A problem* Π E-reduces *to a problem* Π' *(denoted* $\Pi \propto_E \Pi'$*) if there exist polynomial time computable functions* f*,* g *and a constant* β *such that*

- f *maps an instance* $\mathcal{I}$ *of* Π *to an instance* $\mathcal{I}'$ *of* Π' *such that* $\text{OPT}(\mathcal{I})$ *and* $\text{OPT}(\mathcal{I}')$ *are related by a polynomial factor i.e. there exists a polynomial* $p(n)$ *such that* $\text{OPT}(\mathcal{I}') \leq p(|\mathcal{I}|)\text{OPT}(\mathcal{I})$.
- g *maps solutions* S' *of* $\mathcal{I}'$ *to solutions* S *of* $\mathcal{I}$ *such that*

$$\mathcal{E}(\mathcal{I}, S) \leq \beta \mathcal{E}(\mathcal{I}', S').$$

Remark 1 *Among the many approximation preserving reductions in the literature, the* L*-reduction appears to be the strictest. The* E*-reduction appears to be slightly weaker (in that it allows polynomial scaling of the problems), but is stricter than any of the other known reductions. Since all the reductions given in chapter are* E*-reductions, they would also qualify as approximation-preserving reductions under most other definitions and in particular, they fit the definitions of* F*-reductions and* P*-reductions of Crescenzi and Panconesi* [28].

Remark 2 *Having* $\Pi \propto_E \Pi'$ *implies that* Π *is as well approximable as* Π'*; in fact, an* E*-reduction is an* FPTAS*-preserving reduction. An important benefit is that this reduction can be applied uniformly at all levels of approximability. This is not the case with the other existing definitions of* FPTAS*-preserving reduction in the literature. For example, the* FPTAS*-preserving reduction (*F*-reduction) of Crescenzi and Panconesi* [28] *is much more unrestricted in scope and does not share this important property of the* E*-reduction. Note that Crescenzi and Panconesi* [28] *showed that there exists a problem* $\Pi' \in$ PTAS *such that for any problem* $\Pi \in$ APX, $\Pi \propto_F \Pi'$*. Thus, there is the undesirable situation that a problem* Π *with no* PTAS *has a* FPTAS*-preserving reduction to a problem* Π' *with a* PTAS*.*

Remark 3 *The* L*-reduction of Papadimitriou and Yannakakis* [87] *enforces the condition that the optima of an instance* $\mathcal{I}$ *of* Π *be linearly related to the optima of the instance* $\mathcal{I}'$ *of* Π' *to which it is mapped. This appears to be an unnatural restriction considering that the reduction itself is allowed to be an arbitrary polynomial time computation. This is the only real difference between their* L*-reduction and our* E*-reduction, and an* E*-reduction in which the linearity relation of the optimas is satisfied is an* L*-reduction. Intuitively, however, in the study of approximability the desirable attribute is simply that the errors in the corresponding solutions are closely (linearly) related. The somewhat artificial requirement of a linear relation between the optimum values precludes reductions between problems which are related to each other by some* scaling factor*. For instance, it seems desirable that two problems whose objective functions are simply related by any fixed polynomial factor should be inter-reducible under any reasonable definition of an approximation-preserving reduction. Our relaxation of the* L*-reduction constraint is motivated precisely by this consideration.*

Let $\mathcal{C}$ be any class of NPO problems. Using the notion of an E-reduction, we define hardness and completeness of problems with respect $\mathcal{C}$, as well its closure and polynomially-bounded sub-class.

Definition 19 (Hard and Complete Problems) *A problem* Π' *is said to be* $\mathcal{C}$-hard *if for all problems* $\Pi \in \mathcal{C}$*, we have* $\Pi \propto_E \Pi'$*. A* $\mathcal{C}$*-hard problem* Π *is said to be* $\mathcal{C}$-complete *if in addition* $\Pi \in \mathcal{C}$*.*

Definition 20 (Closure) *The* closure *of $\mathcal{C}$, denoted by $\overline{\mathcal{C}}$, is the set of all* NPO *problems* Π *such that* $\Pi \propto_E \Pi'$ *for some* $\Pi' \in \mathcal{C}$.

Remark 4 *The closure operation maintains the set of complete problems for a class*.

2.4 MAX SNP Closure and APX-PB

In this section, we will establish the following theorem and examine its implications. The proof is based on the results of Arora et al [7] on efficient proof verifications.

Theorem 3 $\overline{\text{MAX SNP}} = \text{APX-PB}$.

Remark 5 *The seeming weakness that* $\overline{\text{MAX SNP}}$ *only captures polynomially bounded* APX *problems can be removed by using looser forms of approximation-preserving reduction in defining the closure*. *In particular, Crescenzi and Trevisan [29] define the notion of a* PTAS*-preserving reduction under which* $\text{APX} = \overline{\text{APX-PB}}$. *Using their result in conjunction with the above theorem, it is easily seen that* $\overline{\text{MAX SNP}} = \text{APX}$. *This weaker reduction is necessary to allow for reductions from fine-grained optimization problems to coarser (polynomially-bounded) optimization problems (cf*. [29]*)*.

The following is a surprising consequence of Theorem 3.

Theorem 4 $\overline{\text{MAX NP}} = \overline{\text{MAX SNP}}$.

Papadimitriou and Yannakakis [87] (implicitly) introduced both these closure classes but did not conjecture them to be the same. It would be interesting to see if this equality can be shown independent of the result of Arora et al [7]. We also obtain the following resolution to the problem posed by Papadimitriou and Yannakakis [87] of finding complete problems for MAX NP.

Theorem 5 MAX SAT *is complete for* MAX NP.

The following sub-sections establish that $\overline{\text{MAX SNP}} \supseteq \text{APX-PB}$. The idea is to first E-reduce any minimization problem in APX-PB to a maximization problem in therein, and then

E-reduce any maximization problem in APX-PB to a specific complete problem for MAX SNP, viz., MAX 3-SAT. Since an E-reduction forces the optimas of the two problems involved to be related by polynomial factors, it is easy to see that $\overline{\text{MAX SNP}} \subseteq$ APX-PB. Combining these two facts, we obtain Theorem 3.

2.4.1 Reducing Minimization Problems to Maximization Problems

Observe that the fact that Π belongs to APX implies the existence of an approximation algorithm A and a constant c such that

$$\frac{\text{OPT}(\mathcal{I})}{c} \leq V(\mathcal{I}, A(\mathcal{I})) \leq c \times \text{OPT}(\mathcal{I}).$$

Henceforth, we will use $a(\mathcal{I})$ to denote $V(\mathcal{I}, A(\mathcal{I}))$. We first reduce any minimization problem $\Pi \in$ APX-PB to a maximization problem $\Pi' \in$ APX-PB, where the latter is obtained by merely modifying the objective function for Π, as follows. Let Π' have the objective function

$$V'(\mathcal{I}, S) = \max\{1, (c+1)a(\mathcal{I}) - cV(\mathcal{I}, S)\},$$

for all instances $\mathcal{I}$ and solutions S for Π. Clearly, $V'(\mathcal{I}, S)$ takes only positive values. To ensure that $V'(\mathcal{I}, S)$ is integer-valued, we can assume without loss of generality, that c is an integer (a real-valued performance ratio can always be rounded up to the next integer). It can be verified that the optimum value for any instance $\mathcal{I}$ of Π' always lies between $a(\mathcal{I})$ and $(c+1)a(\mathcal{I})$. Thus A is a $(c+1)$-approximation algorithm for Π'.

Now given a solution S' for instance $\mathcal{I}$ of Π' such that it has error δ, we want to construct a solution S for instance $\mathcal{I}$ of Π such that the error is at most $\beta\delta$ for some β. We will show this for $\beta = (c+1)$.

First consider the case when $V'(\mathcal{I}, S') = 1$ i.e. $\delta = a(\mathcal{I}) - 1$. In this case, we simply output the solution $S = A(\mathcal{I})$. If $a(\mathcal{I}) = 1$ then we are trivially done else we observe that

$$\mathcal{E}(I, S) \leq (c-1) \leq (c+1)(a(\mathcal{I}) - 1) \leq \beta\mathcal{E}'(I, S').$$

On the other hand, if $V'(\mathcal{I}, S') > 1$, we may proceed as follows. If S' is a δ-error solution to the optimum of Π', i.e.,

$$V'(\mathcal{I}, S) \geq \frac{\text{OPT}'(\mathcal{I})}{1+\delta} \geq (1-\delta)OPT'(\mathcal{I}),$$

where $\text{OPT}'(\mathcal{I})$ is the optimal value of V' for $\mathcal{I}$, we can conclude that

$$\begin{aligned} V(\mathcal{I}, S) &= \frac{(c+1)a(\mathcal{I}) - V'(\mathcal{I}, S)}{c} \\ &\leq \frac{(c+1)a(\mathcal{I}) - \text{OPT}'(\mathcal{I}) + \delta \times \text{OPT}'(\mathcal{I})}{c} \\ &\leq \frac{c \times \text{OPT}(\mathcal{I}) + \delta \times \text{OPT}'(\mathcal{I})}{c} \\ &\leq \text{OPT}(\mathcal{I}) + (c+1)\delta\text{OPT}(\mathcal{I}). \end{aligned}$$

Thus a solution s to Π' with error δ is a solution to Π with error at most $(c+1)\delta$, implying an E-reduction with $\beta = c+1$.

2.4.2 NP Languages and MAX 3-SAT

The following theorem, adapted from a result of Arora, Lund, Motwani, Sudan, and Szegedy [7], is critical to our E-reduction of maximization problems to MAX 3-SAT.

Theorem 6 ([7]) *Given a language $L \in$ NP and an instance $x \in \Sigma^n$, one can compute in polynomial time an instance $\mathcal{F}_x$ of* MAX 3-SAT*, with the following properties.*

1. *The formula $\mathcal{F}_x$ has m clauses, where m depends only on n.*
2. *There exists a constant $\epsilon > 0$, independent of the input x, such that $(1-\epsilon)m$ clauses of $\mathcal{F}_x$ are satisfied by some truth assignment.*
3. *If $x \in L$, then $\mathcal{F}_x$ is (completely) satisfiable.*
4. *If $x \notin L$, then no truth assignment satisfies more than $(1-\epsilon)m$ clauses of $\mathcal{F}_x$.*

5. *Given a truth assignment which satisfies more than* $(1-\epsilon)m$ *clauses of* $\mathcal{F}_x$*, a truth assignment which satisfies* $\mathcal{F}_x$ *completely (or, alternatively, a witness showing* $x \in L$*) can be constructed in polynomial time*.

Some of the properties above may not be immediately obvious from the construction given by Arora, Lund, Motwani, Sudan, and Szegedy [7]. It is easy to verify that they provide a reduction with properties (1), (3) and (4). Property (5) is obtained from the fact that all assignments which satisfy most clauses are actually close (in terms of Hamming distance) to valid codewords from a linear code, and the uniquely error-corrected codeword obtained from this "corrupted code-word" will satisfy all the clauses of $\mathcal{F}_x$.

Property (2) requires a bit more care and we provide a brief sketch of how it may be ensured. The idea is to revert back to the PCP model and redefine the proof verification game. Suppose that the original game had the properties that for $x \in L$ there exists a proof such that the verifier accepts with probability 1, and otherwise, for $x \notin L$, the verifier accepts with probability at most $1/2$. We now augment this game by adding to the proof a 0th bit which the prover uses as follows: if the bit is set to 1, then the prover "chooses" to play the old game, else he is effectively "giving up" on the game. The verifier in turn first looks at the 0th bit of the proof. If this is set, then she performs the usual verification, else she tosses an unbiased coin and accepts if and only if it turns up heads. It is clear that for $x \in L$ there exists a proof on which the verifier always accepts. Also, for $x \notin L$ no proof can cause the verifier to accept with probability greater than $1/2$. Finally, by setting the 0th bit to 0, the prover can create a proof which the verifier accepts with probability exactly $1/2$. This proof system can now be transformed into a 3-CNF formula of the desired form.

2.4.3 Reducing Maximization Problems to MAX 3-SAT

We have already established that, without loss of generality, we only need to worry about maximization problems $\Pi \in$ APX-PB. Consider such a problem Π, and let A be a polynomial-time algorithm which delivers a c-approximation for Π, where c is some constant. Given any instance $\mathcal{I}$ of Π, let $p = ca(\mathcal{I})$ be the bound on the optimum value for $\mathcal{I}$ obtained by running A on input $\mathcal{I}$. Note that this may be a stronger bound than the a priori polynomial bound on the optimum value

for any instance of length $|\mathcal{I}|$. An important consequence is that $p \leq c\text{OPT}(\mathcal{I})$.

We generate a sequence of NP decision problems $L_i = \{\mathcal{I} | \text{OPT}(\mathcal{I}) \geq i\}$ for $1 \leq i \leq p$. Given an instance $\mathcal{I}$, we create p formulas $\mathcal{F}_i$, for $1 \leq i \leq p$, using the reduction from Theorem 6, where ith formula is obtained from the NP language L_i.

Consider now the formula $\mathcal{F} = \bigwedge_{i=1}^{p} \mathcal{F}_i$ that has the following features.

- The number of satisfiable clauses of $\mathcal{F}$ is exactly

$$MAX = (1-\epsilon)mp + \epsilon m\text{OPT}(\mathcal{I}),$$

  where ϵ and m are as guaranteed by Theorem 6.

- Given an assignment which satisfies $(1-\epsilon)mp + \epsilon mj$ clauses of $\mathcal{F}$, we can construct in polynomial time a solution to $\mathcal{I}$ of value at least j. To see this, observe the following: any assignment which so many clauses must satisfy more than $(1-\epsilon)m$ clauses in at least j of the formulas $\mathcal{F}_i$. Let i be the largest index for which this happens; clearly, $i \geq j$. Furthermore, by property (5) of Theorem 6, we can now construct a truth assignment which satisfies $\mathcal{F}_i$ completely. This truth assignment can be used to obtain a solution S such that $V(\mathcal{I}, S) \geq i \geq j$.

In order to complete the proof it remains to be shown that given any truth assignment with error δ, i.e., which satisfies $MAX/(1+\delta)$ clauses of $\mathcal{F}$, we can find a solution S for $\mathcal{I}$ with error $\mathcal{E}(\mathcal{I}, S) \leq \beta\delta$ for some constant β. We show that this is possible for $\beta = (c^2 + c\epsilon)/\epsilon$. The main idea behind finding such a solution is to use the second property above to find a "good" solution to $\mathcal{I}$ using a "good" truth assignment for $\mathcal{F}$.

Suppose we are given a solution which satisfies $MAX/(1+\delta)$ clauses. Since $MAX/(1+\delta) \geq (1-\delta)\,MAX$ and $MAX = (1-\epsilon)mp + \epsilon m\text{OPT}(\mathcal{I})$, we can use the second feature from above to construct a solution S_1 such that

$$\begin{aligned} V(\mathcal{I}, S_1) &\geq \frac{(1-\delta)\,MAX - (1-\epsilon)mp}{\epsilon m} \\ &\geq (1-\delta)\text{OPT}(\mathcal{I}) - \frac{\delta}{\epsilon}p \end{aligned}$$

$$\geq \left(1 - \delta\left(1 + \frac{c}{\epsilon}\right)\right) \mathrm{OPT}(\mathcal{I}).$$

Suppose $\delta \leq (c-1)\epsilon/(c(c+\epsilon))$. Let $\delta^* = \delta(1+c/\epsilon)$, and $\gamma = \delta^*/(1-\delta^*)$. Then it is readily seen that

$$V(\mathcal{I}, S_1) \geq \frac{\mathrm{OPT}(\mathcal{I})}{1+\gamma}$$

and than

$$0 \leq \gamma \leq \left(\frac{c^2 + c\epsilon}{\epsilon}\right)\delta.$$

On the other hand, if $\delta > (c-1)\epsilon/(c(c+\epsilon))$, then the error in a solution S_2 obtained by running the c-approximation algorithm for Π is given by

$$c - 1 \leq \left(\frac{c^2 + c\epsilon}{\epsilon}\right)\delta.$$

Therefore, choosing $\beta = (c^2 + c\epsilon)/\epsilon$, we immediately obtain that the solution with larger value, among S_1 and S_2, has error at most $\beta\delta$. Thus, this reduction is indeed an E-reduction.

2.5 Generic Reductions and an Approximation Hierarchy

In this section we describe a generic technique for turning a hardness result into an approximation preserving reduction.

We start by listing the kind of constraints imposed on the hardness reduction, the approximation class and the optimization problem. We will observe at the end that these restrictions are obeyed by all known hardness results and the corresponding approximation classes.

Definition 21 (Additive Problems) *An* NPO *problem* Π *is said to be* additive *if there exists an operator* $+$ *and a polynomial time computable function* f *such that* $+$ *maps a pair of instances* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *to an instance* $\mathcal{I}_1 + \mathcal{I}_2$ *such that* $\mathrm{OPT}(\mathcal{I}_1 + \mathcal{I}_2) = \mathrm{OPT}(\mathcal{I}_1) + \mathrm{OPT}(\mathcal{I}_2)$*, and* f *maps a solution* s *to* $\mathcal{I}_1 + \mathcal{I}_2$ *to a pair of solutions* s_1 *and* s_2 *to* $\mathcal{I}_1$ *and* $\mathcal{I}_2$*, respectively, such that* $V(\mathcal{I}_1 + \mathcal{I}_2, s) = V(\mathcal{I}_1, s_1) + V(\mathcal{I}_2, s_2)$.

Definition 22 (Downward Closed Family) *A family of functions* $F = \{f : \mathcal{Z}^+ \to \mathcal{Z}^+\}$ *is said*

to be downward closed *if for all $g \in F$ and for all constants c (and in particular for all integers $c > 1$), $g'(n) \in O(g(n^c))$ implies that $g' \in F$. A function g is said to be* hard *for the family F if for all $g' \in F$, there exists a constant c such that $g'(n) \in O(g(n^c))$; the function g is said to be* complete *for F if g is hard for F and $g \in F$.*

Definition 23 (F-APX) *For a downward closed family F, the class F-*APX *consists of all polynomially bounded optimization problems approximable to within a ratio of $g(|\mathcal{I}|)$ for some function $g \in F$.*

Definition 24 (Canonical Hardness) *An* NP *maximization problem* Π *is said to be* canonically hard *for the class F-*APX *if there exists a transformation T mapping instances of* 3-SAT *to instances of* Π*, constants n_0 and c, and a gap function G which is hard for the family F, such that given an instance x of* 3-SAT *on $n \geq n_0$ variables and $N \geq n^c$, $\mathcal{I} = T(x, N)$ is an instance of* Π *with the following properties.*

- *If $x \in$* 3-SAT*, then* OPT$(\mathcal{I}) = N$.
- *If $x \notin$* 3-SAT*, then* OPT$(\mathcal{I}) = N/G(N)$.
- *Given a solution S to $\mathcal{I}$ with $V(\mathcal{I}, S) > N/G(N)$, a truth assignment satisfying x can be found in polynomial time.*

In the above definition, the transformation T from 3-SAT to Π is somewhat special in that one can specify the size/optimum of the reduced problem and T can produce a mapped instance of the desired size. This additional property is easily obtained for additive problems, by using sufficient number of additions till the optimum of the reduced problem is close to the target optimum, and then adding a problem of known optimum value to the reduced problem.

Canonical hardness for NP *minimization problems* is analogously defined: OPT$(\mathcal{I}) = N$ when the formula is satisfiable and OPT$(\mathcal{I}) = NG(N)$, otherwise. Given any solution with value less than $NG(N)$, one can construct a satisfying assignment in polynomial time.

2.5.1 The Reduction

Theorem 7 *If F is a downward closed family of functions, and an additive* NPO *problem* Ω *is canonically hard for the class* F-APX-PB*, then all problems in* F-APX-PB E*-reduce to* Ω.

Proof: Let Π be a polynomially bounded optimization problem in F-APX, approximable to within $c(.)$ by an algorithm A, and let Ω be a problem shown to be hard to within a factor $G(.)$ where G is hard for F. Let V and V' denote the objective functions of Π and Ω, respectively. We start with the special case where both Π and Ω are maximization problems. We describe the functions f, g and the constant β as required for an E-reduction.

Let $\mathcal{I} \in \Pi$ be an instance of size n; pick N so that $c(n)$ is $O(G(N))$. To describe our reduction, we need to specify the functions f and g. The function f is defined as follows. Let $m = V(\mathcal{I}, A(\mathcal{I}))$. For each $i \in \{1, \ldots, mc(n)\}$, let L_i denote the NP-language $\{\mathcal{I} | \text{OPT}(\mathcal{I}) \geq i\}$. Now for each i, we create an instance $\phi_i \in \Omega$ of size N such that if $\mathcal{I} \in L_i$ then OPT(ϕ_i) is N, and it is $N/G(N)$ otherwise. We define $f(\mathcal{I}) = \phi = \sum_i \phi_i$.

We now construct the function g. Given an instance $\mathcal{I} \in \Pi$ and a solution s' to $f(\mathcal{I})$, we compute a solution s to $\mathcal{I}$ in the following manner. We first use A to find a solution s_1. We also compute a second solution s_2 to $\mathcal{I}$ as follows. Let j be the largest index such that the solution s' projects down to a solution s'_j to the instance ϕ_j such that $V'(\phi_j, s'_j) > N/G(N)$. This in turn implies we can find a solution s_2 to witness $V(\mathcal{I}, s_2) \geq j$. Our solution s is the one among s_1 and s_2 that yields the larger objective function value.

We now show that the reduction is an E-reduction with $\beta = 1 + c(n)/(G(N) - 1)$.

Let $\alpha = \text{OPT}(I)/m$. Observe that

$$\text{OPT}(I') = Nm\left(\alpha + \frac{c(n)}{G(N)} - \frac{\alpha}{G(N)}\right).$$

Consider the following two cases:

Case 1 [$j \leq m$]: In this case, $V(\mathcal{I}, s) = m$. Since s is a solution to $\mathcal{I}$ of error at most $(\alpha - 1)$, it suffices to argue that the error of s' as a solution to ϕ is at least $(\alpha - 1)/\beta$. We start with the following upper bound on $V(\phi, s')$.

$$V(\phi, s') \leq Nm\left(1 + \frac{c(n)}{G(N)} - \frac{1}{G(N)}\right).$$

Thus the approximation factor achieved by s' is given by

$$\begin{aligned}
\mathcal{E}(\phi, s') &\geq \left(\frac{Nm\left(\alpha + \frac{c(n)}{G(N)} - \frac{\alpha}{G(N)}\right)}{Nm\left(1 + \frac{c(n)}{G(N)} - \frac{1}{G(N)}\right)}\right) - 1 \\
&\geq (\alpha - 1)\left(\frac{G(N) - 1}{G(N) + c(n) - 1}\right) \\
&= \frac{\alpha - 1}{\beta}.
\end{aligned}$$

So in this case s_1 (and hence s) is a solution to $\mathcal{I}$ with an error of at most $\beta\epsilon$, if s' is a solution to ϕ with an error of ϵ.

Case 2 [$j \geq m$]: Let $j = \gamma m$. Note that $\gamma > 1$ and that the error of s as a solution to $\mathcal{I}$ is $(\alpha - \gamma)/\gamma$. We bound the value of the solution s' to ϕ as

$$V(\phi, s') \leq Nm\left(\gamma + \frac{c(n)}{G(N)} - \frac{\gamma}{G(N)}\right),$$

and its error as

$$\begin{aligned}
\mathcal{E}(\phi, s') &= \left(\frac{\alpha + \frac{c(n)}{G(N)} - \frac{\alpha}{G(N)}}{\gamma + \frac{c(n)}{G(N)} - \frac{\gamma}{G(N)}}\right) - 1 \\
&\geq \left(\frac{\alpha - \gamma}{\gamma}\right)\left(\frac{1}{1 + \frac{c(n)}{\gamma(G(N)-1)}}\right) \\
&\geq \left(\frac{\alpha - \gamma}{\gamma}\right)\frac{1}{\beta}.
\end{aligned}$$

(The final inequality follows from the fact that $1 + \frac{c(n)}{\gamma(G(N)-1)} \leq 1 + \frac{c(n)}{(G(N)-1)} = \beta$.)

Thus in this case also we find that s (by virtue of s_2) is a solution to $\mathcal{I}$ of error at most $\beta\epsilon$ if s' is a solution to ϕ of of error ϵ.

We now consider the more general cases where Π and Ω are not both maximization problems.

For the case where both are minimization problems, the above transformation works with one minor change. When creating ϕ_i, the NP language consists of instances $(\mathcal{I}, i)$ such that there exists s with $V(\mathcal{I}, s) \leq i$.

For the case where Π is a minimization problem and Ω is a maximization problem, we first E-reduce Π to a maximization problem Π' and then proceed as before. The reduction proceeds as follows. Since Π is a polynomially bounded optimization problem, we can compute an upper bound on the value of any solution s to an instance $\mathcal{I}$. Let m be such a bound for an instance $\mathcal{I}$. The objective function of Π' on the instance $\mathcal{I}$ is defined as $V'(\mathcal{I}, s) = \lfloor 2m^2/V(\mathcal{I}, s) \rfloor$. To begin with, it is easy to verify that $\Pi \in F$-APX implies $\Pi' \in F$-APX.

Let s be a solution to instance $\mathcal{I}$ of Π of error β. We will show that s as a solution to instance $\mathcal{I}$ of Π' has an error of at least $\beta/2$. Assume, without loss of generality, that $\beta \neq 0$. Then

$$V(\mathcal{I}, s) - \text{OPT}(\mathcal{I}) = \beta \text{OPT}(\mathcal{I}) \geq 1.$$

Multiplying by $2m^2/(\text{OPT}(\mathcal{I})V(\mathcal{I}, s))$, we get

$$\frac{2m^2}{V(\mathcal{I}, s)} - \frac{2m^2}{\text{OPT}(\mathcal{I})} = \beta \frac{2m^2}{V(\mathcal{I}, s)} \geq 2.$$

This implies that

$$\begin{aligned} \frac{2m^2}{\text{OPT}(\mathcal{I})} - \frac{2m^2}{V(\mathcal{I}, s)} &\geq 1 + \frac{1}{2} \times \frac{2m^2}{\text{OPT}(\mathcal{I})} - \frac{2m^2}{V(\mathcal{I}, s)} \\ &= 1 + \frac{\beta}{2} \times \frac{2m^2}{V(\mathcal{I}, s)}. \end{aligned}$$

Upon rearranging,

$$V'(\mathcal{I}, s) \leq \frac{1}{(1 + \beta/2)} \left(\frac{2m^2}{\text{OPT}(\mathcal{I})} - 1 \right) \leq \frac{1}{(1 + \beta/2)} \left\lfloor \frac{2m^2}{\text{OPT}(\mathcal{I})} \right\rfloor.$$

Thus the reduction from Π to Π' is an E-reduction.

Finally, the last remaining case, i.e., Π being a maximization problem and Ω being a minimization problem, is dealt with similarly: we transform Π into a minimization problem Π'. ■

Remark 6 *This theorem appears to merge two different notions of the relative ease of approximation of optimization problems. One such notion would consider a problem* Π_1 *easier than* Π_2 *if there exists an approximation preserving reduction from* Π_1 *to* Π_2*. A different notion would regard* Π_1 *to be easier than* Π_2 *if one seems to have a better factor of approximation than the other. The above statement essentially states that these two comparisons are indeed the same. For instance, the MAX CLIQUE problem and the CHROMATIC NUMBER problem, which are both in* poly-APX*, are inter-reducible to each other. The above observation motivates the search for other interesting function classes* f*, for which the class* f-APX *may contain interesting optimization problems.*

2.5.2 Applications

Theorem 7 can be used to obtain structural characterizations of the classes poly-APX and log-APX. We need to introduce two other syntactic optimization classes.

Definition 25 (RMAX(k) [85]) RMAX(k) *is the class of* NPO *problems expressible as finding a structure* S *which maximizes the objective function*

$$V(\mathcal{I}, S) = \begin{cases} |\{\vec{x} \mid S(\vec{x})\}| & \text{if } \forall \vec{y}, \Phi(\mathcal{I}, S, \vec{y}) \\ 0 & \text{otherwise} \end{cases}$$

where S *is a single predicate and* $\Phi(\mathcal{I}, S, \vec{y})$ *is a quantifier-free CNF formula in which* S *occurs at most* k *times in each clause and all its occurrences are negative.*

The results of Panconesi and Ranjan [85] can be adapted to show that MAX CLIQUE is complete under E-reductions for the class RMAX(2).

Definition 26 (MIN F$^+\Pi_2(k)$ [73]) MIN F$^+\Pi_2(k)$ *is the class of* NPO *problems expressible as finding a structure* S *which minimizes the objective function*

$$V(\mathcal{I}, S) = \begin{cases} |\{\vec{x} : S(\vec{x})\}| & \text{if } \forall \vec{x}, \exists \vec{y}, \Phi(\mathcal{I}, S, \vec{x}, \vec{y}) \\ 0 & \text{otherwise} \end{cases}$$

where S is a single predicate, $\Phi(\mathcal{I}, S, \vec{y})$ is a quantifier-free CNF formula in which S occurs at most k times in each clause and all its occurrences are positive.

The results of Kolaitis and Thakur [73] can be adapted to show that SET COVER is complete under E-reductions for the class MIN $\mathrm{F}^{+}\Pi_2(1)$.

The following may now be concluded from Theorem 7.

Theorem 8

a) $\overline{\text{RMAX(2)}} = \text{poly-APX}$.

b) *If SET COVER is canonically hard to approximate to within a factor of* $\Omega(\log n)$, *then* $\text{log-APX} = \overline{\text{MIN } \mathrm{F}^{+}\Pi_2(1)}$.

We briefly sketch the proof of this theorem. The hardness reduction for MAX SAT and CLIQUE are canonical [7, 37]. The classes APX-PB, poly-APX, log-APX are expressible as classes F-APX for downward closed function families. The problems MAX SAT, MAX CLIQUE and SET COVER are additive. Thus, we can now apply Theorem 7.

Remark 7 *We would like to point out that almost all known instances of hardness results seem to be shown for problems which are additive. In particular, this is true for all MAX SNP problems, MAX CLIQUE, CHROMATIC NUMBER, and SET COVER. Two cases where a hardness result does not seem to directly apply to an additive problem is that of LONGEST PATH [64] and BIN PACKING. In the former case, the closely related LONGEST S-T PATH problem is easily seen to be additive and the hardness result essentially stems from this problem. As for the case of BIN PACKING, which does not admit a PTAS, the hardness result is not of a multiplicative nature and in fact this problem can be approximated to within arbitrarily small factors, provided a small additive error term is allowed. This yields a reason why this problem will not be additive. Lastly, the most interesting optimization problems which do not seem to be additive are problems related to GRAPH BISECTION or PARTITION, and these also happen to be notable instances where no hardness of approximation results have been achieved!*

2.6 Local Search and MAX SNP

In this section we present a formal definition of the paradigm of non-oblivious local search. The idea of non-oblivious local search has been implicitly present in some well-known techniques such as the interior-point methods. We will formalize this idea in context of MAX SNP and illustrate its application to MAX SNP problems. Given a MAX SNP problem Π, recall that the goal is to find a structure S which maximizes the objective function: $V(\mathcal{I}, S) = \sum_{\vec{x}} \Phi(\mathcal{I}, S, \vec{x})$. In the subsequent discussion, we view S as a k-dimensional boolean vector.

2.6.1 Classical Local Search

We start by reviewing the standard mechanism for constructing a local search algorithm. A δ-local algorithm $\mathcal{A}$ for Π is based on a *distance function* $\mathcal{D}(S_1, S_2)$ which is the Hamming distance between two k-dimensional vectors. The *δ-neighborhood* of a structure S is given by $N(S, \delta) = \{S' \subseteq U^n \mid D(S, S') \leq \delta\}$, where U is the universe. A structure S is called *δ-optimal* if $\forall S' \in N(S, \delta)$, we have $V(\mathcal{I}, S) \geq V(\mathcal{I}, S')$. The algorithm computes a δ-optimum by performing a series of greedy improvements to an initial structure S_0, where each iteration moves from the current structure S_i to some $S_{i+1} \in N(S_i, \delta)$ of better value (if any). For constant δ, a δ-local search algorithm for a polynomially-bounded NPO problem runs in polynomial time because:

- each local change is polynomially computable, and
- the number of iterations is polynomially bounded since the value of the objective function improves monotonically by an integral amount with each iteration, and the optimum is polynomially-bounded.

2.6.2 Non-Oblivious Local Search

A non-oblivious local search algorithm is based on a 3-tuple $\langle S_0, \mathcal{F}, \mathcal{D} \rangle$, where S_0 is the initial solution structure which must be independent of the input, $\mathcal{F}(\mathcal{I}, S)$ is a real-valued function referred to as the *weight function*, and $\mathcal{D}$ is a real-valued *distance function* which returns the distance between two structures in some appropriately chosen metric. The weight function $\mathcal{F}$ should be such that the

number of distinct values taken by $\mathcal{F}(\mathcal{I}, S)$ is polynomially bounded in the input size. Moreover, the distance function $\mathcal{D}$ should be such that given a structure S and a fixed δ, $N(S, \delta)$ can be computed in time polynomial in $|S|$. Then, as in classical local search, for constant δ, a non-oblivious δ-local algorithm terminates in time polynomial in the input size.

The classical local search paradigm, which we call *oblivious local search*, makes the natural choice for the function $\mathcal{F}(\mathcal{I}, S)$, and the distance function $\mathcal{D}$, i.e., it chooses them to be $V(\mathcal{I}, S)$ and the Hamming distance. However, as we show later, this choice does not always yield a good approximation ratio. We now formalize our notion of this more general type of local search.

Definition 27 (Non-Oblivious Local Search Algorithm) *A* non-oblivious local search *algorithm is a δ-local search algorithm whose weight function is defined to be*

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

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

A non-oblivious local search can be implemented in polynomial time in much the same way as the oblivious local search. Note that the we are only considering polynomially-bounded weight functions and the profits p_i are fixed independent of the input size. In general, the non-oblivious weight functions do not direct the search in the direction of the actual objective function. In fact, as we will see, this is exactly the reason why they are more powerful and allow for better approximations.

We now define two classes of NPO problems.

Definition 28 (Oblivious GLO) *The class of problems* OBLIVIOUS GLO *consists of all* NPO *problems which can be approximated within constant factors by an oblivious δ-local search algorithm for some constant δ.*

Definition 29 (Non-Oblivious GLO) *The class of problems* NON-OBLIVIOUS GLO *consists of all* NPO *problems which can be approximated within constant factors by a non-oblivious δ-local search algorithm for some constant δ.*

Remark 8 *It would be perfectly reasonable to allow weight functions that are non-linear, but we stay with the above definition for the purposes of our study. Allowing only a constant number of predicates in the weight functions enables us to prevent the encoding of arbitrarily complicated approximation algorithms. The structure S is a k-dimensional vector, and so a natural metric for the distance function $\mathcal{D}$ is the Hamming distance. In fact, classical local search is indeed based on the Hamming metric and this is useful in proving negative results for the paradigm. In contrast, the definition of non-oblivious local search allows for other distance functions, but we will use only the Hamming metric in proving positive results in the remainder of this chapter. However, we have found that it is sometimes useful to modify this, for example by modifying the Hamming distance so that the complement of a vector is considered to be at distance 1 from it. Finally, it is sometimes convenient to assume that the local search makes the best possible move in the bounded neighborhood, rather than an arbitrary move which improves the weight function. We believe that this does not increase the power of non-oblivious local search.*

2.7 The Power of Non-Oblivious Local Search

We will show that there exists a choice of a non-oblivious weight function for MAX k-SAT such that any assignment which is 1-optimal with respect to this weight function, yields a performance ratio of $2^k/(2^k-1)$ with respect to the optimal. But first, we obtain tight bounds on the performance of oblivious local search for MAX 2-SAT, establishing that its performance is significantly weaker than the best-known result even when allowed to search exponentially large neighborhoods. We use the following notation: for any fixed truth assignment $\vec{Z}$, S_i is the set of clauses in which exactly i literals are true; and, for a set of clauses S, $W(S)$ denotes the total weight of the clauses in S.

2.7.1 Oblivious Local Search for MAX 2-SAT

We show a strong separation in the performance of oblivious and non-oblivious local search for MAX 2-SAT. Suppose we use a δ-local strategy with the weight function $\mathcal{F}$ being the total weight of the clauses satisfied by the assignment, i.e., $\mathcal{F} = W(S_1) + W(S_2)$. The following theorem shows that for any $\delta = o(n)$, an oblivious δ-local strategy cannot deliver a performance ratio better

than $3/2$. This is rather surprising given that we are willing to allow near-exponential time for the oblivious algorithm.

Theorem 9 *The asymptotic performance ratio of an oblivious δ-local search algorithm for the* MAX 2-SAT *problem is $3/2$ for any positive integral $\delta = o(n)$. This ratio is still bounded by $5/4$ when δ may take any value less than $n/2$.*

Proof: We first show the existence of an input instance for MAX 2-SAT which may elicit a relatively poor performance ratio for any δ-local algorithm provided $\delta = o(n)$. In our construction of such an input instance, we assume that $n \geq 2\delta + 1$. The input instance comprises of a disjoint union of four sets of clauses, say $\Gamma_1, \Gamma_2, \Gamma_3$ and Γ_4, defined as below:

$$\begin{aligned}
\Gamma_1 &= \bigcup_{1 \leq i < j \leq n} (z_i + \overline{z}_j), \\
\Gamma_2 &= \bigcup_{1 \leq i < j \leq n} (\overline{z}_i + z_j), \\
\Gamma_3 &= \bigcup_{0 \leq i \leq \delta} \zeta_{2i+1}, \\
\Gamma_4 &= \bigcup_{2\delta+2 \leq i \leq n} \zeta_i, \\
\zeta_i &= \bigcup_{i < j \leq n} (\overline{z}_i + \overline{z}_j).
\end{aligned}$$

Clearly, $|\Gamma_1| = |\Gamma_2| = \binom{n}{2}$, and $|\Gamma_3| + |\Gamma_4| = \binom{n}{2} - n\delta + \delta(\delta + 1)$. Without loss of generality, assume that the current input assignment is $\vec{Z} = (1, 1, \ldots, 1)$. This satisfies all clauses in Γ_1 and Γ_2. But none of the clauses in Γ_3 and Γ_4 are satisfied. If we flip the assignment of values to any $k \leq \delta$ variables, it would unsatisfy precisely $k(n-k)$ clauses in $\Gamma_1 + \Gamma_2$. This is the number of clauses in $\Gamma_1 + \Gamma_2$ where a flipped variable occurs with an unflipped variable.

On the other hand, flipping the assigned values of any $k \leq \delta$ variables can satisfy at most $k(n-k)$ clauses in $\Gamma_3 + \Gamma_4$ as we next show.

Let $\Pi(n, \delta)$ denote the set of clauses on n variables given by $\bigcup_{0 \leq i \leq \delta} \zeta_{2i+1} + \bigcup_{2\delta+2 \leq i \leq n} \zeta_i$ where $2\delta + 1 \leq n$. We claim the following.

Lemma 1 *Any assignment of values to the n variables such that at most $k \leq \delta$ variables have been assigned value false, can satisfy at most $k(n-k)$ clauses in $\Pi(n, \delta)$.*

Proof: We prove by simultaneous induction on n and δ that the statement is true for any instance $\Pi(n, \delta)$ where n and δ are non-negative integers such that $2\delta + 1 \leq n$. The base case includes $n = 1$ and $n = 2$ and is trivially verified to be true for the only allowable value of δ, namely $\delta = 0$. We now assume that the statement is true for any instance $\Pi(n', \delta')$ such that $n' < n$ and $2\delta' + 1 \leq n'$. Consider now the instance $\Pi(n, \delta)$. The statement is trivially true for $\delta = 0$. Now consider any $\delta > 0$ such that $2\delta + 1 \leq n$. Let $\{z_{j_1}, z_{j_2}, \ldots z_{j_k}\}$ be any choice of $k \leq \delta$ variables such that $j_p < j_q$ for $p < q$. Again the assertion is trivially true if $k = 0$ or $k = 1$. We assume that $k \geq 2$ from now on. If we delete all clauses containing the variables z_1 and z_2 from $\Pi(n, \delta)$, we get the instance $\Pi(n-2, \delta-1)$. We now consider three cases.

Case 1 [$j_1 \geq 3$]: In this case, we are reduced to the problem of finding an upper bound on the maximum number of clauses satisfied by setting any k variables to false in $\Pi(n-2, \delta-1)$. If $k \leq \delta - 1$, we may use the inductive hypothesis to conclude that no more than $(n-2-k)(k)$ clauses will be satisfied. Thus the assertion holds in this case. However, we may not directly use the inductive hypothesis if $k = \delta$. But in this case we observe that since by the inductive hypothesis, setting any $k-1$ variables in $\Pi(n-2, \delta-1)$ to false, satisfies at most $(n-2-(k-1))(k-1)$ clauses, assigning the value false to any set of k variables, can satisfy at most

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

clauses. Hence the assertion holds in this case also.

Case 2 [$j_1 = 2$]: In this case, z_{j_1} satisfies one clause and the remaining $k-1$ variables satisfy at most $(n-2-(k-1))(k-1)$ clauses by the inductive hypothesis on $\Pi(n-2, \delta-1)$. Adding up the two terms, we see that the assertion holds.

Case 3 [$j_1 = 1$]: We analyze this case based on whether $j_2 = 2$ or $j_2 \geq 3$. If $j_2 = 2$, then z_1 and z_2, together satisfy precisely $n-1$ clauses and the remaining $k-2$ variables, satisfy at most

$(n-2-(k-2))(k-2)$ clauses using the inductive hypothesis. Thus the assertion still holds. Otherwise, z_1 satisfies precisely $n-1$ clauses and the remaining $k-1$ variables satisfy no more than $(n-1-(k-1))(k-1)$ clauses using the inductive hypothesis. Summing up the two terms, we get $(n-k)k$ as the upper bound on the total number of clauses satisfied. Thus the assertion holds in this case also.

To see that this bound is tight, simply consider the situation when the k variables set to false are $z_1, z_3, \ldots, z_{2k-1}$, for any $k \leq \delta$. The total number of clauses satisfied is given by $\sum_{i=1}^{k} |\zeta_{2i-1}| = (n-k)k$. ■

Assuming that each clause has the same weight, Lemma 1 allows us to conclude that a δ-local algorithm cannot increase the total weight of satisfied clauses with this starting assignment. An optimal assignment on the other hand can satisfy all the clauses by choosing the vector $\vec{Z} = (0, 0, \ldots, 0)$. Thus the performance ratio of a δ-local algorithm, say R_δ, is bounded as

$$\begin{aligned} R_\delta &= \frac{|\Gamma_1| + |\Gamma_2| + |\Gamma_3| + |\Gamma_4|}{|\Gamma_1| + |\Gamma_2|} \\ &\leq \frac{3\binom{n}{2} + \delta(\delta+1) - \delta n}{2\binom{n}{2}}. \end{aligned}$$

For any $\delta = o(n)$, this ratio asymptotically converges to $3/2$. We next show that this bound is tight since a 1-local algorithm achieves it. However, before we do so, we make another intriguing observation, namely, for any $\delta < n/2$, the ratio R_δ is bounded by $5/4$.

Now to see that a 1-local algorithm ensures a performance ratio of $3/2$, consider any 1-optimal assignment $\vec{Z}$ and let α_i denote the set of clauses containing the variable z_i such that no literal in any clause of α_i is satisfied by $\vec{Z}$. Similarly, let β_i denote the set of clauses containing the variable z_i such that precisely one literal is satisfied in any clause in β_i and furthermore, it is precisely the literal containing the variable z_i. If we complement the value assigned to the variable z_i, it is exactly the set of clauses in α_i which becomes satisfied and the set of clauses in β_i which is no longer satisfied. Since $\vec{Z}$ is 1-optimal, it must be the case that $W(\alpha_i) \leq W(\beta_i)$. If we sum up this inequality over all the variables, then we get the inequality $\sum_{i=1}^{n} W(\alpha_i) \leq \sum_{i=1}^{n} W(\beta_i)$. We observe that $\sum_{i=1}^{n} W(\alpha_i) = 2W(S_0)$ and $\sum_{i=1}^{n} W(\beta_i) = W(S_1)$ because each clause in S_0 gets counted twice while each clause in S_1 gets counted exactly once. Thus the fractional weight of the

number of clauses not satisfied by a 1-local assignment is bounded as

$$\frac{W(S_0)}{W(S_0)+W(S_1)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)} = \frac{1}{3}.$$

Hence the performance ratio achieved by a 1-local algorithm is bounded from above by $3/2$. Combining this with the upper bound derived earlier, we conclude that $R_1 = 3/2$. This concludes the proof of the theorem. ■

2.7.2 Non-Oblivious Local Search for MAX 2-SAT

We now illustrate the power of non-oblivious local search by showing that it achieves a performance ratio of $4/3$ for MAX 2-SAT, using 1-local search with a simple non-oblivious weight function.

Theorem 10 *Non-oblivious 1-local search achieves a performance ratio of* $4/3$ *for* MAX 2-SAT.

Proof: We use the non-oblivious weight function

$$\mathcal{F}(\mathcal{I}, \vec{Z}) = \frac{3}{2}W(S_1) + 2W(S_2).$$

Consider any assignment $\vec{Z}$ which is 1-optimal with respect to this weight function. Without loss of generality, we assume that the variables have been renamed such that each unnegated literal gets assigned the value true. Let $P_{i,j}$ and $N_{i,j}$ respectively denote the total weight of clauses in S_i containing the literals z_j and $\overline{z}_j$, respectively. Since $\vec{Z}$ is a 1-optimal assignment, each variable z_j must satisfy the following equation.

$$-\frac{1}{2}P_{2,j} - \frac{3}{2}P_{1,j} + \frac{1}{2}N_{1,j} + \frac{3}{2}N_{0,j} \leq 0.$$

Summing this inequality over all the variables, and using

$$\begin{aligned} \sum_{j=1}^{n} P_{1,j} = \sum_{j=1}^{n} N_{1,j} &= W(S_1), \\ \sum_{j=1}^{n} P_{2,j} &= 2W(S_2), \end{aligned}$$

$$\sum_{j=1}^{n} N_{0,j} \quad = \quad 2W(S_0),$$

we obtain the following inequality:

$$W(S_2) + W(S_1) \geq 3W(S_0).$$

This immediately implies that the total weight of the unsatisfied clauses at this local optimum is no more than $1/4$ times the total weight of all the clauses. Thus, this algorithm ensures a performance ratio of $4/3$. ■

Remark 9 *The same result can be achieved by using the oblivious weight function, and instead modifying the distance function so that it corresponds to distances in a hypercube augmented by edges between nodes whose addresses are complement of each other.*

2.7.3 Generalization to MAX k-SAT

We can also design a non-oblivious weight function for MAX k-SAT such that a 1-local strategy ensures a performance ratio of $2^k/(2^k - 1)$. The weight function $\mathcal{F}$ will be of the form $\mathcal{F} = \sum_{i=0}^{k} c_i W(S_i)$ where the coefficients c_i's will be specified later.

Theorem 11 *Non-oblivious 1-local search achieves a performance ratio of* $2^k/(2^k-1)$ *for* MAX k-SAT.

Proof: Again, without loss of generality, we will assume that the variables have been renamed so that each unnegated literal is assigned true under the current truth assignment. Thus the set S_i is the set of clauses with i unnegated literals.

Let $\Delta_i = c_i - c_{i-1}$ and let $\frac{\partial \mathcal{F}}{\partial z_j}$ denote the change in the current weight when we flip the value of z_j, that is, set it to 0. It is easy to verify the following equation:

$$\frac{\partial \mathcal{F}}{\partial z_j} = -\Delta_k P_{k,j} + \sum_{i=k}^{2} (\Delta_i N_{i-1,j} - \Delta_{i-1} P_{i-1,j}) + \Delta_1 N_{0,j} \tag{2.1}$$

Thus when the algorithm terminates, we know that $\frac{\partial \mathcal{F}}{\partial z_j} \leq 0$, for $1 \leq j \leq n$. Summing over all values of j, and using the fact $\sum_{j=1}^{n} P_{i,j} = iW(S_i)$ and $\sum_{j=1}^{n} N_{i,j} = (k-i)W(S_i)$ we get the

following inequality.

$$k\Delta_k W(S_k) + \sum_{i=k-1}^{2} (i\Delta_i - (k-i)\Delta_{i+1}) W(S_i) \geq k\Delta_1 W(S_0). \tag{2.2}$$

We now determine the values of Δ_i's such that the coefficient of each term on the left hand side is unity. It can be verified that

$$\Delta_i = \frac{1}{(k-i+1)\binom{k}{i-1}} \sum_{j=0}^{k-i} \binom{k}{j}$$

achieves this goal. Thus the coefficient of $W(S_0)$ on the right hand side of equation (2.2) is $2^k - 1$. Clearly, the weight of the clauses not satisfied is bounded by $1/2^k$ times the total weight of all the clauses. It is worthwhile to note that this is regardless of the value chosen for the coefficient c_0. ■

2.8 Local Search for CSP and MAX SNP

We now introduce a class of constraint satisfaction problems such that the problems in MAX SNP are exactly equivalent to the problems in this class. Furthermore, every problem in this class can be approximated to within a constant factor by a non-oblivious local search algorithm.

2.8.1 Constraint Satisfaction Problems

The connection between the syntactic description of optimization problems and their approximability through non-oblivious local search is made via a problem called MAX k-CSP which captures all the problems in MAX SNP as a special case.

Definition 30 (k-ary Constraint) *Let $Z = \{z_1, \ldots, z_n\}$ be a set of boolean variables. A k-ary constraint on Z is $C = (V; P)$, where V is a size k subset of Z, and $P : \{T, F\}^k \rightarrow \{T, F\}$ is a k-ary boolean predicate.*

Definition 31 (MAX k-CSP) *Given a collection C_1, ..., C_m of weighted k-ary constraints over the variables $Z = \{z_1, \ldots, z_n\}$, the* MAX k-CSP *problem is to find a truth assignment satisfying a maximum weight sub-collection of the constraints.*

The following theorem shows that MAX k-CSP problem is a "universal" MAX SNP problem, in that it contains as special cases all problems in MAX SNP.

Theorem 12

a) *For fixed k,* MAX k-CSP $\in$ MAX SNP.

b) *Let* $\Pi \in$ MAX SNP. *Then, for some constant k,* Π *is a* MAX k-CSP *problem. Moreover, the k-*CSP *instance corresponding to any instance of this problem can be computed in polynomial time.*

Proof: The proof of part (b) is implicit in Theorem 1 in [87], and so we concentrate on proving part (a). Our goal is to obtain a representation of the k-CSP problem in the MAX SNP syntax:

$$\max_S |\{x \mid \Phi(\mathcal{I}, S, x)\}|.$$

The input structure is $\mathcal{I} = (Z \cup \{T, F\} \cup \text{MAX}; \{\text{ARG}, \text{EVAL}\})$, where $Z = \{z_1, \ldots, z_n\}$, MAX contains the integers $[1, \max\{k, n, m\}]$, the predicate ARG encodes the sets V_i, and the predicate EVAL encodes the predicates P_i, as described below.

- ARG(r, s, z_t) is 3-ary predicate which is true if and only if the rth argument of C_s is the variable z_t, for $1 \leq r \leq k$, $1 \leq s \leq m$, and $1 \leq t \leq n$.
- EVAL$(s, v_1, \ldots, v_k)$ is a $(k+1)$-ary predicate which is true if and only if $P_s(v_1, \ldots, v_k)$ evaluates to true, for $1 \leq s \leq m$ and all $v_i \in \{T, F\}$.

The structure S is defined as $(Z; \{\text{TRUE}\})$, where TRUE is a unary predicate which denotes an assignment of truth values to the variables in Z. The vector x has $k+1$ components which will be called $x_1, \ldots, x_k$ and s, for convenience. The intention is that the x_i's refer to the arguments of the sth constraint.

All that remains is to specify the quantifier-free formula Φ. The basic idea is that $\Phi(\mathcal{I}, S, x)$ should evaluate to true if and only if the following two conditions are satisfied:

- the arguments of the constraint C_s are given by the variables $x_1, \ldots, x_k$, in that order; and,

- the values given to these variables under the truth assignment specified by S are such that the constraint is satisfied.

The formula Φ is given by the following expression, with the two sub-formulas ensuring these two conditions.

$$\left(\bigwedge_{r=1}^{k} \text{ARG}(r, s, x_r)\right) \wedge \left(\bigvee_{v_1,\ldots,v_k \in \{T,F\}} \left(\text{EVAL}(s, v_1, \ldots, v_k) \wedge \left(\bigwedge_{r=1}^{k} v_r \Leftrightarrow \text{TRUE}(x_r)\right)\right)\right)$$

It is easy to see that the first sub-formula has the desired effect of checking that the x_r's correspond to the arguments of C_s. The second sub-formula considers all possible truth assignment to these k variables, and checks that the particular set of values assigned by the structure S will make P_s evaluate to true.

For a fixed structure S, there is exactly one choice of x per constraint that could make Φ evaluate to true, and this is happens if and only if that constraint is satisfied. Thus, the value of the solution given by any particular truth assignment structure S is exactly the number of constraints that are satisfied. This shows that the MAX SNP problem always has the same value as intended in the k-CSP problem.

Finally, there are still a few things which need to be checked to ensure that this is a valid MAX SNP formulation. Notice that all the predicates are of bounded arity and the structures consist of a bounded number of such predicates, i.e., independent of the input size which is given by MAX. Further, although the length of the formula is exponential in k, it is independent of the input. ■

2.8.2 Non-Oblivious Local Search for MAX k-CSP

A suitable generalization of the non-oblivious local search algorithm for MAX k-SAT yields the following result.

Theorem 13 *A non-oblivious 1-local search algorithm has performance ratio* 2^k *for* MAX k-CSP.

Proof: We use an approach similar to the one used in the previous section to design a non-oblivious weight function $\mathcal{F}$ for the weighted version of the MAX k-CSP problem such that a 1-local

algorithm yields 2^k performance ratio to this problem.

We consider only the constraints with at least one satisfying assignment. Each such constraint can be replaced by a monomial which is the conjunction of some k literals such that when the monomial evaluates to true the corresponding literal assignment represents a satisfying assignment for the constraint. Furthermore, each such monomial has precisely one satisfying assignment. We assign to each monomial the weight of the constraint it represents. Thus any assignment of variables which satisfies monomials of total weight W_0, also satisfies constraints in the original problem of total weight W_0.

Let S_i denote the monomials with i true literals, and assume that the weight function $\mathcal{F}$ is of the form $\sum_{i=1}^{k} c_i W(S_i)$. Thus, assuming that the variables have been renamed so that the current assignment gives value true to each variable, we know that for any variable z_j, $\frac{\partial \mathcal{F}}{\partial z_j}$ is given by equation (2.1). As before, using the fact that for any 1-optimal assignment, $\frac{\partial \mathcal{F}}{\partial z_j} \leq 0$ for $1 \leq j \leq n$, and summing over all values of j, we can write the following inequality.

$$k\Delta_1 W(S_0) + \sum_{i=2}^{k-1}((k-i)\Delta_{i+1} - i\Delta_i)W(S_i) \leq k\Delta_k W(S_k)\ . \tag{2.3}$$

We now determine the values of Δ_i's such that the coefficient of each term on the left hand side is unity. It can be verified that

$$\Delta_i = \frac{1}{i\binom{k}{i}} \sum_{j=0}^{i-1} \binom{k}{j}$$

achieves this goal. Thus the coefficient of $W(S_k)$ on the right hand side of equation (2.1) is $2^k - 1$. Clearly, the total weight of clauses satisfied is at least $1/2^k$ times the total weight of all the clauses with at least one satisfiable assignment. ■

We conclude the following theorem.

Theorem 14 *Every optimization problem* $\Pi \in$ MAX SNP *can be approximated to within some constant factor by a (uniform) non-oblivious 1-local search algorithm, i.e.,*

$$\text{MAX SNP} \subseteq \text{NON-OBLIVIOUS GLO}.$$

For a problem expressible as k-CSP*, the performance ratio is at most* 2^k.

2.9 Non-Oblivious versus Oblivious GLO

In this section, we show that there exist problems for which no constant factor approximation can be obtained by any δ-local search algorithm with oblivious weight function, even when we allow δ to grow with the input size. However, a simple 1-local search algorithm using an appropriate non-oblivious weight function can ensure a constant performance ratio.

2.9.1 MAX 2-CSP

The first problem is an instance of MAX 2-CSP where we are given a collection of monomials such that each monomial is an "and" of precisely two literals. The objective is to find an assignment to maximize the number of monomials satisfied.

We show an instance of this problem such that for every $\delta = o(n)$, there exists an instance one of whose local optima has value that is a vanishingly small fraction of the global optimum.

The input instance consists of a disjoint union of two sets of monomials, say Γ_1 and Γ_2, defined as below:

$$\begin{aligned} \Gamma_1 &= \bigcup_{1 \le i < j \le n} (\overline{z}_i \wedge \overline{z}_j), \\ \Gamma_2 &= \bigcup_{1 \le i \le \delta} \bigcup_{i < j \le n} (z_i \wedge z_j). \end{aligned}$$

Clearly, $|\Gamma_1| = \binom{n}{2}$, and $|\Gamma_2| = n\delta - \binom{\delta+1}{2}$. Consider the truth assignment $\vec{Z} = (1, 1, \ldots, 1)$. It satisfies all monomials in Γ_2 but none of the monomials in Γ_1. We claim that this assignment is δ-optimal with respect to the oblivious weight function. To see this, observe that complementing the value of any $p \le \delta$ variables will unsatisfy at least $\delta p/2$ monomials in Γ_2 for any $\delta = o(n)$. On the other hand, this will satisfy precisely $\binom{p}{2}$ monomials in Γ_1. For any $p \le \delta$, we have $(\delta p)/2 \ge \binom{p}{2}$, and so Z is a δ-local optimum.

The optimal assignment on the other hand, namely $\vec{Z}_{\mathrm{OPT}} = (0, 0, \ldots, 0)$, satisfies all monomials in Γ_1. Thus, for $\delta < n/2$, the performance ratio achieved by any δ-local algorithm is no more than $\binom{n}{2}/(n\delta - \binom{\delta+1}{2})$ which asymptotically diverges to infinity for any $\delta = o(n)$. We have already seen in Section 2.8 that a 1-local non-oblivious algorithm ensures a performance ratio of 4 for this

problem. Since this problem is in MAX SNP, we obtain the following theorem.

Theorem 15 *There exist problems in* MAX SNP *such that for $\delta = o(n)$, no δ-local oblivious algorithm can approximate them to within a constant performance ratio, i.e.,*

$$\text{MAX SNP} \not\subseteq \text{OBLIVIOUS GLO}.$$

2.9.2 Vertex Cover

Ausiello and Protasi [10] have shown that VERTEX COVER does not belong to the class GLO and, hence, there does not exist any constant δ such that an oblivious δ-local search algorithm can compute a constant factor approximation. In fact, their example can be used to show that for any $\delta = o(n)$, the performance ratio ensured by δ-local search asymptotically diverges to infinity. However, we show that there exists a rather simple non-oblivious weight function which ensures a factor 2 approximation via a 1-local search. In fact, the algorithm simply enforces the behavior of the standard approximation algorithm which iteratively builds a vertex cover by simply including both end-points of any currently uncovered edge.

We assume that the input graph G is given as a structure $(V, \{E\})$ where V is the set of vertices and $E \subseteq V \times V$ encodes the edges of the graph. Our solution is represented by a 2-ary predicate M which is iteratively constructed so as to represent a maximal matching. Clearly, the end-points of any maximal matching constitute a valid vertex cover and such a vertex cover can be at most twice as large as any other vertex cover in the graph. Thus M is an encoding of the vertex cover computed by the algorithm.

The algorithm starts with M initialized to the empty relation and at each iteration, at most one new pair is included in it. The non-oblivious weight function used is as below:

$$\mathcal{F}(\mathcal{I}, M) \;=\; \sum_{\langle x,y,z\rangle \in V^3} [\Phi_1(x,y,z) - 2\Phi_2(x,y,z) - \Phi_3(x,y,z)],$$

where

$$\Phi_1(x,y,z) \;=\; (M(x,y) \wedge E(x,y) \wedge (x = z)),$$

$$\begin{aligned}\Phi_2(x,y,z) &= (M(x,y) \wedge M(x,z)),\\ \Phi_3(x,y,z) &= (M(x,y) \wedge \overline{E(x,y)}).\end{aligned}$$

Let M encode a valid matching in the graph G. We make the following observations.

- Any relation M' obtained from M by either deleting an edge from it, or including an edge which is incident on an edge of M, or including a non-existent edge, has the property that $\mathcal{F}(\mathcal{I}, M') \leq \mathcal{F}(\mathcal{I}, M)$. Thus in a 1-local search from M, we will never move to a relation M' which does not encode a valid matching of G.
- On the other hand, if a relation M' corresponds to the encoding of a matching in G which is larger than the matching encoded by M, then $\mathcal{F}(\mathcal{I}, M') > \mathcal{F}(\mathcal{I}, M)$. Thus if M does not encode a maximal matching in G, there always exist a relation in its 1-neighborhood of larger weight than itself.

These two observations, combined with the fact that we start with a valid initial matching (the empty matching), immediately allow us to conclude that any 1-optimal relation M always encodes a maximal matching in G. We have established the following.

Theorem 16 *A 1-local search algorithm using the above non-oblivious weight function achieves a performance ratio of* 2 *for the VERTEX COVER problem.*

Theorem 17 GLO *is a* strict *subset of* NON-OBLIVIOUS GLO**.**

As an aside, it can be seen that this algorithm has the same performance starting with an arbitrary initial solution. This is because for any relation M not encoding a matching of the input graph, deleting one of the violating members strictly increases $\mathcal{F}(\mathcal{I}, M)$.

2.10 The Traveling Salesman Problem

The TSP(1,2) problem is the traveling salesman problem restricted to complete graphs where all edge weights are either 1 or 2; clearly, this satisfies the triangle inequality. Papadimitriou and

Yannakakis [88] showed that this problem is hard for MAX SNP. The natural weight function for TSP(1,2), that is, the weight of the tour, can be used to show that a 4-local algorithm yields a $3/2$ performance ratio. The algorithm starts with an arbitrary tour and in each iteration, it checks if there exist two disjoint edges (a, b) and (c, d) on the tour such that deleting them and replacing them with the edges (a, c) and (b, d) yields a tour of lesser cost.

Theorem 18 *A 4-local search algorithm using the oblivious weight function achieves a* $3/2$ *performance ratio for* TSP(1,2).

Proof: Let C be a 4-optimal solution and let π be a permutation such that the vertices in C occur in the order $v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}$. Consider any optimal solution O. With each unit cost edge e in O, we associate a unit cost edge e' in C as follows. Let $e = (v_{\pi_i}, v_{\pi_j})$ where $i < j$. If $j = i + 1$ then $e' = e$. Otherwise, consider the edges $e_1 = (v_{\pi_i}, v_{\pi_{i+1}})$ and $e_2 = (v_{\pi_j}, v_{\pi_{j+1}})$ on C. We claim either e_1 or e_2 must be of unit cost. Suppose not, then the tour C' which is obtained by simply deleting both e_1 and e_2 and inserting the edges e and $f = (v_{\pi_{i+1}}, v_{\pi_{j+1}})$ has cost at least one less than C. But C is 4-optimal and thus this is a contradiction.

Let U_O denotes the set of unit cost edges in O and let U_C be the set of unit cost edges in C which form the image of U_O under the above mapping. Since an edge $e' = (v_{\pi_i}, v_{\pi_{i+1}})$ in U_C can only be the image of unit cost edges incident on v_{π_i} in O and since O is a tour, there are at most two edges in U_O which map to e'. Thus $|U_C| \geq |U_O|/2$ and hence

$$\frac{cost(O)}{cost(C)} \geq \frac{|U_O| + 2(n - |U_O|)}{|U_O|/2 + 2(n - |U_O|/2)} \geq \frac{2}{3}.$$

■

In fact, the above bound can be shown to be tight.

2.11 Maximum Independent Sets in Bounded Degree Graphs

The input instance to the maximum independent set problem in bounded degree graphs, denoted MIS-B, is a graph G such that the degree of any vertex in G is bounded by a constant Δ. We present an algorithm with performance ratio $(\sqrt{8\Delta^2 + 4\Delta + 1} - 2\Delta + 1)/2$ for this problem when $\Delta \geq 10$.

Our algorithm uses two local search algorithms such that the larger of the two independent sets computed by these algorithms, gives us the above claimed performance ratio. We refer to these two algorithms as $\mathcal{A}_1$ and $\mathcal{A}_2$.

In our framework, the algorithm $\mathcal{A}_1$ can be characterized as a 3-local algorithm with the weight function simply being $|I| - 3|(I \times I) \cap E|$. Thus if we start with I initialized to empty set, it is easy to see that at each iteration, I will correspond to an independent set in G. A convenient way of looking at this algorithm is as follows. We define an $i \leftrightarrow j$ swap to be the process of deleting i vertices from S and including j vertices from the set $V - S$ to the set S. In each iteration, the algorithm $\mathcal{A}_1$ performs either a $0 \leftrightarrow j$ swap where $1 \leq j \leq 3$, or a $1 \leftrightarrow 2$ swap. A $0 \leftrightarrow j$ swap however, can be interpreted as j applications of $0 \leftrightarrow 1$ swaps. Thus the algorithm may be viewed as executing a $0 \leftrightarrow 1$ swap or a $1 \leftrightarrow 2$ swap at each iteration. The algorithm terminates when neither of these two operations is applicable.

Let I denote the 3-optimal independent set produced by the algorithm $\mathcal{A}_1$. Furthermore, let O be any optimal independent set and let $X = I \cap O$. We make the following useful observations.

- Since for no vertex in I, a $0 \leftrightarrow 1$ swap can be performed, it implies that each vertex in $V - I$ must have at least one incoming edge to I.
- Similarly, since no $1 \leftrightarrow 2$ swaps can be performed, it implies that at most $|I - X|$ vertices in $O - I$ can have precisely one edge coming into I. Thus $|O - X| - |I - X| = |O| - |I|$ vertices in $O - X$ must have at least two edges entering the set I.

A rather straightforward consequence of these two observations is the following lemma.

Lemma 2 *The algorithm $\mathcal{A}_1$ has performance ratio $(\Delta + 1)/2$ for* MIS-B.

Proof: The above two observations imply that the minimum number of edges entering I from the vertices in $O - X$ is $|I - X| + 2(|O| - |I|)$. On the other hand, the maximum number of edges coming out of the vertices in I to the vertices in $O - X$ is bounded by $|I - X|\Delta$. Thus we must have

$$|I - X|\Delta \geq |I - X| + 2(|O| - |I|)\,.$$

Rearranging, we get

$$\frac{|I|}{|O|} \geq \frac{2}{\Delta+1} + \frac{|X|(\Delta-1)}{|O|(\Delta+1)},$$

which yields the desired result. ■

This nearly matches the approximation ratio of $\Delta/2$ due to Hochbaum [53]. It should be noted that the above result holds for a broader class of graphs, viz., k-claw free graphs. A graph is called *k-claw free* if there does not exist an independent set of size k or larger such that all the vertices in the independent set are adjacent to the same vertex. Lemma 2 applies to $(\Delta+1)$-claw free graphs.

Our next objective is to further improve this ratio by using the algorithm $\mathcal{A}_1$ in combination with the algorithm $\mathcal{A}_2$. The following lemma uses a slightly different counting argument to give an alternative bound on the approximation ratio of the algorithm $\mathcal{A}_1$ when there is a constraint on the size of the optimal solution.

Lemma 3 *For any real number $c < \Delta$, the algorithm $\mathcal{A}_1$ has performance ratio $(\Delta - c)/2$ for* MIS-B *when the optimal value itself is no more than $((\Delta - c)|V|)/(\Delta + c + 4)$.*

Proof: As noted earlier, each vertex in $V - I$ must have at least one edge coming into the set I and at least $|O| - |I|$ vertices in O must have at least two edges coming into I. Therefore, the following inequality must be satisfied:

$$|I|\Delta \geq |V| - |I| + |O| - |I| \,.$$

Thus $|I| \geq (|V| + |O|)/(\Delta + 2)$. Finally, observe that

$$\frac{|V| + |O|}{\Delta + 2} \geq \frac{2}{\Delta - c}|O|$$

whenever $|O| \leq (\Delta - c)|V|/(\Delta + c + 4)$. ■

The above lemma shows that the algorithm $\mathcal{A}_1$ yields a better approximation ratio when the size of the optimal independent set is relatively small.

The algorithm $\mathcal{A}_2$ is simply the classical greedy algorithm. This algorithm can be conveniently included in our framework if we use directed local search. If we let $N(I)$ denote the set of neighbors

of the vertices in I, then the weight function is simply $|I|(\Delta+1) + |V - (I + N(I))| - |(I \times I) \cap E|(\Delta+1)$. It is not difficult to see that starting with an empty independent set, a 1-local algorithm with directed search on above weight function simply simulates a greedy algorithm. The greedy algorithm exploits the situation when the optimal independent set is relatively large in size. It does so by using the fact that the existence of a large independent set in G ensures a large subset of vertices in G with relatively small average degree. The following two lemmas characterize the performance of the greedy algorithm.

Lemma 4 *Suppose there exists an independent set $X \subseteq V$ such that the average degree of vertices in X is bounded by α. Then for any $\alpha \geq 1$, the greedy algorithm produces an independent set of size at least $|X|/(1+\alpha)$.*

Proof: The greedy algorithm iteratively chooses a vertex of smallest degree in the remaining graph and then deletes this vertex and all its neighbors from the graph. We examine the behavior of the greedy by considering two types of iterations. First consider the iterations in which it picks a vertex outside X. Suppose in the ith such iteration, it picks a vertex in $V - X$ with exactly k_i neighbors in the set X in the remaining graph. Since each one of these k_i vertices must also have at least k_i edges incident on them, we loose at least k_i^2 edges incident on X. Suppose only p such iterations occur and let $\sum_{i=1}^{p} k_i = x$. We observe that $\sum_{i=1}^{p} k_i^2 \leq \alpha|X|$. Secondly, we consider the iterations when the greedy selects a vertex in X. Then we do not loose any other vertices in X because X is an independent set. Thus the total size of the independent set constructed by the greedy algorithm is at least $p+q$ where $q = |X| - x$.

By the Cauchy-Schwartz inequality, $\sum_{i=1}^{p} k_i^2 \geq x^2/p$. Therefore, we have $(1+\alpha)|X| \geq x^2/p + x$. Rearranging, we obtain that

$$p \geq \frac{x^2}{(1+\alpha)|X| - x} \geq \frac{x^2}{(1+\alpha)|X|} \geq \frac{|X|}{1+\alpha} + \frac{q^2}{(1+\alpha)|X|} - \frac{2q}{1+\alpha}.$$

Thus

$$p + q \geq \frac{|X|}{1+\alpha} + \frac{q^2}{(1+\alpha)|X|} - \frac{2q}{1+\alpha} + q.$$

But $2q/(1+\alpha) \leq q$ for $\alpha \geq 1$, and the result follows. ■

Lemma 5 *For* $\Delta \geq 10$ *and any non-negative real number* $c \leq 3\Delta - \sqrt{8\Delta^2 + 4\Delta + 1} - 1$, *the algorithm* $\mathcal{A}_2$ *has performance ratio* $(\Delta - c)/2$ *for* MIS-B *when the optimal value itself is at least* $((\Delta - c)|V|)/(\Delta + c + 4)$.

Proof: Observe that the average degree of vertices in O is bounded by $(|V - O|\Delta/|O|)$ and thus using the fact that $|O| \geq (\Delta - c)|V|/(\Delta + c + 4)$, we know that the algorithm $\mathcal{A}_2$ computes an independent set of size at least $|O|/(1 + \alpha)$ where $\alpha = (4\Delta + 2\Delta c)/(\Delta - c)$, and $\alpha \geq 1$ for $c \geq 0$. Hence it is sufficient to determine the range of values c can take such that the following inequality is satisfied:

$$\frac{|O|}{1+\alpha} \geq \left(\frac{2}{\Delta - c}\right)|O|.$$

Substituting the bound on the value of α and rearranging the terms of the equation, yields the following quadratic equation :

$$c^2 - (6\Delta - 2)c + \Delta^2 - 10\Delta \geq 0\ .$$

Since c must be strictly bounded by Δ, the above quadratic equation is satisfied for any choice of $c \leq 3\Delta - \sqrt{8\Delta^2 + 4\Delta + 1} - 1$ if $\Delta \geq 10$. ■

Combining the results of Lemmas 3 and 5 and choosing the largest allowable value for c, we get the following result.

Theorem 19 *An approximation algorithm which simply outputs the larger of the two independent sets computed by the algorithms* $\mathcal{A}_1$ *and* $\mathcal{A}_2$, *has performance ratio* $(\sqrt{8\Delta^2 + 4\Delta + 1} - 2\Delta + 1)/2$ *for* MIS-B.

The performance ratio claimed above is essentially $\Delta/2.414$. This improves upon the long-standing approximation ratio of $\Delta/2$ due to Hochbaum [53], when $\Delta \geq 10$. However, more recently, there has been a flurry of new results for this problem. Berman and Furer [18] have given an algorithm with performance ratio $(\Delta + 3)/5 + \epsilon$ when Δ is even, and $(\Delta + 3.25)/5 + \epsilon$ for odd Δ, where $\epsilon > 0$ is a fixed constant. Halldorsson and Radhakrishnan [50] have shown that algorithm $\mathcal{A}_1$ when run on k-clique free graphs, yields an independent set of size at least $2n/(\Delta + k)$. They

combine this algorithm with a clique-removal based scheme to achieve a performance ratio of $\Delta/6(1+o(1))$.

In conclusion, note that Khanna, Motwani and Vishwanathan [69] have recently shown that a semi-definite programming technique can be used to obtain a $(\Delta \log\log \Delta)/(\log \Delta)$-approximation algorithm for this problem.

Chapter 3

Towards a Structural Characterization of PTAS

3.1 Introduction *

In the preceding chapter, we saw that we can use syntactic classes to provide a structural basis for many natural approximation classes such as APX, log-APX and poly-APX. The starting point of our study was the availability of syntactic classes which were contained in these approximation classes and contained problems which were hard to approximate beyond the factors allowed for in the definition of the corresponding classes. In this context, it is interesting to attempt a similar characterization of other approximation classes as the closure of a syntactic class. In this chapter, we focus on the class PTAS. The starting point for our investigation is the remarkable paper by Baker [13] which presents PTAS's for a surprisingly large number of NP-hard graph optimization problems *when the input graph is planar*. It is our contention that the defining characteristic of PTAS problems is a "restriction on input structure" for some syntactic classes. Our work provides evidence towards this claim, raising the possibility that this input restriction may well be a notion related to planarity.

This may appear surprising at first considering that several well-known PTAS problems do not exhibit a graph structure, leave alone a planar graph structure, e.g., knapsack and multiprocessor scheduling. However, we will demonstrate that these problems are in fact special cases of generic problems arising from a planar input-restriction of generalizations of classes such as MAX SNP. It appears that most problems in PTAS fit directly into our framework. Moreover, our framework trivially implies membership in PTAS for problems such as PLANAR MAX SAT that were not known to be in PTAS earlier.

We identify a family of syntactic classes with input restrictions that subsume planar-restricted versions of classes such as MAX SNP, RMAX(2), and MIN $F^{+}\Pi_2(1)$. We then demonstrate that these classes are contained in PTAS using a collection of techniques that considerably generalize (and simplify) the idea used by Baker [13]. Our techniques are based on a separator theorem for outerplanar graphs due to Bodlaender [21] that strengthens the planar separator theorem of Lipton and Tarjan [79]. While almost every known problem in PTAS is easily seen to belong to our syntactic classes, there are a few problems, namely bin packing, planar graph TSP and some optimization

* This chapter is based on joint work with Rajeev Motwani [68].

problems on dense instances, that do not fit into our framework. We point out that, strictly speaking, many of these problems do not belong to PTAS, and hence would not serve as counter-examples to the possibility that some extension of our framework (via closure) is a characterization of PTAS.

Before describing our results in greater detail, we briefly review some earlier results concerning the role of planarity in PTAS. Lipton and Tarjan [80] were the first to present a PTAS for an NP-hard optimization problem on planar graphs, namely, maximum independent sets (MIS) in planar graphs. Subsequently, Chiba, Nishizeki, and Saito [24] extended this to finding a maximum induced subgraph satisfying a hereditary property determined by connected components.

The essential idea in results of this nature is that a recursive application of the planar separator theorem allows one to delete $o(n)$ vertices from a given planar graph G, so as to create a graph G' such that every component of G' has size $O(\log n)$. Then, it is possible to compute optimal solutions in each component in polynomial time, and the union of these optimal solutions is guaranteed to be a solution to the original problem. Using the fact that any such property must have a solution of size $\Omega(n)$, we can conclude that the resulting solution is a $(1-\epsilon)$ approximation, for any fixed $\epsilon > 0$ and sufficiently large n.

Hunt *et al* [54] applied Baker's technique to satisfiability problems where each clause consists of an application of a bounded-arity function from a given set of boolean functions. They also showed that for certain MAX SNP-complete problems, such as bounded-degree MIS, Baker's results can be extended to δ-near-planar graphs, i.e., graphs embeddable in the plane with at most δn edge crossings. They also show that Baker's framework can be suitably adapted to yield parallel approximation schemes.

Recently, Grigni, Koustsoupias, and Papadimitriou [48] gave a PTAS for TSP on unweighted planar graphs. Their algorithm combines a new separator theorem with a careful decomposition to create such an approximation scheme. Soon thereafter, Arora [3] showed a PTAS for the Euclidean TSP problem.

The rest of this chapter is organized as follows. Section 3.2 gives an overview of the results and discusses some applications and directions for extending our results. Section 3.3 reviews basic concepts related to outerplanar graphs and tree decompositions. Finally, in Sections 3.4, 3.5, 3.6, and 3.7, we sketch the PTAS results for our proposed classes.

3.2 Summary of Results and the Implications

We begin by defining a new family of syntactic subclasses of NPO that subsume the known syntactically-defined optimization classes such as MAX SNP, MAX NP, RMAX and MIN $F^+\Pi_2$ (see Chapters 1 and 2 for definitions). Let $x_1, \ldots, x_n$ be a set of boolean variables. A *positive literal* is a variable x_i, and a *negative literal* is a negated variable $\overline{x}_i$. We use the abbreviation FOF to denote quantifier-free first-order formulas over these literals. We associate a weight with each FOF and with each variable; depending on the problem at hand, we may refer to the weights as *profits* or *costs*. The weight of a collection $\mathcal{C}$ of FOF's for a given truth assignment T is the net weight of the satisfied formulas, and the weight of T is the net weight of the variables assigned TRUE.

Definition 32 (Positive and Negative Minterms)*A* minterm *is any conjunction of literals. A minterm is* positive *if it has only positive literals, and is* negative *if it has only negative literals.*

The following are the new *syntactic* classes of optimization problems. The class MPSAT captures constraint satisfaction problems where the objective is to compute solutions satisfying as many constraints as possible. It contains the usual satisfiability problems such as MAX 3-SAT. The other two classes, TMAX and TMIN, capture problems where a feasible solution must satisfy given constraints, and the objective is to find extremal feasible solutions. For instance, the MIS problem requires a solution containing an extremal number of vertices that satisfy constraints ruling out the inclusion of both end-points of an edge. We restrict our attention to monadic structures. The latter two classes complement each other, and are essentially dual to the class MPSAT. Naturally, we do not consider the minimization counterpart of MPSAT (where the goal is to *minimize* the number of unsatisfied constraints) since it contains problems that are NP-hard for small optimum values *even for planar input-restriction* [77].

Definition 33 (MPSAT) *The class* MPSAT *consists of all* NPO *problems expressible as: given a collection $\mathcal{C}$ of* FOF*s over n variables such that each formula $\Phi \in \mathcal{C}$ is a disjunction of $O(n^{O(1)})$ minterms, find a truth assignment T of weight at most W maximizing the total weight of the* FOF*s in $\mathcal{C}$ that are satisfied.*

Definition 34 (TMAX) *The class* TMAX *consists of* NPO *problems expressible as: given a collection $\mathcal{C}$ of* FOF*s over n variables such that each formula $\Phi \in \mathcal{C}$ is a disjunction of $O(n^{O(1)})$* negative *minterms, find a maximum weight truth assignment T that satisfies each* FOF *in $\mathcal{C}$.*

Definition 35 (TMIN]) *The class* TMIN *consists of* NPO *problems expressible as: given a collection $\mathcal{C}$ of* FOF*s over n variables such that each formula $\Phi \in \mathcal{C}$ is a disjunction of $O(n^{O(1)})$* positive *minterms, find a minimum weight truth assignment T that satisfies each* FOF *in $\mathcal{C}$.*

Observe that these classes are fairly strong generalizations of existing syntactic classes, in that they allow each constraint to depend on an unbounded number of variables and also allow each minterm to be of unbounded size. In Section 3.7, we further generalize these definitions to problems over multi-valued domains allowing Min-Max objective functions, thereby obtaining the class MINMAX. This class captures problems such as multiprocessor scheduling.

We now endow these definitions with a restriction on the structure of the input formulas. This restriction is formulated in terms of incidence graphs. The classes of interest are those where the instances are restricted to having planar incidence graphs.

Definition 36 (Incidence Graph) *Let $\mathcal{I}$ be an instance of a problem Π in* MPSAT, TMAX, TMIN, *or* MINMAX. *Then, the* incidence graph *of $\mathcal{I}$, denoted $G(\mathcal{I}) = (V, E)$, is defined as follows:*

- *V has a v-vertex for each variable, and an f-vertex for each* FOF*;*
- *for each* FOF *$\Phi \in \mathcal{S}$, and each variable x_i in Φ, there is an edge between the v-vertex for x_i and the f-vertex for Φ.*

Definition 37 (Planar Input-Restructions) *Let $\mathcal{S}$ be any syntactic optimization class. The class* PLANAR *$\mathcal{S}$ is the class $\mathcal{S}$ restricted to instances with planar incidence graphs.*

3.2.1 Main Results and Implications

Our main results are summarized in the following proposition and brief proof sketches are given in the following sections. A similar result can be obtained for the class k-OUTERPLANAR MINMAX defined in Section 3.7; we omit the detailed exposition of this and other extensions.

Proposition 1 *Each of* PLANAR MPSAT, PLANAR TMAX, *and* PLANAR TMIN *is contained in* PTAS.

Some representative applications of this proposition are listed below.

- PLANAR MAX SAT belongs to PLANAR MPSAT and hence has a PTAS. This is a new result. We believe that no PTAS was known earlier for any class of satisfiability problems involving constraints with an *unbounded* number of variables. The most general previous result, due to Hunt *et al* [54], was for formulas of bounded-arity drawn from a fixed family.

- The *knapsack* problem is in PLANAR MPSAT. A variable encodes the placement of an item in the knapsack, and a formula encodes the profit obtained by placing an item in the knapsack. The incidence graph is a matching and hence planar. This explains the PTAS [94] for knapsack.

- The *weighted MIS* problem in planar graphs is in PLANAR TMAX. Vertices are encoded by variables, and for each edge $e = (x, y)$ there is a formula $\Phi_e = (\overline{x}+\overline{y})$. For a planar graph G, the resulting incidence graph is also planar. This explains the PTAS [13] for planar weighted MIS.

- The *minimum weighted vertex cover (WVC)* problem in planar graphs is in PLANAR TMIN. Vertices are encoded by variables, and for each edge $e = (x, y)$ there is a formula $\Phi_e = (x+y)$. For a planar input graphs, the incidence graph is also planar. This explains the PTAS [13] for planar WVC.

- *Multiprocessor scheduling* on m machines, for any fixed m, is in OUTERPLANAR MINMAX with an empty set of constraints. This explains the PTAS [47] for multiprocessor scheduling.

- Consider the problem of finding the maximum induced subgraph satisfying a hereditary property Π determined by components [24]. For planar graphs, this can be placed in the class PLANAR TMAX by an approximation-preserving reduction such as the E-reduction [67]. The idea is to first apply planar separator theorem to delete $o(n)$ vertices such that the largest

component size is only $O(1)$. This explains the PTAS [24] for such induced subgraph problems.

We can show that almost every known problem in PTAS can be placed in our framework in a like manner. There are three notable exceptions: bin packing [96, 56], some "dense" NP-hard problems [5], and planar-graph TSP [48]. We present one possible explanation for this incompleteness in the first two cases.

Consider first the classical bin packing problem. Vega and Lueker [96] show that there is a bin packing algorithm that, for any $\epsilon > 0$, achieves a packing into at most $(1+\epsilon)\text{OPT} + 1$ bins in polynomial time; in fact, Karmakar and Karp [56] give an algorithm that achieves a bound of $\text{OPT} + o(\text{OPT})$, implying a *fully polynomial* approximation scheme (FPTAS). While this may appear to be a PTAS, observe that via a reduction from the partition problem [41] it is easy to show that bin packing cannot be approximated within a ratio better than 1.5 unless P = NP. The apparent contradiction is resolved by noting the crucial role played by the additive error of 1. Their scheme is in fact an *asymptotic* PTAS [83]. The asymptotic approximation ratio of an algorithm A is defined as

$$R_A^{\infty} = \inf \left\{ \begin{array}{l} r \geq 1 : \text{for some} N \in Z^+,\ R_A(I) \leq r \text{ for all} \\ I \text{ such that } OPT(I) \geq N \end{array} \right\}$$

where R_A is the performance ratio for an instance I [41]. The point is that while bin packing has an asymptotic PTAS, it does not belong to PTAS and hence is ineligible for consideration in our framework. In fact, Karmakar and Karp [56] give an algorithm that achieves a bound of $\text{OPT} + o(\text{OPT})$, implying a *fully polynomial* approximation scheme (FPTAS). But bin packing is *strongly* NP-complete and hence cannot have an FPTAS [41]. This apparent contradiction can also be resolved by noting that the additive error could be relatively large unless one assumes that the optimum value is some suitably fast growing function of the input size, i.e., the Karmakar and Karp result is actually an *asymptotic* FPTAS [83].

Consider now the recent results due to Arora, Karger, and Karpinski [5] where they provide a unified scheme to obtain PTAS for dense instances of certain NP-hard optimization problems where, informally speaking, "dense" means that the optimal solution value is within a constant factor of the

maximum possible value of a solution. For instance, they show that MAX CUT has a PTAS if the input instance contains $\Omega(n^2)$ edges. Specifically, for any $\epsilon > 0$, they obtain in polynomial time a solution of size OPT $- \epsilon n^2$. Consequently, restricting to problem instances such that the optimum value is $\Omega(n^2)$, such a result translates into a PTAS scheme for MAX CUT.

In other words, we contend that these approximation schemes are possible because of the re-interpretation of the additive error term as a relative error term when the optimum value is appropriately constrained. These observations should however point to the delicacy of the arguments that may be required to make further progress on the questions raised in our work.

However, there remain a couple of notable exceptions. Arora et al [5] provide a PTAS for dense graph bisection even when the optimum value is small; clearly, this remains unexplained by the preceding discussion. Similarly, we are also unable to capture in any obvious manner the recent PTAS results for planar-graph TSP problem [48] and the Euclidean TSP problem [3]. It is possible that this is only a technical difficulty, since, for instance, the PTAS result of Grigni *et al* uses techniques that bear some similarity to the ones used by Baker and in our work. In any case, our contention here is not that this framework exhaustively captures all problems in PTAS, but merely that it provides a unified framework for explaining known membership in PTAS, and for obtaining new results of this type. Such a unification raises the possibility that a generalization of this framework, or some variant thereof, may yield a syntactic characterization under closure for PTAS.

3.2.2 Possibility of Extension of Classes

In this section, we discuss the potential for extending our framework to more general classes that retain membership in PTAS. There are two natural directions for generalizing the syntactic classes wherein our results apply. One possibility is to relax some constraints in the logical definition of the optimization classes. So, for instance, it may seem natural to relax the restriction of $n^{O(1)}$ minterms in the class MPSAT by considering CNF representations of the formulas (or oracle representations, for that matter). But this definition is immediately general enough, even with an input restriction to *trees*, to capture 3-SAT as a special case, and hence would not permit a PTAS. Similarly, one could considering relaxing the constraint enforcing strictly negative minterms in the class TMAX

and strictly positive minterms in the class TMIN. But once again, relaxing this immediately rules out the possibility of PTAS in a strong sense since then we can encode PLANAR 3-SAT. However, our results for MPSAT, TMAX, and TMIN generalize to non-boolean domains, i.e., they hold even when the variables take on a polynomially-bounded number of distinct values. Furthermore, they hold even when the notion of constraints is relaxed to functions over non-boolean domains modulo some restrictions. Our techniques can also be used to obtain PTAS for certain classes of integer linear programs. We omit a formal exposition of these extensions.

The other direction for extending our results is by relaxing the planarity restriction. In the final version, we will describe an extension our results to δ-near-planar graphs provided that the optimum value is proportional to the number of vertices in the incidence graph. (This is similar to the results of Hunt et al [54] for specific MAX SNP problems which clearly satisfy the restriction on the optimum value.) The key idea is to start with a near-planar embedding and replace each edge crossing with a Lichtenstein gadget [77] that introduces auxiliary formulas. In other words, problems on such graphs have an approximation-preserving reduction to planar graphs. An interesting direction for future work is: are there more general classes of graphs to which our results can be extended? One possibility is to consider families of graphs with excluded minors.

3.2.3 Overview of Proof Techniques

All known PTAS results for planar graph problems employ the following paradigm: find a separator of the planar graph G such that the problem is either optimally or approximately solvable on the components resulting from the deletion of the separator; show that the separator contributes only a small fraction of the optimal solution, obtain a solution by combining the solutions on the components. Some earlier results [80, 24] were based on finding small separators that yielded small components. Baker [13] observed that many NP-hard optimization problems can be solved optimally in polynomial time on $O(1)$-*outerplanar graphs*, and that separators can be found that yield $O(1)$-outerplanar components containing sufficient information about the optimal solution to permit a PTAS.

Our approach extends Baker's paradigm by computing either an exact solution or a PTAS for generic syntactic problems on outerplanar graphs, and then extending this to planar graphs.

While Baker's paradigm employs dynamic programming on $O(1)$-outerplanar explicitly based on a planar embedding, our approach, on the other hand, is to use a more structured and uniform representation of outerplanar graphs in terms of a *tree-decomposition* [21]. In our abstraction, dynamic programming combined with rounding techniques can be used to solve optimally or approximately the proposed syntactic problems when the incidence graphs are restricted to being $O(1)$-outerplanar. The syntactic structure of the problem is then used to extend the solution to the original planar graph by either using a second-level of dynamic programming, or a pigeon-hole argument similar to Baker's.

3.3 Outerplanar Graphs and Tree Decompositions

Our constructions of PTAS for the planar versions of these complexity classes depends on the strengthening of planar separator theorem by Bodlaender [21] for $O(1)$-outerplanar graphs.

Definition 38 (p-Outerplanar Graph) *A* 1-outerplanar *graph, or simply an* outerplanar *graph, is a planar graph that has an embedding on the plane with all vertices appearing on the outer face; a p-outerplanar graph is a planar graph that has an embedding on the plane in which deleting all vertices on the outer face yields a $(p-1)$-outerplanar graph.*

Observe that the outer face of a 1-outerplanar graph need not be a simple cycle; for example, a tree is outerplanar and two cycles meeting at a single vertex is also outerplanar. We describe a useful structural property of outerplanar graphs based on the notion of treewidth due to Robertson and Seymour [93].

Definition 39 (Tree Decomposition) *A tree decomposition of a graph $G = (V, E)$ is a 2-tuple $\langle \{X(i) \mid i \in I\}, T = (I, F) \rangle$ such that:*

- *for each $v \in V$, there exists an $i \in I$ with $v \in X(i)$;*
- *for all $(v, w) \in E$, there exists an $i \in I$ with $v, w \in X(i)$;*
- *and, for all $v \in V$, the set $\{i \in I \mid v \in X(i)\}$ induces a subtree.*

Definition 40 (Width and Treewidth) *The* width *of a tree-decomposition* $\langle \{X(i) \mid i \in I\}, T = (I, F)\rangle$ *of a graph* G *is* $\max\{|X(i)| \mid i \in I\} - 1$. *The* treewidth *of a graph is the minimum width over all tree-decompositions*.

For any node Y in the decomposition tree, define the graph $G[Y]$ as the subgraph of the graph G induced by the set of vertices $V[Y]$ consisting of the vertices stored at all the nodes in the subtree rooted at Y. Thus, for the root X, we have $V[X] = V$ and $G[X] = G$.

Proposition 2 *A tree decomposition of a graph* G *may be viewed as a separator tree. In particular, at the root* X *with children* L *and* R*,*

1. $V[L] \cap V[R] \subseteq X$*, and*
2. *there are no edges from* $V[L] \setminus X$ *to* $V[R] \setminus X$ *in* G.

Further, the same property holds recursively at each node in the tree.

Lemma 6 ([21]) *A* p*-outerplanar graph has treewidth at most* $3p - 1$.

Definition 41 (Nice Tree Decomposition) *A tree decomposition* $\langle \{X(i) \mid i \in I\}, T = (I, F)\rangle$ *is* nice *if one can choose a root* r *in the tree such that:*

- T *is a binary tree,*
- *if* i *is a leaf, then* $|X(i)| = 1$ (START node)*;*
- *if* i *has two children* j *and* k*, then* $X(i) = X(j) = X(k)$ (JOIN node)*;*
- *and, if* i *has one child* j*, then either*
  - *there exists a vertex* v *with* $X(i) = X(j) - \{v\}$ (FORGET node)*, or*
  - *there exists a vertex* v *with* $X(i) = X(j) \cup \{v\}$ (INTRODUCE node).

We use the following known results: there is always a nice tree decomposition of optimal width [21]; and, there is a linear time algorithm to a construct tree-decomposition of graphs with $O(1)$ treewidth [22].

3.4 Planar MPSAT

Let w_v denote the weight associated with the variable v and p_f denote the profit associated with the formula f; furthermore, let W denote the weight threshold and let P denote the sum of all the profits.

We begin by sketching a *pseudo-polynomial* time algorithm for the case of $O(1)$-outerplanar graphs. The first step is to compute a nice tree decomposition T for a given $O(1)$-outerplanar graph G; recall that it has treewidth $O(1)$. We need some definitions.

Definition 42 (Tuple) *A tuple over a set of variables x_1, ..., x_p and a set of formulas $f_1, \ldots, f_q$ is a vector of the form*

$$\langle v_1, \ldots, v_p, m_1, \ldots, m_q \rangle,$$

where each $v_i \in \{0, 1\}$ and each m_i is either a minterm that occurs in formula f_i or is NULL.

Definition 43 (Consistent Tuple) *A tuple*

$$\vec{t} = \langle v_1, \ldots, v_p, m_1, \ldots, m_q \rangle$$

over a set of variables $x_1, \ldots, x_p$ and formulas $f_1, \ldots, f_q$ is called a consistent tuple *if whenever a variable x_i occurs in a minterm m_j in t, then $v_i = 1$ if x_i occurs as a positive literal in m_j, and $v_i = 0$ otherwise.*

In our algorithms, we maintain the following 2-dimensional structure at each node in the tree T.

Definition 44 (VLIST) *For a node X in the decomposition tree,* VLIST$[\vec{t}, p]$ *denotes the minimum weight needed to achieve a profit of p in the* MPSAT *subproblem represented by the subgraph $G[X]$ when the variables and formulas at node X are set in accordance with the tuple $\vec{t}$.*

Definition 45 (EXT) *Given a tuple $\vec{t}$ and a variable or formula x, the extension set* EXT$(\vec{t}, x)$ *consists of all tuples that are extensions of $\vec{t}$ with each possible value that can be assigned to x.*

Definition 46 (PROJ) *Given a tuple $\vec{t}$ and a variable or formula x, the projection* $\text{PROJ}(\vec{t}, x)$ *is the tuple obtained from $\vec{t}$ by dropping the component corresponding to x.*

Definition 47 (WT and PT) *Given a tuple $\vec{t}$,* $\text{WT}(\vec{t})$ *denotes the weight of variables set to true in $\vec{t}$, and* $\text{PT}(\vec{t})$ *denotes the total profit associated with formulas appearing in $\vec{t}$ with non-*NULL *minterms.*

We now show that the VLIST array can be efficiently computed at the root of a nice treewidth decomposition tree using a bottom-up approach. The entries in the VLIST array at each node are initialized to $+\infty$. At the leaves, the array can be trivially computed in polynomial time. To compute the array at an intermediate node such that it has already been computed for its children, we analyze three cases:

- at a JOIN node X with children L and R, we compute

$$\text{VLIST}_X[\vec{t}, p] = \min_{0 \leq i \leq p} \left\{ \text{VLIST}_L[\vec{t}, i + \text{PT}(\vec{t})] + \text{VLIST}_R[\vec{t}, p - i] - \text{WT}(\vec{t}) \right\},$$

- at a FORGET node X with a child L such that $X = L \setminus \{x\}$, we compute

$$\text{VLIST}_X[\vec{t}, p] = \min_{\vec{t'} \in \text{EXT}(\vec{t}, x)} \text{VLIST}_L[\vec{t'}, p],$$

- finally, at an INTRODUCE node X with a child L such that $X = L \cup \{x\}$, we consider two subcases.

  If x corresponds to a variable v then

$$\text{VLIST}_X[\vec{t}, p] = \begin{cases} \text{VLIST}_L[\text{PROJ}(\vec{t}, x), p] & \text{if } \vec{t} \text{ is consistent and } v \text{ is false in } \vec{t} \\ \text{VLIST}_L[\text{PROJ}(\vec{t}, x), p] + w_v & \text{if } \vec{t} \text{ is consistent and } v \text{ is true in } \vec{t} \\ +\infty & \text{otherwise} \end{cases}$$

  On the other hand, if x corresponds to a formula f then

$$\text{VLIST}_X[\vec{t}, p] = \begin{cases} \text{VLIST}_L[\text{PROJ}(\vec{t}, x), p - p_f] \\ \quad \text{if } \vec{t} \text{ is consistent and contains a non-NULL minterm for } f \\ \text{VLIST}_L[\text{PROJ}(\vec{t}, x), p] \\ \quad \text{if } \vec{t} \text{ is consistent and contains a NULL minterm for } f \\ +\infty \qquad\qquad \text{otherwise} \end{cases}$$

In all cases, the computation of VLIST takes polynomial time, and we perform this computation exactly once at each node. Hence, the VLIST computation at the root R_T of T takes only polynomial time. Finally, observe that by definition of VLIST, the largest value occurring in VLIST(R_T) is the desired optimal solution. The proof of correctness of this computation, as well as a bound of $O(n^{O(1)}P^2)$ on the running time, is omitted. By using a standard technique of rounding the lower order bits, we can obtain a PTAS from this pseudo-polynomial time algorithm. The details are omitted; see for example [86]. We obtain the following results.

Lemma 7 *The class* $O(1)$-OUTERPLANAR MPSAT *is contained in* PTAS.

Corollary 1 $O(1)$-OUTERPLANAR MPSAT *is in* P *when the formulas have uniform profits*.

Thus satisfiability problems such as MAX 3-SAT become polynomial-time solvable on this subclass of planar graphs. Consider now the following theorem.

Theorem 20 *The class* PLANAR MPSAT *is in* PTAS.

Proof: This is established by combining Baker-style arguments with a dynamic programming scheme. Consider the input graph G and assume that it is t-outerplanar for $t \leq n + m$, where n is the number of variables and m is the number of formulas in the collection $\mathcal{C}$. Divide the vertices into t levels, called $L_1, \ldots, L_t$, such that L_t corresponds to the outer face and each level L_i is the outer face obtained by removing the levels $L_t, \ldots, L_{i+1}$.

Fix any optimal truth assignment, and let s_i denote the *weight of the clauses satisfied* in level L_i by this assignment. Partition the levels $L_1, \ldots, L_t$ into $p + 1$ groups, $S_0, \ldots, S_p$, where group S_r

is the union of the levels L_i whose index i is congruent to $3r$, $3r+1$, or $3r+2$ modulo p', where $p' = 3(p+1)$. By pigeon-holing, there exists a group S_j such that

$$\sum_{L_i \in S_j} s_i \leq \frac{\text{OPT}}{p+1}.$$

This special group S_j may be identified by trying all possible choices of j, and picking the one which yields the best solution. Having fixed the choice of S_j, delete all vertices in the levels whose index is congruent to $3j+1$ modulo p', thereby separating the graph into a collection of disjoint $(p'-1)$-outerplanar graphs, say $G_1, G_2, \ldots, G_t$, such that the total optimal value on this collection is at least $(1-(3/p)) \times \text{OPT}$.

For a given weight threshold W, we know how to obtain a pseudo-polynomial time algorithm for any G_i. But we need to find an "optimal partitioning" of the total weight W into sub-thresholds $W_1, W_2, \ldots, W_t$, where W_i is the weight threshold allocated to G_i. To this end, we begin by performing the VLIST computation for each G_i. Once this two-dimensional array is computed, we can extract from it a one-dimensional profit array $A_i[0..P]$ for each G_i, such that $A_i[p]$ denotes the minimum weight needed to achieve profit p in the instance encoded by G_i.

Next, we create a complete binary tree whose leaves correspond to $G_1, G_2, \ldots, G_t$ and an intermediate node corresponds to the union of the graphs represented by its children. Thus, the root corresponds to $\bigcup_{i=1}^{t} G_i$. This can now be used to iteratively compute the profit arrays associated with any non-leaf vertex. Consider a non-leaf vertex v such that we already have the profit arrays for its two children u and w. The array at v can be computed as follows:

$$\forall i \in [0..P]\, \forall j \in [0..i]\; A^v[i] \leftarrow \min\{A^v[i], A^u[j] + A^w[i-j]\}.$$

We omit the formal proof of correctness of this computation and of the total running time bound as a polynomial in n and P. Thus, we have a pseudo-polynomial time approximation scheme for the constrained optimization problem. The final step is to use the standard lower-order bit rounding technique to obtain a PTAS. We omit the details (refer to Papadimitriou [86]). ■

3.5 Planar TMAX

We sketch the PTAS for PLANAR TMAX. As before, w_v is the weight of variable v; since all constraints must be satisfied, we do not associate any profits with the formulas. We begin with the following lemma.

Lemma 8 *The class* $O(1)$-OUTERPLANAR TMAX *is contained in* P.

In the proof sketch for this lemma, we will define VLIST as a one-dimensional array.

Definition 48 (VLIST) *For a node* X *in the decomposition tree,* VLIST$[\vec{t}]$ *denotes the maximum weight achievable in the* TMAX *subproblem represented by the subgraph* $G[X]$ *when the variables and formulas at node* X *are set in accordance with the tuple* $\vec{t}$.

The entries in the VLIST array at each node are initialized to $-\infty$. In the rest of this section, we will assume that a tuple *does not* contain a NULL term for any formula, i.e., we consider only those tuples that satisfy all their formulas.

We briefly sketch the computation of the VLIST array at the root of a nice treewidth decomposition tree using a bottom-up approach. At the leaves, the array can be trivially computed in polynomial time. To compute the array at an intermediate node where it has already been computed for its children, we consider three cases:

- at a JOIN node X with children L and R, we compute

$$\text{VLIST}_X[\vec{t}] = \text{VLIST}_L[\vec{t}] - \text{WT}(\vec{t}),$$

- at a FORGET node X with a child L such that $X = L \setminus \{x\}$, we compute

$$\text{VLIST}_X[\vec{t}] = \max_{\vec{t'} \in \text{EXT}(\vec{t},x)} \text{VLIST}_L[\vec{t'}],$$

- finally, at an INTRODUCE node X with a child L such that $X = L \cup \{x\}$, we consider two subcases.

If x corresponds to a variable v then

$$\text{VLIST}_X[\vec{t}] = \begin{cases} \text{VLIST}_L[\text{PROJ}(\vec{t}, x)] & \text{if } \vec{t} \text{ is consistent and } v \text{ is false in } \vec{t} \\ \text{VLIST}_L[\text{PROJ}(\vec{t}, x)] + w_v & \text{if } \vec{t} \text{ is consistent and } v \text{ is true in } \vec{t} \\ -\infty & \text{otherwise} \end{cases}$$

On the other hand, if x corresponds to a formula f then

$$\text{VLIST}_X[\vec{t}] = \begin{cases} \text{VLIST}_L[\text{PROJ}(\vec{t}, x)] & \text{if } \vec{t} \text{ is consistent} \\ -\infty & \text{otherwise} \end{cases}$$

We omit the proof of correctness of this computation and the running time bound of $O(n^{O(1)})$. A careful look at this computation reveals that we did not really use the negativity constraint on the minterms and thus we can conclude a stronger result.

Corollary 2 *The class* $O(1)$ OUTERPLANAR TMAX *without negativity constraints on the minterms is in* P.

We now sketch the extension to planar graphs leading to the following theorem.

Theorem 21 *The class* PLANAR TMAX *is contained in* PTAS.

Proof Sketch : The idea is to remove a collection of layers whose deletion leaves an instance with optimal value at least $(1 - O(1/p)) \times$ OPT and which is a collection of p-outerplanar graphs, say $G_1, \ldots, G_t$. Fix any optimal assignment and let s_i denote the *weight of the variables* set to true in level L_i. Partition the levels $L_1, \ldots, L_t$ into $p+1$ groups, $S_0, \ldots, S_p$, where group S_r is the union of the levels L_i whose index i is congruent to $2r$ or $2r+1$ modulo p', where $p' = 2(p+1)$. By pigeon-holing, there exists a group S_j such that

$$\sum_{L_i \in S_j} s_i \leq \frac{\text{OPT}}{p+1}.$$

This special group S_j may be identified by trying all possible choices of j, and picking the one that yields the best solution. Having fixed the choice of S_j, delete all *variable vertices* in the levels

represented in S_j. Compute the optimal solutions in each of the resulting $(2p+1)$-outerplanar graphs (recall that the formula vertices do not have any edges between them). Since these graphs do not share any vertices, we can compute their optimal solutions independently and combine these solutions without any conflict. The combined solution is finally extended by setting all the variables in the special group S_j to FALSE and the remaining variables according to the value assigned by the above solutions. It is easy to see that due to the *negative minterm property*, this scheme of composing the optimal solution results in a solution satisfying all the constraints. And finally, the weight of the solution is at least $(1 - 1/(p+1))$OPT. □

3.6 Planar TMIN

An algorithm similar to the one in Section 3.5 can be used to establish that $O(1)$-OUTERPLANAR TMIN is in P and also that $O(1)$-OUTERPLANAR TMIN without positivity constraints on the minterms lies in P. However, the scheme needed to create a PTAS for planar graph is slightly different; it is a variant of the scheme used by Baker [13] for the minimum vertex cover problem. The idea is to consider groups of p consecutive layers such that the last two layers of any group are identical to the first two layers of the next group. If we compute solutions for the p-outerplanar graphs defined by these layers and combine them by setting a variable to TRUE whenever it is set to TRUE in any solution and FALSE otherwise, it is easy to see that such a solution must indeed satisfy all the constraints because each constraint vertex along with its constituent variable vertices must be completely contained in one of the groups. Now, if we fix any optimal assignment, there exists a way of defining a shift of these layers such that the total weight of OPT in the overlapping layers is only $O(\text{OPT}/p)$. Hence, we have a PTAS.

3.7 $O(1)$-Outerplanar MINMAX

In this section we generalize our syntactic classes to capture problems over multi-valued domains where the objective is to have a "balanced solution" satisfying a given set of constraints. However, our results hold only for $O(1)$-outerplanar graphs – a class general enough to capture problems such

as multiprocessor scheduling as a special case.

Definition 49 (k-ary term)

A k-ary term *over a set of* n *variables is a constraint of the form* $\langle x_{i_1} = s_1, \ldots, x_{i_p} = s_p \rangle$ *where* $s_i \in [0..k]$ *and* k *is a constant. An assignment* satisfies *the term if and only if its projection on the variables in the term coincides with the specified values.*

Definition 50 (k-ary formula)

A k-ary formula *over a set of* n *variables is a set of* k*-ary terms such that the formula is* satisfied *by an assignment if and only if it satisfies at least one term in the formula.*

Definition 51 (Variable Class)

Given an assignment s*, a* variable class i *denotes the set of variables with value* i *in assignment* s*.*

We define a new class as follows.

Definition 52 (MINMAX)

The class MINMAX *consists of* NPO *problems expressible as: given a collection* $\mathcal{C}$ *of* FOFs *over* n *variables such that each formula* $\Phi \in \mathcal{C}$ *has* $O(n^{O(1)})$ k*-ary terms, find an assignment which minimizes the maximum-weight variable class.*

We briefly sketch the idea used to obtain a PTAS for $O(1)$-OOUTERPLANAR MINMAX. As we did before, for $O(1)$-outerplanar graphs we perform a bottom-up VLIST computation on a nice tree-decomposition. The VLIST array is a 2-dimensional array – the array entry VLIST$[\vec{t}, \langle w_0, w_1, \ldots, w_k \rangle]$ stores a YES/NO answer to the question of whether there is a feasible solution which respects the constraint set and the variable assignment specified by the tuple, and partitions the variable weights into $\langle w_0, w_1, \ldots, w_k \rangle$; where $0 \leq w_i \leq W$ and $W = \sum_{i=1}^{n} \text{WT}(x_i)$. This computation can be performed in time $O((nW)^{O(1)})$, since k is a constant. At the root, we simply choose the optimal feasible solution over all tuples. Hence, we have a pseudo-polynomial time algorithm. But observe that OPT $\geq W/(k+1)$ and therefore, once again using the lower order bit rounding techniques, we can convert this into a PTAS result.

Chapter 4

The Approximability of Constraint Maximization Problems

4.1 Introduction *

Thus far, our focus has been to identify the structure that underlies all problems in a given approximation class. But if we consider an approximation class such as APX, we observe that not all problems in this class are equally hard. In particular, APX contains problems in P which we can solve optimally and problems such as MAX 3-SAT which can't be approximated beyond a certain constant factor. At this juncture, we would like to ask if we can characterize which optimization problems are easy and which ones hard; and to ask if there are any characteristic features, specifc only to hard problems, that can be isolated. Of course, no complete characterization is possible: Rice's theorem allows one to disguise an optimization problem so cleverly that it would be undecidable to determine if a given problem is NP-hard or polynomial time solvable. Even if the problem is presented in its simplest form — it may be the case that the answer need not be "easy" or "NP-hard" — this is established by a theorem of Ladner [75].

In the presence of such barriers one is forced to weaken one's goals and focus one's attention onto a restricted subclass of NP Optimization problems in the hope that some features of hard problems can be isolated from this subclass. Our choice of the appropriate restriction comes from the work of Schaefer [95] who carried out an analogous investigation in the case of decision problems. Schaefer considered a restriction of NP that he called "satisfiability problems" and successfully characterized every problem in this (infinite) class as being easy (polynomial time decidable) or hard (NP-hard). He refers to this as a "dichotomy theorem", since it partitions the class of problems studied into two polynomial time equivalent classes. A further study along these lines — looking for other dichotomic classes within NP — was carried out more recently by Feder and Vardi [35].

Schaefer's work is central to our technical results. We sketch his result in some detail here; a formal presentation may be found in Section 4.2. Schaefer considers what he calls "generalized satisfiability problems". A typical instance to a typical problem consists of n boolean variables with m constraints on them. The constraints are of finite arity and come from a finite collection of templates, $\mathcal{F}$, which define the satisfaction problem, which we will call SAT($\mathcal{F}$). For every such collection $\mathcal{F}$ he investigates the complexity of determining if the given instance is satisfiable, i.e.,

* This chapter is based on joint work with Madhu Sudan [70].

is there an assignment to the n variables satisfying all m constraints. He enumerates six classes of constraint sets for which the satisfiability problem SAT($\mathcal{F}$) is decidable in polynomial time. These classes include 2CNF formulae, satisfiability of linear equations modulo two and satisfiability of Horn clauses (and some other simple classes). He shows that whenever a family contains members not strictly in the six cases he considers, then the satisfibility problem is NP hard!

We generalize Schaefer's characterization to optimization problems. We define a class of optimization problems called MAXCSP which is a subclass of MAX SNP and forms what we feel is a combinatorial core of MAX SNP. As in Schaefer's case, a problem of this class is described by a constraint set $\mathcal{F}$. An instance of this problem is given by n boolean variables and m constraints, and the objective is to find an assignment satisfying the maximum number of constraints. It turns out that MAXCSP contains most of the well-known MAX SNP problems, such as MAX CUT, MAX 2SAT and MAX 3SAT. We show that every problem in MAXCSP is either solvable exactly in polynomial time or MAX SNP-hard. We are able to characterize exactly the structure of the easy problems. Our characterization of easy problems in this class shows that s-t MIN CUT problem is essentially the only easy problem in this class. Barring this problem, every other problem that we study in this class, is complete for the class (and hence for MAX SNP). In particular, there are no problems which possess a PTAS (without possessing an exact optimization algorithm). Our result provides a new insight into questions such as : Why do all the known MAX SNP problems that are NP-hard, also turn out to be MAX SNP-hard? What is the unifying element among these problems?

The remainder of this chapter is organized as follows. Section 4.2 introduces the definitions and notation needed to describe our work. A formal statement and an overview of our main result is given in Section 4.3. Sections 4.4 and 4.5 are devoted to proving the main result, that is, the classification theorem for MAXCSP. In Section 4.6, we show that the MAXCSP version of the problems characterized as hard by Schaefer, are MAX SNP-hard at the *gap location 1*. Finally, in Section 4.7, we show how to eliminate the replication assumption from Schaefer's proof.

4.2 Preliminaries

We start with the definition of a constraint.

Definition 53 [Constraint] *A* constraint *is a function $f : \{0,1\}^k \to \{0,1\}$. We say f is* satisfied *by an assignment $s \in \{0,1\}^k$ if $f(s) = 1$. We refer to k as the* arity *of the constraint f. A constraint with no satisfying assignments is called* unsatisfiable.

Often we apply a constraint f of arity k to a subset of k variables from a larger set. In such cases we think of f as a constraint on the larger set of variables.

Definition 54 [Constraint Application] *Given n boolean variables $X_1, \ldots, X_n$ and a constraint f of arity k, and indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$, the pair $(f, (i_1, \ldots, i_k))$ is referred to as an* application *of the constraint f to $X_1, \ldots, X_n$. An assignment $X_i = s_i$ for $i \in \{1, \ldots, n\}$ and $s_i \in \{0,1\}$* satisfies *the application if $f(s_{i_1}, \ldots, s_{i_k}) = 1$.*

Definition 55 [Constraint Set] *A* constraint set *$\mathcal{F} = \{f_1, \ldots, f_l\}$ is a finite collection of constraints.*

Definition 56 [Constraint Satisfaction Problem (MAXCSP($\mathcal{F}$))] *Given a constraint set $\mathcal{F}$, the* constraint satisfaction problem MAXCSP($\mathcal{F}$) *is defined as follows :*

INPUT : *A collection of m constraint applications of the form $\{(f_j, (i_1(j), \ldots, i_{k_j(j)}))\}_{j=1}^m$, on boolean variables $X_1, X_2, ..., X_n$ where $f_j \in \mathcal{F}$ and k_j is the arity of f_j.*

OBJECTIVE : *Find a boolean assignment to X_i's so as to maximize the number of applications of the constraints $f_1, \ldots, f_m$ that are satisfied by the assignment.*

Notice that the above definition gives a new optimization problem for every family $\mathcal{F}$. Schaefer's class of decision problems SAT($\mathcal{F}$) can be described in terms of the above as: The members of SAT($\mathcal{F}$) are all the instances of MAXCSP($\mathcal{F}$) whose optimum equals the number of applied constraints (i.e., all constraints are satisfiable). Schaefer's dichotomy theorem essentially shows that the only families $\mathcal{F}$ for which SAT($\mathcal{F}$) is in P, is if all constraints in $\mathcal{F}$ are either satisfied by the all zeroes assignment, the all ones assignment or all constraints are linear constraints over GF(2) or all constraints are Horn clauses. A formal statement is as below; we need some additional definitions.

Definition 57 [Weakly Positive and Weakly Negative Functions] *A function is called* weakly positive *(*weakly negative*) if it may be expressed as a* CNF *formula such that each clause has at most one negated (unnegated) variable.*

Definition 58 [Affine] *A function is said to be* affine *if it may be expressed as a system of linear equations of the form* $\sum_{i=1}^{k} x_i = 0$ *and* $\sum_{i=1}^{k} x_i = 1$ *(i.e. it evaluates to one iff the input variables satisfy the given equation system); the addition operation being modulo 2.*

Theorem 22 ([95]) *Let* $\mathcal{F}$ *be a finite set of boolean functions. Then SAT*$(\mathcal{F})$ *is always either in* P *or* NP*-hard. Furthermore, it is in* P *if and only if one of the following conditions is true:*

1. *Every* $f \in \mathcal{F}$ *is 0-valid.*
2. *Every* $f \in \mathcal{F}$ *is 1-valid.*
3. *Every* $f \in \mathcal{F}$ *is weakly positive.*
4. *Every* $f \in \mathcal{F}$ *is weakly negative.*
5. *Every* $f \in \mathcal{F}$ *is affine.*
6. *Every* $f \in \mathcal{F}$ *is bijunctive (i.e. expressible as a CNF formula with at most 2 literals per clause).*

Our main result asserts that a dichotomy holds for MAXCSP$(\mathcal{F})$ as well. However the characterization of the easy functions and the hard ones is quite different. (In particular, many more constraint sets $\mathcal{F}$ are hard now.) In order to describe our result fully we need some more definitions.

Definition 59 *Given a constraint set* $\mathcal{F}$*, the constraint set* $\mathcal{F}'$ *is the set of constraints* f *in* $\mathcal{F}$ *which are satisfiable.*

It is easy to see that for any constraint set $\mathcal{F}$, an instance of MAXCSP$(\mathcal{F})$ can be mapped to an instance of MAXCSP$(\mathcal{F}')$, such that the objective function value is preserved on each input assignment. Hence our characterizations will essentially be characterizations of MAXCSP$(\mathcal{F}')$.

Definition 60 [i-valid function] *For* $i \in \{0, 1\}$*, a function* f *of arity* k *is called* i-valid *if it is satisfied by the assignment* i^k*.*

Definition 61 [Minterm] *Given a function f on variables $x_1, \ldots, x_k$, a conjunction of literals drawn from these variables, say $x_{i_1}, \ldots, x_{i_l}, \overline{x_{j_1}}, \ldots, \overline{x_{j_m}}$, is called a minterm of f if it satisfies the following properties:*

1. *Any assignment $s = s_1, \ldots, s_k$, which satisfies $s_{i_1} = \cdots = s_{i_l} = 1$ and $s_{j_1} = \cdots = s_{j_m} = 0$ satisfies f.*
2. *The collection is minimal with respect to property (1).*

Definition 62 [Positive and Negative Minterms] *A minterm of f which consists only of unnegated variables is called a positive minterm. A minterm which consists only of negated variables is called a negative minterm.*

Definition 63 [2-Monotone Function] *A function f is called* 2-monotone *if it has at most two minterms such that at most one of them is positive and at most one is negative.*

4.3 Overview of the Main Result

Our main result is as stated below.

Theorem 23 *The problem* MAXCSP($\mathcal{F}$) *is always either in* P *or is* MAX SNP*-hard. Furthermore, it is in* P *if and only if one of the following conditions is true:*

1. *Every $f \in \mathcal{F}'$ is 0-valid.*
2. *Every $f \in \mathcal{F}'$ is 1-valid.*
3. *Every $f \in \mathcal{F}'$ is 2-monotone.*

This theorem follows from Lemmas 9,10,11 and 23. A second result that follows easily as a consequence of Schaefer's result and our notion of approximation preserving reductions is the following:

Theorem 24 *For every constraint set $\mathcal{F}$ either SAT($\mathcal{F}$) is easy to decide, or there exists $\epsilon = \epsilon_{\mathcal{F}} > 0$ such that it is NP-hard to distinguish satisfiable instances of SAT($\mathcal{F}$), from instances where $1 - \epsilon$ fraction of the constraints are not satisfiable.*

Schaefer's result characterizes which function families are easy to decide and which ones are hard.

4.3.1 Discussion

The main feature of Theorem 25 is that the family of constraint sets which lead to hard problems is significantly larger than in Schaefer's case. This is not surprising given that problems such as 2SAT and Linear systems over GF(2) are easy problems for the decision version and the maximization versions are known to be hard, even to approximate [87, 4]. Nevertheless, the set of problems that are shown to be easy is extremely small. MAXCSP($\mathcal{F}$) for $\mathcal{F}$ which is 0-valid or 1-valid is really a trivial problem; leaving only the class of 2-monotone functions as somewhat interesting. But the class of functions with such properties seems to be really small and maximum flow or s-t min cut appear to be the only natural optimization problems with this property. Thus, we feel, that the correct way to interpret Theorem 25 is to think of it as saying that every constraint satisfaction problem is either solvable by a maximum flow computation or it is MAX SNP-hard.

Another interesting feature of the above result is that the dichotomy holds for two different properties — the complexity and the approximability — simultaneously. *Easy* problems are easy to compute *exactly* and the hard ones are *hard* to even *approximate*. The middle regime — problems that are easy to approximate but hard to compute exactly — are ruled out. This may be somewhat surprising initially, but becomes inevitable once the form of approximation preserving reductions we use here becomes clear. Essentially all reductions we use are exactly those that might be used for *exact* optimization problems. In fact the ease with which these reductions apply is the reason why Theorem 24 falls out easily from this work.

The technical aspects of the proof of the dichotomy theorem may be of some independent interest. In order to prove such a theorem, one needs to find succinct characterizations of what makes a function, say 2-monotone, as well as a succinct proof when a function is *not* 2-monotone. We find such a characterization, in Lemma 17, which turns out to be useful in establishing Theorem 25.

One technical nit-picky point that we face in this study is the role of constants and repetitions in MAXCSP. In particular, if $f \in \mathcal{F}$, should $f|_{x_1=0}$ given by $f|_{x_1=0}(x_2, \ldots, x_k) = f(0, x_2, \ldots, x_k)$ also be considered a member of the constraint set? Similarly should $f'(x_1, x_2) =$

$f(x_1, \ldots, x_1, x_2, \ldots, x_2)$ be considered a member of the constraint set. Allowing for these repetitions and constants makes the analysis much easier; however they may change the complexion of the problems significantly. For instance given a set of linear equalities of the form $\sum_i x_i = 0$, it is trivial to find an assignment which satisfies all the equations — namely the all 0's assignment. However once one is allowed to fix some variables to the constant 1, the problem no longer remains easy. In our presentation, we initially assume we can use constants and repetitions to make our analysis simpler. Later we remove the assumptions — and Theorem 25 and Theorem 24 is shown without the use of any constants or repetition. In fact, in the process we remove a minor irritant from Schaefer's proof which actually needed to use repetitions.

4.3.2 Related Work

Theorem 25 was independently discovered by Creignou [26]. Here we clarify the main points of difference between this work and that of [26]. In [26], the "easy" problems are characterized by a graph-theoretic representation (this is possible since the functions involved in the easy side can be expressed as CNF formulas such that each clause is implicative, that is of the form $(x \longrightarrow y) \equiv (\neg x \vee y)$) and the proof uses graph-theoretic ideas. Our result are stated and established via techniques in the more general context of constraint satisfaction. The proof might be adapted to problems over other domains (larger than the boolean one), whereas it seems unlikely that one could extend the proof of [26] in such a way.

Additionally, technical aspects of the proof given here may be of some independent interest. Particularly, the notion of α-implementation defined here gives a clear way to translate all the hardness results shown here into hardness of *approximation* results. Also, the fact that we do not need to use repetition of variables in functions, to obtain hardness results is another technical improvement on previous results, including that of [95].

4.4 Polynomial Time Solvability

From this section onwards we omit the notation $\mathcal{F}'$ and assume we have a constraint $\mathcal{F}$ such that $\mathcal{F} = \mathcal{F}'$.

Lemma 9 *The problem* MAXCSP($\mathcal{F}$) *is in* P *if each* $f_i \in \mathcal{F}$ *is 0-valid.*

Proof: Set each variable to zero; this satisfies all the constraints. ■

Lemma 10 *The problem* MAXCSP($\mathcal{F}$) *is in* P *if each* $f_i \in \mathcal{F}$ *is 1-valid.*

Proof: Set each variable to one; this satisfies all the constraints. ■

Lemma 11 *The problem* MAXCSP($\mathcal{F}$) *is in* P *if each* f_i *is a 2-monotone function.*

Proof: We reduce the problem of finding the maximum number of satisfiable constraints to the problem of finding the minimum number of unsatisfied constraints. This problem, in turn, reduces to the problem of finding s-t min-cut in directed graphs. 2-monotone constraints have the following possible forms : (a) $x_{i_1}x_{i_2}...x_{i_p}$, (b) $\overline{x_{j_1}}\,\overline{x_{j_2}}...\overline{x_{j_q}}$, and (c) $x_{i_1}x_{i_2}...x_{i_p}+\overline{x_{j_1}}\,\overline{x_{j_2}}...\overline{x_{j_q}}$ where $p, q \geq 1$.

Construct a directed graph G with two special nodes F and T and a vertex x_i corresponding to each variable in the input instance. Let ∞ denote an integer larger than the total number of constraints. Now we proceed as follows for each of the above classes of constraints :

- For a constraint C of the form (a), create a new node e_C and add an edge from each x_i to e_C of cost ∞ and a unit cost edge from e_C to T.
- For a constraint C of the form (b), create a new node $\overline{e_C}$ and add an edge of cost ∞ from $\overline{e_C}$ to each x_i and an edge from F to $\overline{e_C}$ of unit cost.
- Finally, for a constraint C of the form (c), we create two nodes e_C and $\overline{e_C}$ and connect e_C to $x_{i_1}, x_{i_2}, \ldots$ and connect $\overline{e_C}$ to $x_{j_1}, x_{j_2}, \ldots$ as described above and replace the unit cost edges from F and to T by a unit cost edge from e_C to $\overline{e_C}$.

Using the correspondence between cuts and assignments which places vertices corresponding to true variables on the T side of the cut, we find that the cost of a minimum cut separating T from F, equals the minimum number of constraints that can be left unsatisfied. ■

The next lemma shows that s-t min-cut problem with polynomially bounded integral weights is in MAXCSP($\mathcal{F}$) for some 2-monotone constraint set $\mathcal{F}$. Since the previous lemma shows how

to solve MAXCSP($\mathcal{F}$) for any 2-monotone constraint set $\mathcal{F}$ by reduction to s-t min-cut problem, it seems that s-t min-cut problem is the hardest (and perhaps the only) interesting problem in MAXCSP($\mathcal{F}$).

Lemma 12 *The s-t min-cut problem with polynomially bounded integral weights is in* MAXCSP($\mathcal{F}$) *for some 2-monotone constraint set $\mathcal{F}$.*

Proof: Let $\mathcal{F}$ be a family of three functions $\{f_1, f_2, f_3\}$ such that $f_1(X) = \bar{X}$, $f_2(X,Y) = X + \bar{Y}$ and $f_3(Y) = Y$.

Now given an instance $G = (V, E)$ to s-t min-cut problem, we construct an instance of MAXCSP($\mathcal{F}$) on variables $X_1, X_2, ..., X_n$ where X_i corresponds to the vertex $x_i \in V$:

- For each edge $e = (s, x)$ with weight w_e, we have w_e copies of the constraint $f_1(X)$.
- For each edge $e = (x, t)$ with weight w_e, we have w_e copies of the constraint $f_3(X)$.
- For each edge $e = (x, y)$ with weight w_e and such that $x, y \notin \{s, t\}$, we have w_e copies of the constraint $f_2(X, Y)$.

Given a solution to this instance of MAXCSP($\mathcal{F}$), we construct an s-t cut by placing the vertices corresponding to the false variables on the s-side of the cut and the remaining on the t-side of the cut. It is easy to verify that an edge e contributes to the cut iff its corresponding constraint is unsatisfied. Hence the optimal MAXCSP($\mathcal{F}$) solution and the optimal s-t min-cut solution coincides. ■

4.5 Proof of MAX SNP-Hardness

In this section we prove that a constraint set which is not entirely 0-valid or entirely 1-valid or entirely 2-monotone gives a MAX SNP-hard problem. The main MAX SNP-hard problem which we reduce to any of these new ones is the MAX CUT problem shown to be MAX SNP hard by Papadimitriou and Yannakakis [87]. Initially we consider the case where we are essentially allowed to repeat variables and set some variables to true or false. This provides a relatively painless proof that if a function is not 2-monotone, then it provides a MAX SNP hard problem. We then use

the availability of functions that are not 0-valid or 1-valid to implement constraints which force variables to be 1 and 0 respectively, as well as to force variables to be equal. This eventually allows us to use the hardness lemma. We first start with some notation.

4.5.1 Notation

Given an assignment s to an underlying set of variables, $Z(s)$ denotes the set of positions corresponding to variables set to zero and $O(s)$ denotes the set of positions corresponding to variables set to one. More formally, given an assignment $s = s_1 s_2 ... s_n$ to $X_1, X_2, ..., X_n$, where $s_i \in \{0, 1\}$, we have $Z(s) = \{i \mid s_i = 0\}$ and $O(s) = \{i \mid s_i = 1\}$. The notation $s[0 \rightarrow *]$ denotes the set of all assignments s' such that $O(s) \subseteq O(s')$ and similarly, $s[1 \rightarrow *]$ denotes the set of all assignments s' such that $Z(s) \subseteq Z(s')$.

Definition 64 [Unary Functions] *The functions $T(X) = X$ and $F(X) = \bar{X}$ are called* unary functions.

Definition 65 [XOR and REP Functions] *The function $f(X, Y) = X \oplus Y$ is called the* XOR *function and its complement function, namely $X = Y$, is called the* REP *function.*

Definition 66 [C-closed Function] *A function f is called* C-closed (or complementation-closed) *if for all assignments s, $f(s) = f(\bar{s})$.*

Definition 67 [v-Consistent Set] *A set V of positions is* v-consistent *for a constraint f iff every assignment with all variables occupying the positions in V set to value v is a satisfying assignment for f.*

4.5.2 α-Implementations and MAX SNP-Hard Functions

We next describe the primary form of a reduction which we use to give the hardness results. As pointed out by Papadimitriou and Yannakakis, in order to get hardness of approximation results, the reductions used need to satisfy certain approximation preserving features. Here we show how to implement a given function f using a family of other functions $\mathcal{F}$, so as to be useful in approximation preserving reductions.

Definition 68 [α-Implementation] *An instance of* MAXCSP($\mathcal{F}$) *over a set of variables* $\vec{X} = \{X_1, X_2, ..., X_p\}$ *and* $\vec{Y} = \{Y_1, Y_2, ..., Y_q\}$ *is called an* α-implementation *of a boolean function* $f(\vec{X})$*, where* α *is a positive integer, iff the following conditions are satisfied:*

(a) *no assignment of values to* $\vec{X}$ *and* $\vec{Y}$ *can satisfy more than* α *constraints,*

(b) *for any assignment of values to* $\vec{X}$ *such that* $f(\vec{X})$ *is true, there exists an assignment of values to* $\vec{Y}$ *such that precisely* α *constraints are satisfied,*

(c) *for any assignment of values to* $\vec{X}$ *such that* $f(\vec{X})$ *is false, no assignment of values to* $\vec{Y}$ *can satisfy more than* $(\alpha - 1)$ *constraints, and finally*

(d) *for any assignment to* $\vec{X}$ *which does not satisfy* f*, there always exists an assignment to* $\vec{Y}$ *such that precisely* $(\alpha - 1)$ *constraints are satisfied.*

We refer to the set $\vec{X}$ *as the* function variables *and the set* $\vec{Y}$ *as the* auxiliary variables.

Thus a function f 1-implements itself. We will say that MAXCSP($\mathcal{F}$) implements function f if it α_f-implements f for some constant α_f. The following lemma shows that the α-implementations of functions compose together. The criteria for "implementation" given above are somewhat more stringent than used normally. While properties (1)-(3) are perhaps seen elsewhere, property (4) is somewhat more strict, but turns out to be critical in composing implementations together.

Lemma 13 [Composition Lemma] *Given two constraint sets* $\mathcal{F}_f$ *and* $\mathcal{F}_g$ *such that* MAXCSP($\mathcal{F}_f$) *can* α_f*-implement a function* f*, and* MAXCSP($\mathcal{F}_g$) *can* α_g*-implement a function* $g \in \mathcal{F}_f$*, then* MAXCSP($\{\mathcal{F}_f \setminus \{g\}) \cup \mathcal{F}_g$) *can* α*-implement the function* f *for some constant* α*.*

Proof: Let β be the number of occurrences of a constraint involving the function g in the MAXCSP($\mathcal{F}_f$) instance α_1-implementing f, then clearly, by replacing each occurrence of g by its α_2-implementation, we obtain a MAXCSP($\{\mathcal{F}_f \setminus \{g\}) \cup \mathcal{F}_f$) instance which α-implements f for $\alpha = \alpha_1 + \beta(\alpha_2 - 1)$. ■

The MAX SNP-hardness of MAX CUT implies that MAXCSP($\{$XOR$\}$) is MAX SNP-hard and hence the below :

Lemma 14 *If* MAXCSP($\mathcal{F}$) *can implement the* XOR *function, then* MAXCSP($\mathcal{F}$) *is* MAX SNP*-hard*.

Lemma 15 MAXCSP($\{f, T, F\}$) *can implement the* XOR *function if* f *is either the function* $X+Y$ *or* $X\bar{Y}$ *or* $\bar{X}+\bar{Y}$.

Proof: If $f = X+Y$, then the instance $\{f(X,Y), f(X,Y), F(X), F(Y)\}$ is a 3-implementation of $X \oplus Y$; if $f = X\bar{Y}$, then the instance $\{f(X,Y), f(Y,X)\}$ is a 1-implementation of $X \oplus Y$; and finally, if $f = \bar{X}+\bar{Y}$, then $\{f(X,Y), f(X,Y), T(X), T(Y)\}$ is a 3-implementation of $X \oplus Y$. ■

Lemma 16 MAXCSP($\{f, T, F\}$) *can implement the* REP *function if* f *is the function* $\bar{X}+Y$.

Proof: The instance $\{f(X,Y), f(X,Y), F(X), T(Y)\}$ is a 3-implementation of the function REP. ■

4.5.3 Characterizing 2-Monotone Functions

In order to prove the hardness of a constraint which is not 2-monotone, we require to identify some characteristics of such constraints. The following gives a characterization, which turns out to be useful.

Lemma 17 [Characterization Lemma] *A function* f *is a 2-monotone function if and only if all the following conditions are satisfied:*

(a) for every satisfying assignment s *of* f*, either* $s[1 \to *]$ *or* $s[0 \to *]$ *is a set of satisfying assignments,*

(b) if V_1 *is 1-consistent and* V_2 *is 1-consistent for* f*, then* $V_1 \cap V_2$ *is 1-consistent, and*

(c) if V_1 *is 0-consistent and* V_2 *is 0-consistent for* f*, then* $V_1 \cap V_2$ *is 0-consistent*.

Proof: We use the fact that a function can be expressed in DNF form as the sum of its minterms. For a 2-monotone function this implies that we can express it as a sum of two terms. Every satisfying assignment must satisfy one of the two terms and this gives Property (a). Properties (b) and (c) are obtained from the fact that the function has at most one positive and one negative minterm.

Conversely, if a function is not 2-monotone, then it either has a minterm which is not monotone positive or negative or it has more than one positive (or negative) minterm. In the former case, the function will violate Property (a), and in the latter one of Properties (b) or (c). ■

Observe that a 2-monotone function is always either 0-valid or 1-valid or both.

4.5.4 MAX SNP-hardness of Non 2-Monotone Functions

We now use the characterization from the previous subsection to show that if one is allowed to "force" constants or "repetition" of variables, then the presence of non-2-monotone constraint gives hard problems. Rather than using the ability to force constants and repetitions as a binding requirement, we use them as additional constraints to be counted as part of the objective function. This is helpful later, when we try to remove the use of these constraints.

Lemma 18 [Hardness Lemma] *If f is not 2-monotone,* MAXCSP($\{f, T, F, \text{REP}\}$) *can implement the function* XOR.

Proof: We prove this by using the Characterization Lemma for 2-monotone functions. Let k denote the arity of f. If f is not 2-monotone, it must violate one of the three conditions (a), (b) and (c) stated in the Characterization Lemma.

Suppose f violates the property (a) above. Then for some satisfying assignment s, there exist two assignments s_0 and s_1 such that $Z(s) \subset Z(s_0)$ and $O(s) \subset O(s_1)$, but $f(s_0) = f(s_1) = 0$. Without loss of generality, we assume that $s = 0^p1^q$, $s_0 = 0^{p+a}1^{q-a}$ and $s_1 = 0^{p-b}1^{q+b}$. Thus we have the following situation :

	$\overbrace{\quad}^{p-a}$	$\overbrace{\quad}^{a}$	$\overbrace{\quad}^{b}$	$\overbrace{\quad}^{q-b}$	$f()$
s	00...0	00...0	11...1	11...1	1
s_0	00...0	00...0	00...0	11...1	0
s_1	00...0	11...1	11...1	11...1	0
s_2	00...0	11...1	00...0	11...1	_

Observe that both a and b are non-zero. We consider the MAXCSP($\{f, T, F, \text{REP}\}$) instance with the following set of constraints on variables $X_1, X_2, ..., X_k$:

- constraints $F(X_i)$ for $1 \leq i \leq (p-a)$,
- constraints REP(X_{p-a+1}, X_{p-a+i}) for $2 \leq i \leq a$,
- constraints REP(X_{p+1}, X_{p+i}) for $2 \leq i \leq b$,
- constraints $T(X_i)$ for $(p+a+b+1) \leq i \leq k$, and
- the constraint $f(X_1, X_2, ..., X_k)$.

It is now easy to verify that for $\alpha = (k-1)$, this instance α-implements the function $X_{p-a+1} \oplus X_{p+1}$ if $f(s_2) = 1$ and $X_{p-a+1}\overline{X_{p+1}}$, otherwise. The claim now follows immediately from Lemma 15.

Next suppose f violates the property (b) above. Then there exists an unsatisfying assignment s such that s sets all variables in $V_1 \cap V_2$ to 1, and at least one variable in each of $V_1 \setminus (V_1 \cap V_2)$ and $V_2 \setminus (V_1 \cap V_2)$ to be false. Consider one such unsatisfying assignment s. Without loss of generality, we have the following situation :

$$s \quad \overbrace{\underbrace{00...0}_{p}\ \underbrace{11...1}_{q}}^{V_1} \overbrace{\underbrace{11...1}_{r}\ \underbrace{11...1}_{s}\ \underbrace{00...0}_{t}}^{V_2} \ \underbrace{00...0}_{u}\underbrace{11...1}_{v}$$

($V_1 \setminus O(s)$ = p, $V_1 \cap V_2$ = r, $V_2 \setminus O(s)$ = t)

We consider the MAXCSP$(\{f, T, F, \text{REP}\})$ instance with the following set of constraints on variables $X_1, X_2, ..., X_k$:

- constraints REP(X_1, X_i) for $2 \leq i \leq p$,
- constraints $T(X_i)$ for $(p+1) \leq i \leq (p+q+r+s)$,
- constraints REP$(X_{p+q+r+s+1}, X_{p+q+r+s+i})$ for $2 \leq i \leq t$,
- constraints $F(X_i)$ for $(p+q+r+s+t+1) \leq i \leq (p+q+r+s+t+u)$,
- constraints $T(X_i)$ for $(p+q+r+s+t+u) \leq i \leq (p+q+r+s+t+u+v)$, and finally

- the constraint $f(X_1, X_2, ..., X_k)$ where $k = (p + q + r + s + t + u + v)$.

It is now easy to verify that for $\alpha = (k - 1)$, this instance α-implements the function $X_1 + X_{p+q+r+s+1}$. Again, the claim now follows immediately from Lemma 15.

Finally, the case in which f violates the property (c) above, can be handled in an analogous manner. ■

4.5.5 Implementing the REP Function

We now start on the goal of removing the use of the unary and replication constraints above. In order to do so we use the fact that we have available to us functions which are not 0-valid and not 1-valid. It turns out that the case in which the same function is not 0-valid and not 1-valid and further has the property that its behavior is closed under complementation (i.e., $f(s) = f(\bar{s})$) is somewhat special. We start by analyzing this case first.

Lemma 19 [Replication Lemma] *Let f be a non-trivial function which is C-closed and is neither 0-valid nor 1-valid. Then an instance of* MAXCSP($\{f\}$) *can implement the* REP *function.*

Proof: Let k denote the arity of f and let k_0 and k_1 respectively denote the maximum number of 0's and 1's in any satisfying assignment for f; clearly $k_0 = k_1$. Now let $S_X = \{X_1, X_2, ..., X_{2k}\}$ and $S_Y = \{Y_1, Y_2, ..., Y_{2k}\}$ be two disjoint sets of $2k$ variables each. We begin by creating an instance $\mathcal{I}$ of MAXCSP($\{f\}$) as follows. For each satisfying assignment s, there are $\binom{2k}{i}\binom{2k}{k-i}$ constraints in $\mathcal{I}$ such that every i-variable subset of S_X appears in place of 0's in S_X and every $(k-i)$ variable subset of S_Y appears in place of 1's in the assignment s, where i denotes the number of 0's in s.

Clearly, any solution which assigns identical values to all variables in S_X and the complementary value to all variables in S_Y, satisfies all the constraints in $\mathcal{I}$. Let Z and O respectively denote the set of variables set to zero and one respectively. We claim that any solution which satisfies all the constraints must satisfy either $Z = S_X$ or $Z = S_Y$.

To see this, assume without loss of generality that $|S_X \cap Z| \geq k$. This implies that $|S_Y \cap O| \geq k$ or else there exists a constraint in $\mathcal{I}$ with all its input variables set to zero and is hence unsatisfied.

This in turn implies that no variable in S_X can take value one; otherwise, there exists a constraint with $k_1 + 1$ of its inputs set to one, and is unsatisfied therefore. Finally, we can now conclude that no variable in S_Y takes value zero; otherwise, there exists a constraint with $k_0 + 1$ of its inputs set to zero and is unsatisfied therefore. Thus, $Z = S_X$. Analogously, we could have started with the assumption that $|S_X \cap O| \geq k$ and established $Z = S_Y$. Hence an assignment satisfies all the constraints in $\mathcal{I}$ iff it satisfies either the condition $Z = S_X$ or the condition $Z = S_Y$.

We now augment the instance $\mathcal{I}$ of MAXCSP($\{f\}$) as follows. Consider a least hamming weight satisfying assignment s for f. Without loss of generality, we assume that $s = 10^p1^q$. Clearly then, $s' = 0^{p+1}1^q$ is not a satisfying assignment. Since f is C-closed, we have the following situation :

		$\overbrace{\quad}^{p}$	$\overbrace{\quad}^{q}$	$f()$
s'	0	00...0	11...1	0
s	1	00...0	11...1	1
$\bar{s}$	0	11...1	00...0	1
$\bar{s'}$	1	11...1	00...0	0

Consider the constraints $f(X, X_1, X_2, ..., X_p, Y_1, Y_2, ..., Y_q)$ and $f(Y, X_1, X_2, ..., X_p, Y_1, Y_2, ..., Y_q)$. If $X = 1$, then to satisfy the constraint $f(X, X_1, X_2, ..., X_p, Y_1, Y_2, ..., Y_q)$, we must have $Z = S_X$. Otherwise, we have $X = 0$ and then to satisfy the constraint $f(X, X_1, X_2, ..., X_p, Y_1, Y_2, ..., Y_q)$ we must have $Z = S_Y$. In either case, the only way we can also satisfy the constraint

$$f_v(Y, X_1, X_2, ..., X_p, Y_1, Y_2, ..., Y_q)$$

is by assigning Y an identical value. Thus these set of constraints α-implements the function $X = Y$ where α is simply the total number of constraints; all constraints can be satisfied iff $X = Y$ and otherwise, there exists an assignment to variables in S_X and S_Y such that precisely $\alpha - 1$ constraints are satisfied.

■

4.5.6 Implementing the Unary Functions

If the function(s) which is (are) not 0-valid and 1-valid is (are) not closed under complementation, then they can be used to get rid of the unary constraints. This is shown in the next lemma.

Lemma 20 [Unary Lemma] *Let f_0 and f_1 be two non-trivial functions, possibly identical, which are not 0-valid and 1-valid respectively. Then if neither f_0 nor f_1 is C-closed, an instance of* MAXCSP($\{f_0, f_1\}$) *can implement both the unary functions $T(.)$ and $F(.)$.*

Proof: We will only sketch the implementation of function $T(.)$; the analysis for the function $F(.)$ is identical. Now suppose neither f_v is C-closed, $v \in \{0, 1\}$. We begin by considering an instance each of MAXCSP($\{f_0\}$) and MAXCSP($\{f_1\}$), say $\mathcal{I}_0$ and $\mathcal{I}_1$ respectively. Both of these instances are constructed in a manner identical to the instance $\mathcal{I}_A$ above. Now we argue that any solution which satisfies all the constraints in $\mathcal{I}_0$ and $\mathcal{I}_1$, must set all variables in S_X to 0 and all variables in S_Y to 1.

So we have two functions f_0 and f_1 such that neither is C-closed. Suppose $|S_X \cap O| \geq k$, then we must have $|S_Y \cap O| \geq k$. To see this, consider a satisfying assignment s such that $f_0(\bar{s}) = 0$; there must exist such an assignment since f_0 is not C-closed. Now if $|S_Y \cap Z| \geq k$, then clearly at least one constraint corresponding to s is unsatisfied - the one in which the positions in $O(s)$ are occupied by the variables in $(S_Y \cap Z)$ and the positions in $Z(s)$ are occupied by the variables in $(S_X \cap O)$. Thus we must have $|S_Y \cap O| \geq k$. But if we have both $|S_X \cap O| \geq k$ and $|S_Y \cap O| \geq k$, then there is at least one unsatisfied constraint in the instance $\mathcal{I}_1$ since f_1 is not 1-valid. Thus this case cannot arise.

So we now consider the case $|S_X \cap Z| \geq k$. Then for constraints in $\mathcal{I}_0$ to be satisfied, we must once again have $|S_Y \cap O| \geq k$; else there is a constraint with all its inputs set to zero and is hence unsatisfied. This can now be used to conclude that $S_Y \cap Z = \phi$ as follows. Consider a satisfying assignment with smallest number of ones - this number is positive since f_0 is not 0-valid. If we consider all the constraints corresponding to this assignment with inputs from S_Y and $S_X \cap Z$ only, it is easy to see that there will be at least one unsatisfied constraint if $S_Y \cap Z \neq \phi$. Hence each variable in S_Y is set to one in this case. Finally, using the constraints on the function f_1 which is not 1-valid, it is easy to conclude that in fact $Z = S_X$.

Now let $s = 10^p1^q$ be a least hamming weight satisfying assignment for f_0; p, q may be zero but s contains at least a single one as f_0 is not 0-valid. Then the constraint

$$f_0(X, X_1, X_2, ..., X_p, Y_1, Y_2, ..., Y_q)$$

can be satisfied iff $X = 1$. Thus all the constraints in $\mathcal{I}_0$ and $\mathcal{I}_1$ are satisfied along with above constraint iff $X = 1$ and otherwise, we can still satisfy all the constraints in $\mathcal{I}_0$ and $\mathcal{I}_1$. Hence this is indeed an implementation of the function $T(.)$. The function $F(.)$ can be implemented in an analogous manner.

■

4.5.7 REP Helps Implement MAX SNP-Hard Functions

Lemma 21 *Suppose f is a non-trivial function which is neither 0-valid nor 1-valid. Then* MAXCSP($\{f, \text{REP}\}$) *implements the* XOR *function.*

Proof: Without loss of generality, assume $s = 0^p1^q$ is a satisfying assignment for f. We consider two disjoint set of variables $S_X = \{X_1, X_2, ..., X_p\}$ and $S_Y = \{Y_1, Y_2...., Y_q\}$. Consider the MAXCSP($\{f, \text{REP}\}$) instance which consists of constraints REP(X_1, X_i) for $i \in [2..p]$, constraints REP(Y_1, Y_j) for $j \in [2..q]$ and the constraint $f(X_1, X_2, ..., X_p, Y_1, Y_2, ..., Y_q)$. It is now easy to verify that this yields a $(p+q-1)$-implementation of the function $X_1 \oplus Y_1$ if $\bar{s} = 1^p0^q$ is a satisfying assignment, and of the function $\bar{X}_1Y_1$ otherwise. Now an application of the Composition Lemma yields the lemma. ■

Corollary 3 *Suppose f is a non-trivial function which is neither 0-valid nor 1-valid. Then* MAXCSP($\{f, \text{REP}\}$) *is* MAX SNP*-hard.*

Proof: Immediately follows from Lemma 14 and Lemma 21 above. ■

4.5.8 Unary Functions Help Implement either REP or MAX SNP-Hard Functions

Lemma 22 *Let f be a function which is not 2-monotone. Then* MAXCSP($\{f, T, F\}$) *can implement either the* XOR *or the* REP *function.*

Proof: Since f is not 2-monotone and non-trivial, it must be sensitive to at least two variables. Consider the boolean k-cube with each vertex s labeled by the function value $f(s)$; where k is the arity of function f. Let V_i denote the set of vertices labeled i, $i \in \{0,1\}$. If $|V_i| \leq |V_{1-i}|$, we claim that it must be the case that there exists a vertex in V_i which has at least two neighbors in V_{1-i}. This is readily seen using the expansion properties of the k-cube; any set S of at most 2^{k-1} vertices must have expansion factor at least one. Furthermore, the expansion factor is precisely one only when the set S induces a boolean $(k-1)$-cube. But the later case can't arise since it would imply that f is a single variable function. Hence there must exist a vertex $s \in V_i$ which has two neighbors in V_{1-i}.

Let s_i and s_j be these two neighbors of s, differing in the ith and the jth bit position respectively. Without loss of generality, we may assume that $i = 1$ and $j = 2$. Consider now the input instance which has a constraint of the form $f(X_1, X_2, Y_1, Y_2, ..., Y_{k-2})$ and constraints of the form $T(Y_i)$ for each Y_i appearing in $O(s) \cap O(s_1) \cap O(s_2)$ and of the form $F(Y_i)$ for each Y_i appearing in $Z(s) \cap Z(s_1) \cap Z(s_2)$. It is now easy to verify that this set of constraints implements one of the functions $X_1 + X_2$, $X_1 \oplus X_2$, $\bar{X}_1 + \bar{X}_2$, $\bar{X}_1 + X_2$ or $\overline{X_1 \oplus X_2}$. The former three implement $X_1 \oplus X_2$ while the later two implement the constraint $X_1 = X_2$. ■

The following is a straightforward corollary.

Corollary 4 *Let f be a function which is not 2-monotone. Then* MAXCSP$(\{f, T, F\})$ *is* MAX SNP*-hard.*

Proof: If MAXCSP$(\{f, T, F\})$ can implement the REP function, then the corollary follows using the Composition Lemma, the Hardness Lemma and the Lemmas 14 and 16. Otherwise, it follows from Lemma 15. ■

Lemma 23 *If $\mathcal{F}$ is a constraint set such that there exist (1) $f_0 \in \mathcal{F}$ which is not* 0*-valid, (2) $f_1 \in \mathcal{F}$ which is not* 1*-valid and (3) $f_2 \in \mathcal{F}$ which is not* 2*-monotone. The* MAXCSP$(\mathcal{F})$ *is MAX SNP-hard.*

Proof: If either f_0 or f_1 is C-closed then using the Replication Lemma, we can implement the REP function and using the Composition Lemma along with Lemma 21 allows to conclude that MAXCSP$(\{f_0, f_1, f_2\})$ implements XOR function.

If neither f_0 nor f_1 is C-closed, then using the Unary Lemma, MAXCSP($\{f_0, f_1, f_2\}$) can implement the unary functions $T(.)$ and $F(.)$, and then using the Composition Lemma along with Lemma 22, we conclude that MAXCSP($\{f_0, f_1, f_2\}$) implements either the XOR function or the REP function. In the latter case, we can use Lemma 21 to conclude that MAXCSP($\{f_0, f_1, f_2\}$) can implement the XOR function.

In either of the two situations above, we may conclude by Lemma 14 that MAXCSP($\{f_0, f_1, f_2\}$) is MAX SNP-hard. ■

4.6 Hardness at Gap Location 1

It is possible to use a notion closely related to α-implementation to conclude from Schaefer's dichotomy theorem and show that in the cases where SAT($\mathcal{F}$) in NP-hard to decide, it is actually hard to distinguish satisfiable instances from instances which are not satisfiable in a constant fraction of the constraints. This is termed hardness at gap location 1 by Petrank [89] who highlights the usefulness of such hardness results in other reductions.

An important characteristic of α implementation of a function f is that if we are given an assignment to the function variables which does not satisfy f, it can always be extended to the auxiliary variables such that precisely $(\alpha - 1)$ constraints are satisfied. This is a useful feature in establishing the hardness results for problems such as MAX 2-SAT which do not have hardness gaps located at 1. However, when dealing with problems with hardness gaps located at 1, such as MAX 3-SAT, it suffices to use a somewhat different notion of α-implementations, called *weak α-implementations*[1]. A weak α-implementation satisfies the condition (a)-(c) of the α-implementations and the condition (d) is replaced by the constraint that the MAXCSP($\mathcal{F}$) instance implementing it has precisely α constraints. Clearly, weak α-implementations can be composed together and they preserve hardness gaps located at 1.

It is not difficult to verify that Schaefer's proof is in fact based on weak α-implementations of functions, and hence one may directly conclude from his proof that his class of NP-hard satisfiability

[1]The name weak α-implementation is slightly misleading because this notion is simultaneously both weaker and stricter than the notion of α-implementations.

problems are all in fact MAX SNP-hard. This yields Theorem 24.

4.7 Strengthening Schaefer's Dichotomy Theorem

Schaefer's proof of NP-hardness in his dichotomy theorem relies on the ability to replicate variables within a constraint application. We observe that to do so, it suffices to create a weak implementation of the function REP. Since given a weak implementation, we can replace any p replicated copies of a variable X by p new variables $X_1, X_2, ..., X_p$ and add constraints of the form REP(X_1, X_2), REP(X_1, X_3), ..., REP(X_1, X_p). We now show how to create a weak implementation of the REP function.

Now Lemmas 19 and 20 show that MAXCSP$(\{f_0, f_1, f_2\})$, where f_0 is not 0-valid and f_1 is not 1-valid, can be used to create either a weak implementation of the function REP or a weak implementation of both unary functions T and F. In the latter case, we can show the following lemma.

Lemma 24 *If f is not weakly negative then* MAXCSP$(\{f, T, F\})$ *can weak implement either the function $x \oplus y$, or the function $x + y$. Similarly, if f is not weakly positive then* MAXCSP$(\{f, T, F\})$ *can weak implement either the function $x \oplus y$, or the function $\bar{x} + \bar{y}$.*

Proof: We only prove the first part - the second part follows by symmetry. We know that f has a maxterm S with at least two positive literals. We consider the function f' which is f existentially quantified over the variables not in S. Let x_1 and x_2 be the two positive literals in S. Set all other variables in S to the value which does not make S true. Then the assignment $x_1 = x_2 = 0$ is a non-satisfying assignment. The assignments $x_1 = 0 \neq x_2$ and $x_1 \neq 0 = x_2$ must be satisfying assignments ny the definition of maxterm. While the assignment $x_1 = x_2 = 1$ may go either way. Depending on this we get either the function $x \oplus y$ or $x + y$. ■

Corollary 5 *If f_2 is not weakly positive and f_3 is not weakly negative, then* MAXCSP$(\{f_2, f_3, T, F\})$ *weak implements (at gap* 1*) the* XOR *function.*

Since the SAT$(\mathcal{F})$ problems that we need to establish as NP-hard in Schaefer's theorem satisfy the condition that there exists $f_0, f_1, f_2, f_3 \in \mathcal{F}$ such that f_0 is not 0-valid and f_1 is not 1-valid,

f_2 is not weakly positive and f_3 is not weakly negative, we conclude that we can weak implement the XOR function. This, in turn, can be used to create a weak implementation of the function REP(x, y) by using the constraints $\{x \oplus z, y \oplus z\}$ for some auxiliary variable z. Thus replication can be eliminated from Schaefer's proof.

Chapter 5

The Approximability of Structure Maximization Problems

5.1 Introduction *

In this chapter, we continue with our study towards understanding the approximation behavior of optimization problems. We now consider another class of maximization problems built on Schaefer's problems. This new class of optimization problems that we consider, called MAXONES, captures some newer problems such as MAX CLIQUE, while continuing to capture some of the old problems like MAX CUT. A problem of this class is also defined by a constraint set $\mathcal{F}$. An instance to this problem consists of m constraints on n variables. The objective now is to find an assignment with the largest number of 1's which satisfies all m constraints. A good way to think about this problem is that we are trying to find the largest subset of a given set of variables among all "feasible" subsets, i.e., among all subsets that satisfy the given m feasibility conditions.

As before, we give a classification theorem to determine the approximability of any problem in this class from merely its description. Our classification theorem shows that this class contains problems that are solvable exactly, MAX SNP-complete problems, problems which are approximable to within polynomial factors but no better, and then problems which are not approximable to within any factor at all. This class thus contains a representative at every known approximability threshold for the naturally-arising maximization problems[1]. We feel that this class sheds light on the approximability of NPO maximization problems in general. Our results here serve to re-affirm the trends emerging from our study of approximability of these problems. In particular, they provide some evidence towards the non-existence of natural maximization problems with a $\log n$-threshold for approximability.

The rest of this paper is organized as follows. In Section 5.2, we formalize our problem and introduce various definitions needed to state our main result. Section 5.3 states the main theorem and gives an interpretation of our results. Section 5.4 establishes a correspondence between the approximation of the weighted and the unweighted problems. Finally, in Section 5.5, we describe the complete proof of our main theorem.

---

* This chapter is based on joint work with Madhu Sudan and David Williamson [71].

[1] Informally speaking, a problem Π has an approximability threshold of $f(n)$ if Π is approximable to within a factor of $O(f(n))$ and is NP-hard to approximate within a factor of $o(f(n))$.

5.2 Definitions

5.2.1 Problem Definition

We begin by defining our problem of interest; we build on the definitions introduced in the preceding chapter.

Definition 69 (Weighted MAXONE($\mathcal{F}$)**)** *Given a constraint set $\mathcal{F}$, the* weighted max ones problem MAXONE($\mathcal{F}$) *is defined as follows:*

- INPUT*: An instance is a triple $(\vec{x}, \vec{C}, \vec{w})$. It consists of a collection of m constraint applications of the form $C_j = (f_j, (i_1, \ldots, i_{k_j}))$, on boolean variables $x_1, \ldots, x_n$, and weights $w_1, \ldots, w_n$, where $f_j \in \mathcal{F}$, k_j is the arity of f_j, and the w_i's are non-negative integers.*
- OBJECTIVE: *Find a boolean assignment $x_1, \ldots, x_n$ satisfying all m constraint applications that maximizes $\sum_i w_i x_i$.*

If $w_i = 1$ for all i, we refer to the problem as the unweighted max ones problem *or simply the* max ones problem*, and the instance is simply a pair $(\vec{x}, \vec{C})$.*

We will also sometimes consider the following simpler decision problem.

Definition 70 (SAT($\mathcal{F}$)**)** *Given a constraint set $\mathcal{F}$, the* constraint satisfaction problem SAT($\mathcal{F}$) *is defined as follows:*

- INPUT*: A collection of m constraint applications of the form $\{(f_j, (i_1, \ldots, i_{k_j})\}$, on boolean variables $x_1, \ldots, x_n$, where $f_j \in \mathcal{F}$, and k_j is the arity of f_j.*
- OBJECTIVE: *Find a boolean assignment $x_1, \ldots, x_n$ satisfying all m constraints.*

We will be considering the approximability of the MAXONE($\mathcal{F}$) problems for different constraint sets $\mathcal{F}$.

Definition 71 *For a constraint set $\mathcal{F}$ and a function $\alpha : \mathcal{Z}^+ \to \mathcal{Z}^+$, an α-approximation algorithm for the weighted* MAXONE($\mathcal{F}$) *problem is an algorithm A which takes as input an instance*

$(\vec{x}, \vec{C}, \vec{w})$ *of* MAXONE($\mathcal{F}$) *with* n *variables, and* m *constraints, and produces an assignment* $x_1, \ldots, x_n$ *satisfying all constraints such that for every other feasible solution* y *to the given instance* $\sum_{i=1}^{n} w_i x_i \geq \alpha(n) \sum_{i=1}^{n} w_i y_i$. *A problem is said to* α*-approximable if it has an* α*-approximation algorithm with running time bounded by a polynomial in* m *and* n.

The important feature to notice above is that when α is allowed to be a function of the instance size, then it is measured as a function of only the number of variables n. It is important to note that the number of non-redundant constraints in any MAXONE problem is bounded by a polynomial in the number of variables. Hence fixing α to be a function of n alone does not result in much loss of information.

Before we can state our precise result, we need some characterizations of constraint functions. Thus we begin with some definitions needed to describe our main result.

5.2.2 Constraint Functions

We need the following additional definitions of contraint functions.

Definition 72 (Affine with Width 2) *A function is* affine with width 2 *if it can be expressed as a conjunction of linear equalities over GF(2) with at most two variables per equality constraint*.

Definition 73 (2CNF) *A function* f *is a* 2CNF *function if it can be expressed in conjunctive normal form with all disjuncts having at most two literals*.

We add one more definition to the above collection.

Definition 74 (Strongly 0-Valid) *We say a function* f *is* strongly 0-valid *if it is satisfied by any assignment with less than or equal to 1 ones*.

5.2.3 Families of Constraint Functions

We use the following shorthand notation:

- Let $\mathcal{F}_1$ be the family of all 1-valid functions.

- Let $\mathcal{F}_2$ be the family of all weakly positive functions.
- Let $\mathcal{F}_3$ be the family of all affine functions of width 2.
- Let $\mathcal{F}_4$ be the family of all affine functions.
- Let $\mathcal{F}_5$ be the family of all strongly 0-valid functions.
- Let $\mathcal{F}_6$ be the family of all weakly negative functions.
- Let $\mathcal{F}_7$ be the family of all 2CNF functions.
- Let $\mathcal{F}_8$ be the family of all 0-valid functions.

5.3 Overview of the Main Result

In this section, we formally state and interpret our results.

5.3.1 Main Theorem

Our main theorem is as follows.

Theorem 25 *1. If $\mathcal{F} \subset \mathcal{F}_i$ for some $i \in \{1,2,3\}$ then the weighted* MAXONE($\mathcal{F}$) *is in P.*

2. *If $\mathcal{F} \subset \mathcal{F}_4$ and $\exists f_i \in \mathcal{F} - \mathcal{F}_i$ for every $i \in \{1,2,3\}$ then the weighted* MAXONE($\mathcal{F}$) *is APX-complete.*

3. *If $\mathcal{F} \subset \mathcal{F}_i$ for some $i \in \{5,6,7\}$ and $\exists f_i \in \mathcal{F} - \mathcal{F}_j$ for every $j \in \{1,2,3,4\}$ then weighted* MAXONE($\mathcal{F}$) *is poly-APX complete.*

4. *If $\mathcal{F} \subset \mathcal{F}_8$ and $\exists f_j \in \mathcal{F} - \mathcal{F}_j$ for every $j \in \{1,\ldots,7\}$, then $\mathcal{F}$ is decidable but finding a solution of positive value is NP-hard.*

5. *If $\mathcal{F}$ is not a subset of $\mathcal{F}_j$ for any $j \in \{1,\ldots,8\}$, then finding any solution satisfying a given set of constraints is NP-hard.*

For contrast, we restate Schaefer's result here.

Theorem 26 (Schaefer [95]) *If* $\mathcal{F} \subset \mathcal{F}_i$ *for some* $i \in \{1, 2, 4, 6, 7, 8\}$*, then* SAT($\mathcal{F}$) *is in* P. *Otherwise* SAT($\mathcal{F}$) *is* NP*-hard.*

Notice that Case 5 of our main theorem follows immediately from Schaefer's theorem.

5.3.2 Discussion

A useful interpretation of the problems studied here is as follows. We study the approximation properties of integer programming problems, say IP($\mathcal{F}$), such that the objective function is of the form MAX $\sum_i w_i x_i$, w_i's are the weights associated with the variables, x_i's are 0/1 variables and the constraints are boolean functions of bounded arity drawn from the given family $\mathcal{F}$. Our main result shows that the precise asymptotic approximability of each problem in this class can be identified merely from the syntactic structure of the constraint set $\mathcal{F}$. We have already seen that certain integer programming frameworks when combined with restriction on the input structure, yield polynomial time approximation schemes. Barland, Kolaitis and Thakur [14] recently initiated a study of syntactic restrictions on a class of integer programs which enable constant factor approximations.

An interesting aspect of our results is that the approximability of the problems in this class is essentially independent of the weights. That is, any hardness of approximation result holds for the unweighted instances arising in IP($\mathcal{F}$), while the approximability results apply to arbitrary weighted instances arising in IP($\mathcal{F}$).

5.4 Preliminaries

In this section, we prove a few preliminary lemmas that we will need in the proof of our main theorem, particularly in Cases 2 and 3. We first show that in these cases, it is essentially equivalent for us to consider the weighted or unweighted MAXONE($\mathcal{F}$) problem. We then show that our ability to work with the weighted MAXONE($\mathcal{F}$) allows us in these cases to use a notion of "implementing" a constraint not in $\mathcal{F}$ in terms of other constraints in $\mathcal{F}$. The idea of implementing constraints is adapted from Chapter 4, and is used heavily in our proofs of Cases 2 and 3.

5.4.1 Weighted vs. unweighted problems

We begin with a slightly stronger definition of polynomial-time solvability of SAT($\mathcal{F}$) that we will need. We then show that given this stronger form of SAT($\mathcal{F}$) that insofar as APX-hardness and poly-APX-hardness are concerned, the weighted and unweighted cases of MAXONE($\mathcal{F}$) are equivalent. We conclude by showing that in Cases 2 and 3 the stronger form of SAT($\mathcal{F}$) holds.

Definition 75 *We say that a constraint satisfaction problem* SAT($\mathcal{F}$) *is* strongly decidable *if given* m *constraints on* n *variables* $x_1, \ldots, x_n$ and an index $i \in \{1, \ldots, n\}$, *there exists a polynomial time algorithm which decides if there exists an assignment to* $x_1, \ldots, x_n$ *satisfying all* m *constraints and additionally satisfying the property* $x_i = 1$.

Lemma 25 *For every strongly decidable function family* $\mathcal{F}$, *for every* ϵ *of the form* $1/l$ *for some positive integer* l *and for every non-decreasing function* $\alpha : \mathcal{Z}^+ \to \mathcal{Z}^+$, α*-approximating the weighted* MAXONE($\mathcal{F}$) *problem reduces to* α'*-approximating the (unweighted)* MAXONE($\mathcal{F}$) *problem, where* $\alpha'(n) = \frac{\alpha(\sqrt{\epsilon n})}{(1+\epsilon)}$.

Proof: Given an instance $(\vec{x}, \vec{C}, \vec{w})$ of a weighted MAXONE($\mathcal{F}$) problem, we create a sequence of instances of weighted MAXONE($\mathcal{F}$) problems with the final instance having $N \leq \frac{n^2}{\epsilon}$ variables, with the weight of every variable being 1, and with the property that given an $\alpha'(N)$-approximate solution to the final instance we can obtain an $\alpha(n)$-approximate solution to the instance I.

Assume w.l.o.g. that $w_1 \geq w_2 \geq \cdots \geq w_n$. From here onwards i will denote the smallest index such that there exists a feasible solution to I with $x_i = 1$. Notice that i can be computed in polynomial time. The instance I_1 is defined to be $(\vec{x}, \vec{C}, \vec{w'})$, where $w'_j = w_j$ if $j \geq i$ and $w'_j = 0$ otherwise. Since no solution of I has $x_j = 1$ for $j < i$, this change in the weight function does not alter the value of any solution. Thus I_1 is essentially equivalent to I. Notice that the weight of the optimal solution to I_1 is at least w'_i.

The instance I_2 is defined to be $(\vec{x}, \vec{C}, \vec{w''})$, where w''_j is w'_j rounded up to the nearest integral positive multiple of $\frac{\epsilon w'_i}{n}$. Notice that the net increase to the weight of any solution by this increase in the weights is at most $\epsilon w'_i$.

Finally the (unweighted) instance I_3 has as its variables N_j copies of every variable x_j, where $N_j = \frac{w''_j}{(\epsilon w'_i/n)}$. Notice that by the definition of w''_j's, the N_j's are integral and $1 \leq N_j \leq \frac{n}{\epsilon}$. Given a constraint C_i on variables $x_{j_1}, \ldots, x_{j_k}$, the instance I_3 has $N_{j_1} \times N_{j_2} \times \cdots \times N_{j_k}$ copies of the constraint C_i applied to all different collections of k-tuples of variables containing a copy of the j_1th variable in the first position, j_2th variable in the second position and so on. The weight of all variables is 1.

Let $N \stackrel{\triangle}{=} \sum_{j=1}^{n} N_j$ denote the number of variables in I_3. We will now show that given $\alpha'(N)$-approximate solution to I_3, we can reconstruct a $\alpha(n)$-approximate solution to I. It is easy to verify that any solution to I_3 can be modified to have all copies of a variable assigned to 1 or all assigned to zero, without changing any 1 to a 0. Thus every solution to I_2 of weight w can be transformed to a solution of weight $\frac{w}{w'_i} \cdot \frac{n}{\epsilon}$ for I_3 and vice versa. Thus given a solution of weight at least $\frac{\text{OPT}(I_3)}{\alpha'(N)}$, we can reconstruct (in polynomial time) a solution of weight at least $\frac{\text{OPT}(I_2)}{\alpha'(N)}$ for I_2. To conclude the argument it suffices to show that this can be used to construct an $\alpha(n)$-approximate solution to I_1.

Our candidate solutions will be the solution which assigns 1 to x_i and the solution to the instance I_2. If the value of the $\alpha'(N)$-approximate solution to I_3 is w, then the value of a solution so returned is at least $\max\{w'_i, w - \epsilon w'_i\}$ which is at least $\frac{w}{1+\epsilon}$. (The inequality $\max\{w'_i, w - \epsilon w'_i\} \geq \frac{w}{1+\epsilon}$ follows from a simple averaging argument.) Since $w \geq \text{OPT}(I_2)/\alpha'(N)$ and $\text{OPT}(I_1) \leq \text{OPT}(I_2)$, we find that this solution has value at least $\frac{\text{OPT}(I_1)}{(1+\epsilon)\alpha'(N)} = \frac{\text{OPT}(I)}{(1+\epsilon)\alpha'(N)}$. Thus this solution is a $(1+\epsilon)\alpha'(N)$ approximate solution to I.

Finally observe that since $N \leq \frac{n^2}{\epsilon}$, this is also an $(1+\epsilon)\alpha'(n^2/\epsilon) = \alpha(n)$-approximate solution to the instance I. ■

As our examination will eventually show, there is really no essential difference in the approximability of the weighted and unweighted problems. For now we will satisfy ourselves by stating this conditionally.

Corollary 6 *For any strongly decidable function family $\mathcal{F}$, the* MAXONE($\mathcal{F}$) *problem is APX-hard if and only if the weighted* MAXONE($\mathcal{F}$) *problem is APX-hard.*

Corollary 7 *For any strongly decidable function family $\mathcal{F}$, the* MAXONE($\mathcal{F}$) *problem is poly-APX-hard if and only if the weighted* MAXONE($\mathcal{F}$) *problem is poly-APX-hard.*

Before concluding we show that most problems of interest to us will be able to use the equivalence between weighted and unweighted problems.

Lemma 26 *If $\mathcal{F} \subset \mathcal{F}_j$ for some $j \in \{1, \ldots, 7\}$, then $\mathcal{F}$ is strongly decidable.*

Proof: For a function $f \in \mathcal{F}$ and an index $i \in \{1, \ldots, k\}$, let $f|_{(i,1)}$ be the function:

$$f|_{(i,1)}(x_1, \ldots, x_k) \stackrel{\text{def}}{=} f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_k).$$

Further let $\mathcal{F}'$ be the family:

$$\mathcal{F}' \stackrel{\text{def}}{=} \mathcal{F} \cup \{f|_{(i,1)} | f \in \mathcal{F}, i \in [k]\}.$$

First observe that the problem described in the lemma can be expressed as a problem of SAT($\mathcal{F}'$). Further, observe that if $\mathcal{F} \subset \mathcal{F}_j$ for $j \in \{1, 2, 3, 4, 6, 7\}$, then $\mathcal{F}' \subset \mathcal{F}_j$ as well. Lastly, if $\mathcal{F}' \subset \mathcal{F}_5$, then $\mathcal{F}' \subset \mathcal{F}_8$. Thus in each case we end up with a problem from SAT($\mathcal{F}$) which is in P by Schaefer's theorem. ■

5.4.2 Implementations and weighted problems

The ability to work with weighted problems now allows us to use existential quantification over auxiliary variables and the notion of functions "implementing" other functions. We start with the definition of implementing a constraint f — an adaptation of the definition given in the preceding chapter.

Definition 76 [Implementation] *A set of constraint applications $C_1, \ldots, C_m$ using constraints from $\mathcal{F}$ over a set of variables $\vec{X} = \{X_1, X_2, ..., X_p\}$ and $\vec{Y} = \{Y_1, Y_2, ..., Y_q\}$ form an* implementation *of a boolean function $f(\vec{X})$ iff the following conditions are satisfied:*

(a) *for any assignment of values to $\vec{X}$ such that $f(\vec{X})$ is true, there exists an assignment of values to $\vec{Y}$ such that all the constraint applications C_j are satisfied, and*

(b) *for any assignment of values to $\vec{X}$ such that $f(\vec{X})$ is false, no assignment of values to $\vec{Y}$ can satisfy all the constraints.*

A constraint set $\mathcal{F}$ implements a constraint f if there exist a implementation of f using constraints from $\mathcal{F}$.

The following lemma establishes the utility of implementations in showing hardness of approximating MAXONE problems.

Lemma 27 *Given function families $\mathcal{F}, \mathcal{F}'$, such that the weighted* MAXONE($\mathcal{F}'$) *problem has a $\alpha(n)$-approximation algorithm and every function $f \in \mathcal{F}$ can be implemented by the family $\mathcal{F}'$, then there exist constants c, d such that the weighted* MAXONE($\mathcal{F}$) *problem has a $\alpha(cn^d)$-approximation algorithm.*

Proof: Let $l = |\mathcal{F}|$ and k be the maximum arity of any function $f \in \mathcal{F}$. Let K be the largest number of auxiliary variables used in implementing any function $f \in \mathcal{F}$ by $\mathcal{F}'$. Notice K is a finite constant for any fixed $\mathcal{F}, \mathcal{F}'$.

Given an instance $I = (\vec{x}, \vec{C}, \vec{w})$ with m constraints on n variables of MAXONE($\mathcal{F}$), we create an instance I' of MAXONE($\mathcal{F}'$) as follows: I' has the variables $x_1, \ldots, x_n$ of I and in addition "auxiliary" variables $\{Y_i^j\}_{i=1,j=1}^{m,K}$. The weights corresponding to $x_1, \ldots, x_n$ is $w_1, \ldots, w_n$ (same as in I) and the auxiliary variables Y_i^j have weight zero. The constraints of I' implement the constraints of I. In particular the constraint $f_i(x_{i_1}, \ldots, x_{i_k})$ of I is implemented by a collection of constraints from $\mathcal{F}'$ (as dictated by the implementation of f_i by $\mathcal{F}'$) on the variables $(x_{i_1}, \ldots, x_{i_k}, Y_i^1, \ldots, Y_i^K)$.

By the definition of an implementation, it is clear that the every feasible solution to I can be extended (by some assignment to the Y variables) into a solution to I'. Alternately every solution to I' immediately projects into a solution of I. Furthermore, the value of the objective function is exactly the same (by our choice of weights). Thus a β-approximate solution to I' gives a β-approximate solution to I.

It remains to study this approximation as a function of the instance size. Observe that the instance size of I' is much larger. Let N denote the number of variables in I'. Then N is upper bounded by $Km + n$, where m is the number of constraints in I. But m, in turn, is at

most ln^k. Thus $N \leq (K+1)ln^k$, implying that an $\alpha(N)$-approximate solution to I', gives an $\alpha((K+1)ln^k)$-approximate solution to I. Thus an $\alpha(N)$-approximation algorithm for the weighted MAXONE($\mathcal{F}'$) problem yields an $\alpha(cn^d)$-approximation for the weighted MAXONE($\mathcal{F}$)-problem, for $c = (K+1)l$ and $d = k$. ■

The main corollaries of the above lemma are listed below.

Corollary 8 *If weighted* MAXONE($\mathcal{F}$) *is APX-hard and $\mathcal{F}'$ implements every function in $\mathcal{F}$, then weighted* MAXONE($\mathcal{F}'$) *is APX-hard.*

Corollary 9 *If weighted* MAXONE($\mathcal{F}$) *is poly APX-hard and $\mathcal{F}'$ implements every function in $\mathcal{F}$, then weighted* MAXONE($\mathcal{F}'$) *is poly APX-hard.*

Lastly we describe one more tool that comes in useful in creating reductions. This is the notion of implementing a property which falls short of being an implementation of an actual function. The target functions in the following definitions are the functions which force variables to being constants (either 0 or 1). However, sometimes we are unable to achieve this. So we end up implementing a weaker form which however suffices for our applications. We next describe this property.

Definition 77 [Existential Zero] *A family of constraints $\mathcal{F}$ can implement the existential zero property if there exists a set of m constraints $f_1, \ldots, f_m$ over n variables $\vec{X}$ and an index $k \in \{1, \ldots, n\}$ such that the following hold:*

- *There exists an assignment $a_{k+1}, \ldots, a_n$ to $X_{k+1}, \ldots, X_n$ such that assigning 0 to the first k variables $X_1, \ldots, X_k$ satisfies all constraints.*
- *Conversely, every assignment satisfying all the constraints must make at least one of the variables in $X_1, \ldots, X_k$ zero.*

An *Existential One* can be defined similarly.

Definition 78 *Given a constraint f of arity k and a set $S \subset \{1, \ldots, k\}$, the constraint $f|_{(S,0)}$ is a constraint of arity $k - |S|$ given by $f|_{(S,0)}(x_1, \ldots, x_{k-|S|}) = f(x_1, 0, 0, x_2, \ldots, x_{k-|S|}, 0, 0)$, where*

the zeroes occur in the indices contained in S. For a constraint set $\mathcal{F}$, the 0-closure of $\mathcal{F}$, denoted $\mathcal{F}|_0$ is the set of constraints $\{f_{(S,0)}|f \in \mathcal{F}, S \subset \{1,\ldots,k\}\}$.

Notice that $\mathcal{F}|_0$ essentially implements every function that can be implemented by $\mathcal{F} \cup \{0\}$, except the function $\{0\}$. We define $\mathcal{F}|_1$ similarly. Then $\mathcal{F}|_{0,1} = \mathcal{F}|_0 \cup \mathcal{F}|_1$.

Lemma 28 *If a family of constraints $\mathcal{F}$ can implement the existential zero property, then $\mathcal{F}$ can implement every function in the family $\mathcal{F}|_0$.*

Proof: We show how to implement the function $f(0, x_1, \ldots, x_{k-1})$. The proof can be extended to other sets by induction. Let the functions $f_1, \ldots, f_m$ implement the existential zero property on variables $Y_1, \ldots, Y_K$ with auxiliary variables $Y_{K+1}, \ldots, Y_N$. Then the constraints $f(Y_i, x_1, \ldots, x_{k-1})$, for $1 \leq i \leq K$, along with the constraints $f_1, \ldots, f_m$ on the variables $Y_1, \ldots, Y_N$ implement the constraint $f(0, x_1, \ldots, x_{k-1})$. (Observe that since at least one of the Y_i's in the set $Y_1, \ldots, Y_K$ is zero, the constraint $f(0, x_1, \ldots, x_{k-1})$ is being enforced.) Furthermore, we can always set all of $Y_1, \ldots, Y_K$ to zero, ensuring that any assignment to $x_1, \ldots, x_{k-1}$ satisfying $f(0, x_1, \ldots, x_{k-1})$ does satisfy all the constraints listed above. ■

Using the same proof as above, we also get the following lemma.

Lemma 29 *If a family of constraints $\mathcal{F}$ can implement the existential one property, then $\mathcal{F}$ can implement every function in the family $\mathcal{F}|_1$.*

5.5 Proof of Main Theorem

We start with the sub-cases that are easier to prove and then move on to the more difficult sub-cases. We first tackle the cases 1, 4 and 5 of Theorem 25. Recall that we already showed that Case 5 follows from Schaefer's theorem.

5.5.1 Case 1: The Polynomial-Time Case

Lemma 30 *The weighted* MAXONE($\mathcal{F}$) *problem is in* P *if each $f_i \in \mathcal{F}$ is 1-valid.*

Proof: Set each variable to one; this satisfies all constraints with a maximum possible weight of solution. ■

Lemma 31 *The weighted* MAXONE($\mathcal{F}$) *problem is in* P *if each* $f_i \in \mathcal{F}$ *is weakly positive*.

Proof: Consider the CNF formulae for the $f_i \in \mathcal{F}$ such that each clause has at most one negated variable. Clearly, clauses consisting of a single literal force the assignment of these variables. Setting these variables may create new clauses of a single literal; set these variables and continue the process until all clauses have at least two literals or until a contradiction is reached, in which case no feasible assignment is possible. In the former case, setting the remaining variables to one satisfies all constraints, and there exists no feasible assignment with a greater weight of ones. ■

Lemma 32 *The weighted* MAXONE($\mathcal{F}$) *problem is in* P *if each* $f_i \in \mathcal{F}$ *is an affine function of width 2*.

Proof: We reduce the problem of finding a feasible solution to checking whether a graph is bipartite, and then use the bipartition to find the optimal solution. Notice that each constraint corresponds to a conjunction of constraints of the form $x_i = x_j$ or $x_i \neq x_j$. Create a vertex x_j for each variable x_j and for each constraint $x_i \neq x_j$, add an edge (x_i, x_j). For each constraint $x_i = x_j$, identify the vertices x_i and x_j; if this creates a self-loop, then clearly no feasible assignment is possible. Check whether the graph is bipartite; if not, then there is no feasible assignment. If so, then for each connected component of the graph choose the larger weight side of the bipartition, and set the corresponding variables to one. ■

5.5.2 Case 4: The Decidable but Non-Approximable Case

Lemma 33 *If* $\mathcal{F} \subset \mathcal{F}_8$, *then* SAT($\mathcal{F}$) *is in* P.

Proof: Follows trivially since every instance is satisfiable. The assignment of 0's to every variable is a satisfying assignment to every instance. ■

Lemma 34 *If* $\mathcal{F} \not\subset \mathcal{F}_j$, *for any* $j \in \{1, \ldots, 7\}$, *then the problem of finding solutions of non-zero value to a given instance of (unweighted)* MAXONE($\mathcal{F}$) *is NP-hard*.

Proof: Given a function $f : \{0,1\}^k \to \{0,1\}$ and an index $i \in [k]$, let $f|_i$ be the function mapping $\{0,1\}^{k-1}$ to $\{0,1\}$ given by

$$f|_i(x_1,\ldots,x_k) \stackrel{\text{def}}{=} f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_k) \wedge f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_k).$$

Let $\mathcal{F}'$ be the set of functions defined as follows:

$$\mathcal{F}' \stackrel{\text{def}}{=} \mathcal{F} \cup \{f|_i \mid f \in \mathcal{F}, i \in [k]\}.$$

I.e., $\mathcal{F}'$ is the set of functions obtained by setting none or one of the variables of $\mathcal{F}$ to 1. We will argue that deciding SAT($\mathcal{F}'$) is NP-hard and then that deciding SAT($\mathcal{F}'$) reduces to finding non-zero solutions to MAXONE($\mathcal{F}$).

First observe that $\mathcal{F}' \not\subset \mathcal{F}_j$, for $j \in \{1,\ldots,8\}$. In particular it is not 0-valid, since $\mathcal{F}$ is not strongly 0-valid. Hence, once again applying Schaefer's result, we find that deciding SAT($\mathcal{F}'$) is NP-hard.

Now given an instance of SAT($\mathcal{F}'$) on variables $a_1,\ldots,a_n$ with constraints $C_1,\ldots,C_m$, with $C_1,\ldots,C_{m'} \in \mathcal{F}$ and $C_{m'+1},\ldots,C_m \in \mathcal{F}' \setminus \mathcal{F}$, consider the instance of MAXONE($\mathcal{F}$) defined on variable set

$$w_1,\ldots,w_{k+1},y_1,\ldots,y_n,z_1,\ldots,z_n$$

with the following constraints:

1. Let f be a non-1-valid function in $\mathcal{F}$. We introduce the constraint $f(w_1,\ldots,w_k)$.
2. For every constraint $C_i(a_{i_1},\ldots,a_{i_k}), 1 \le i \le m'$, we introduce two constraints $C_i(y_{i_1},\ldots,y_{i_k})$ and $C_i(z_{i_1},\ldots,z_{i_k})$.
3. For every constraint $C_i(a_{i_1},\ldots,a_{i_{k-1}}), m'+1 \le i \le m$, we introduce $2(n+k+1)$ constraints. For simplicity of notation, let $C_i(a_{i_1},\ldots,a_{i_{k-1}}) == g(1,a_{i_1},\ldots,a_{i_{k-1}}) \wedge g(0,a_{i_1},\ldots,a_{i_{k-1}})$ where $g \in \mathcal{F}$. The $2(n+k+1)$ constraints are:
   - $g(w_j,y_{i_1},\ldots,y_{i_{k-1}})$, for $1 \le j \le k+1$.
   - $g(z_j,y_{i_1},\ldots,y_{i_{k-1}})$, for $1 \le j \le n$.

- $g(w_j, z_{i_1}, \ldots, z_{i_{k-1}})$, for $1 \leq j \leq k+1$.
- $g(y_j, z_{i_1}, \ldots, z_{i_{k-1}})$, for $1 \leq j \leq n$.

We now show that the instance of MAXONE($\mathcal{F}$) created above has a non-zero satisfying assignment if and only if the instance of SAT($\mathcal{F}'$) has a satisfying assignment. Let $s = s_1 s_2 ... s_k$ be a satisfying assignment for the non 1-valid function f chosen above. First if $a_1, \ldots, a_n$ form a satisfying assignment to the instance of SAT($\mathcal{F}'$), then we claim that the assignment $w_j = s_j$ for $1 \leq j \leq k$, $w_{k+1} = 1$ and $y_j = z_j = a_j$ for $1 \leq j \leq n$ is a satisfying assignment to the instance of MAXONE($\mathcal{F}$) which has at least one 1 (namely w_{k+1}). Conversely, let some non-zero setting $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n, z_1, \ldots, z_n$ satisfy the instance of MAXONE($\mathcal{F}$). W.l.o.g./ assume that one of the variable $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n$ is a 1. Then we claim that the setting $a_j = z_j$, $1 \leq j \leq n$ satisfies the instance of SAT($\mathcal{F}'$). It is easy to see that the constraints $C_i(a_{i_1}, \ldots, a_{i_k})$, $1 \leq i \leq m'$, are satisfied. Now consider a constraint $C_i(a_{i_1}, \ldots, a_{i_{k-1}}) = g(0, a_{i_1}, \ldots, a_{i_{k-1}}) \wedge g(1, a_{i_1}, \ldots, a_{i_{k-1}})$. Since at least one of the variables in the set $w_1, \ldots, w_k$ is a 0 and at least one of the variables in the set $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n$ is 1, we know that both $g(0, z_{i_1}, \ldots, z_{i_{k-1}})$ and $g(1, z_{i_1}, \ldots, z_{i_{k-1}})$ are satisfied and hence $C_i(a_{i_1}, \ldots, a_{i_{k-1}}) = 1$. Thus the reduced instance of MAXONE($\mathcal{F}$) has a non-zero satisfying assignment if and only if the instance of SAT($\mathcal{F}'$) is satisfiable. ■

5.5.3 Case 5: The NP-Hard Case

Lemma 35 *If, for all $j \in \{1, \ldots, 8\}$, $\mathcal{F} \not\subset \mathcal{F}_j$, then deciding* SAT($\mathcal{F}$) *is NP-hard.*

Recall that we have already indicated that this follows from Schaefer's result [95].

5.5.4 Case 2: The APX-Complete Case

We now continue with the proof of Theorem 25; we first turn towards the proof of Case 2.

5.5.4.1 Membership in APX

Lemma 36 *If $\mathcal{F} \subset \mathcal{F}_4$, then the weighted* MAXONE($\mathcal{F}$) *problem is in APX.*

Proof: By Lemmas 26 and 25 it suffices to consider the unweighted case. In this case when all constraints are affine, then satisfying all constraints is essentially the problem of solving a linear system of equations over GF[2]. If the system is overdetermined, then no feasible solution exists. If the system is exactly determined, then the setting of all variables is forced, and we find the assignment with the maximum possible number of ones. If the system is underdetermined, then setting some number of variables arbitrarily determines the remainder of the solution; to be more precise, the variables x can be partitioned into x' and x'' such that $x' = Ax'' \oplus b$ for some 0/1 matrix A and some 0/1 vector b (where matrix arithmetic is carried out over GF[2]). Setting the variables in x'' to 1 with probability 1/2 thus ensures that the probability of each variable is 1 with probability 1/2. The expected number of ones is $n/2$, no worse than a factor of two from the maximum number. ■

5.5.4.2 APX-Hardness

Definition 79 [C-closed] *A constraint f is called* C-closed (or complementation-closed) *if for all assignments s, $f(s) = f(\bar{s})$.*

Definition 80 *The constraint* XOR *is defined to be the constraint* XOR$(x, y) = 1$ *if and only if* $x \oplus y = 1$. *The constraint* REP *is defined to be the constraint* REP$(x, y) = 1$ *if and only if* $x \oplus y = 0$.

Lemma 37 *Given a constraint f which is not* 1*-valid, the following hold:*

1. *If f is C-closed, then f implements* REP *and* XOR.
2. *If g is a constraint which is not C-closed, then $\{f, g\}$ implements an existential zero and an existential one.*
3. *$\{f\}$ either implements an existential zero or implements the functions* REP *and* XOR.

Proof: If f is C-closed, then it is not 0-valid. In this case, we can use the Replication Lemma from Chapter 4 to implement REP and XOR. The general idea there is to create two sets of variables X and Y. For each satisfying assignment s of f with l ones and $k - l$ zeroes, create all possible constraints by applying f to all possible sets of variables such that if $s_i = 0$ then the ith variable is a

variable from X, otherwise it is a variable from Y. One can then show that all variables in X must all be set to the same value, and all variables in Y to the opposite value. The constraint REP can be enforced by considering variables from the same set, and XOR by considering variables from both X and Y. This gives us Part (1) above.

For Part (2), if f is 0-valid, then the constraint $f(X_1, \ldots, X_k)$ implements an existential zero. Hence we can assume that f is neither 0-valid nor 1-valid. We follow the proof of the Unary Lemma from Chapter 4. The general idea there creates two sets of variables X and Y as above. We now use the non C-closed function g to impose the constraint that the variables in X must be set to zero, and those in Y to one, and hence implement the 1 and 0 functions. In particular we can impose the 0 and 1 functions, which gives the existential zero and one properties.

Part (3) follows from the fact that if f is C-closed, then Part (1) applies, else we can apply Part (2) to $\{f, f\}$. ■

Definition 81 *The function* XOR_3 *is defined to be the function* $\text{XOR}_3(x, y, z) = 1$ *if and only if* $x \oplus y \oplus z = 0$. *The function* XOR_4 *is defined to be the function* $\text{XOR}_4(w, x, y, z) = 1$ *if and only if* $w \oplus x \oplus y \oplus z = 0$.

Lemma 38 *The weighted problem* $\text{MAXONE}(\{\text{XOR}_3\})$ *is APX-hard.*

Proof: We reduce the MAX CUT problem to the weighted $\text{MAXONE}(\{\text{XOR}_3\})$ problem as follows. Given a graph $G = (V, E)$ we create a variable x_v for every vertex $v \in V$ and a variable x_e for every edge $e \in E$. The weight w_v associated with the vertex variable x_v is 0. The weight w_e of an edge variable x_e is 1. For every edge e between u and v we create the constraint $x_e \oplus x_u \oplus x_v = 0$. It is clear that any $0/1$ assignment to the x_v's define a cut and for an edge $e = \{u, v\}$, x_e is one iff u and v are on opposite sides of the cut. Thus solutions to the MAXONE problem correspond to cuts in G with the objective function being the number of edges crossing the cut. ■

Lemma 39 *The weighted problem* $\text{MAXONE}(\{\text{XOR}_4, \text{XOR}\})$ *is APX-hard.*

Proof: The proof is similar to that of Lemma 38. In this case, given a graph $G = (V, E)$, we create the variables x_v for every $v \in V$, x_e for every $e \in E$ and one global variable Z (which

is supposed to be zero) and $m \stackrel{\triangle}{=} |E|$ auxiliary variables $y_1, \ldots, y_m$. For every edge $e = \{u, v\}$ in G we impose the constraints $x_e \oplus x_u \oplus x_v \oplus Z = 0$. In addition we throw in the constraints $Z \oplus y_i = 1$ for every $i \in \{1, \ldots, m\}$. Finally we make the weight of the vertex variables and Z zero and the weight of the edge variables and the auxiliary variables y_i is made 1. The optimum to the so created MAXONE problem is MAX CUT$(G) + m$. Since MAX CUT$(G) \geq m/2$, this preserves APX-hardness. ■

Lemma 40 *Suppose* $\{f\}$ *implements the existential zero property and* g *is the constraint* $(x_1 \oplus \cdots \oplus x_p = b)$ *for some integer* $p \geq 3$ *and some* $b \in \{0, 1\}$. *Then the family* $\{f, g\}$ *implements* XOR_3.

Proof: Since $\{f\}$ implements the existential zero property, the set $\{f, g\}$ can implement $\{f, g\}|_0$ (using Lemma 28). In particular, $\{f, g\}$ can implement the constraints $x_1 \oplus x_2 = b$ and $x_1 \oplus x_2 \oplus x_3 = b$. Notice finally that the constraints $x_1 \oplus x_2 \oplus y = b$ and $y \oplus x_3 = b$ implement the constraint $x_1 \oplus x_2 \oplus x_3 = 0$. Thus $\{f, g\}$ implement the constraint XOR_3. ■

Lemma 41 *Suppose* $\{f\}$ *implements the* XOR *function and* g *is the constraint* $(x_1 \oplus \cdots \oplus x_p = b)$ *for some integer* $p \geq 3$ *and some* $b \in \{0, 1\}$. *Then the family* $\{f, g\}$ *either implements* XOR_3 *or* XOR_4.

Proof: First observe that f can also implement the function REP since $x_1 \oplus y = 1$ and $y \oplus x_2 = 1$ implement the constraint $x_1 \oplus x_2 = 0$.

Next observe that we can assume w.l.o.g. that $b = 0$. If not the constraints $x_1 \oplus \cdots \oplus x_{p-1} \oplus y = 1$ and $y \oplus x_p = 1$ implement the function $x_1 \oplus \cdots \oplus x_p = 0$.

Now if p is odd, then the constraints $x_1 \oplus \cdots \oplus x_p = 0$ and $\text{REP}(x_4, x_5)$, $\text{REP}(x_6, x_7)$ and so on up to $\text{REP}(x_{p-1}, x_p)$ implement the constraint $x_1 \oplus x_2 \oplus x_3 = 0$.

Lastly, if p is even, then the constraints $x_1 \oplus \cdots \oplus x_p = 0$ and $\text{REP}(x_5, x_6)$, $\text{REP}(x_7, x_8)$ and so on up to $\text{REP}(x_{p-1}, x_p)$ implement the constraint $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$. ■

Theorem 27 *If* $\mathcal{F} \subset \mathcal{F}_4$ *and* $\exists f_i \in \mathcal{F} - \mathcal{F}_i$ *for every* $i \in \{1, 2, 3\}$ *then* $\text{MAXONE}(\mathcal{F})$ *is APX-hard*.

Proof:[Sketch] Let $g(x_1, \ldots, x_q)$ be a non width-2 affine function and let $x_1, \ldots, x_p$ be a linear constraint of size greater than 2. Then the constraint $\exists x_{p+1}, \ldots, x_q$ s.t. $f(x_1, \ldots, x_q) = 1$

implements the constraint $x_1 \oplus x_2 \oplus \cdots \oplus x_p = 0/1$. Now let f be a non-1-valid function. If $\{f\}$ implements the existential zero property, then by Lemma 40 the set $\{f, g\}$ can implement XOR_3. Else, by Lemmas 37 and 41, $\{f\}$ implements XOR and $\{f, g\}$ either implements $\{\text{XOR}_3\}$ or $\{\text{XOR}_4\}$. Thus in any case $\{f, g\}$ can either implement the function XOR_3 or the set $\{\text{XOR}, \text{XOR}_4\}$, either of which is APX-hard. ■

5.5.5 Case 3: The Poly-APX-Complete Case

5.5.5.1 Membership in poly-APX

Lemma 42 *If $\mathcal{F} \subset \mathcal{F}_i$ for some $i \in \{1, 2, 3, 4, 5, 6, 7\}$ then* MAXONE($\mathcal{F}$) *can be approximated to within a factor of n.*

Proof: Schaefer's results imply a polynomial time algorithm to compute a feasible solution. If the feasible solution has at least one 1, we are done. Else, iteratively try setting every variable to one and computing a feasible solution. Note that if $\mathcal{F}$ is affine (or 2CNF), then the functions obtained by restricting some variable to be 1 remains affine (or resp. 2CNF), and thus this new class is still decidable. Lastly, a strongly 0-valid family remains 0-valid after this restriction and is still decidable. If the decision procedure gives no non-zero solution, then the optimum is zero, else we output a solution of value at least 1. ■

5.5.5.2 poly-APX-hardness

The lemma below shows that there exists a non-C-closed function in $\mathcal{F}$. Since there also exists a non-1-valid function in $\mathcal{F}$, by the proof of Lemma 37, we can implement an existential zero.

Lemma 43 *If $\mathcal{F} \subset \mathcal{F}_i$ for some $i \in \{5, 6, 7\}$, but $\mathcal{F} \not\subset \mathcal{F}_j$ for any $j \in \{1, 2, 3, 4\}$, then there exists a non-C-closed function in $\mathcal{F}$.*

Proof: Notice that a C-closed 0-valid function is also 1-valid. A C-closed weakly positive function will also be weakly negative. Lastly a C-closed 2CNF function is a affine function of width 2. Thus if $\mathcal{F}$ is 0-valid, then the function which is not 1-valid is not C-closed. If $\mathcal{F}$ is weakly

negative, the function which is not weakly positive is not C-closed. Similarly if $\mathcal{F}$ is 2CNF, then the function that is not affine is not C-closed. ■

As a consequence we obtain the following lemma.

Lemma 44 *If $\mathcal{F} \subset \mathcal{F}_i$ for some $i \in \{5,6,7\}$, but $\mathcal{F} \not\subset \mathcal{F}_j$ for any $j \in \{1,2,3,4\}$, then either $\mathcal{F}$ implements every function in $\mathcal{F}|_{0,1}$ or $\mathcal{F}$ implements $\mathcal{F}|_0$ and every function in $\mathcal{F}$ is 0-valid.*

Proof: By Lemma 43 there exists a non C-closed function f in $\mathcal{F}$. Also since $\mathcal{F}$ is not 1-valid, there exists a function $g \in \mathcal{F}$ which is not 1-valid. By Lemma 37 the set $\{f,g\}$ can implement an existential zero and hence $\mathcal{F}$ can implement every function in the family $\mathcal{F}|_0$. Furthermore, if $\mathcal{F}$ contains a non 0-valid function then it can implement $\mathcal{F}|_{0,1}$ (again by Lemma 37), else every element of $\mathcal{F}$ is 0-valid. ■

Our goal will be to implement the function $\bar{x}_1 + \cdots + \bar{x}_k$, for some $k \geq 2$. The following lemma shows that this will imply poly-APX-hardness.

Lemma 45 *If $f = \bar{x}_1 + \cdots + \bar{x}_k$, then* MAXONE$(\{f\})$ *is* poly-APX-*hard*.

Proof: We do a reduction from clique. Given a graph G, construct a MAXONE$(\{f\})$ instance consisting of a variable for every vertex in G and the constraint function f is applied to every subset of k vertices in G which does not induce a clique. It may be verified that the optimum number of ones in any satisfying assignment to the instance created in this manner is $\max\{k-1, \omega(G)\}$, where $\omega(G)$ is the size of the largest clique in G. Given a solution to the MAXONE$(\{f\})$ instance with l ones, it is easy to obtain a clique in G with $l/(k-1)$ vertices. Thus the existence of an α-approximation algorithm for the MAXONE$(\{f\})$ problem implies the existence of a $(k-1)\alpha$-approximation algorithm to the clique problem. The poly-APX-hardness of clique now implies the lemma. ■

We need to consider two cases: the case in which there exists a function that is not 0-valid, and the case in which all functions are 0-valid. In the former case, due to Lemma 44, we can afford to work with the family $\mathcal{F}|_{0,1}$. We first show that in this case, we can implement $\bar{x} + \bar{y}$. We will then turn to the case in which all functions are 0-valid and show that we can either implement $\bar{x}_1 + \cdots + \bar{x}_k$ or an existential one, and this will complete the proof of poly-APX-hardness.

Recall that we have a function in $\mathcal{F}$ that is not weakly positive and a function that is not affine.

Case A : MAXONE($\mathcal{F}$) implements an existential one. Lemmas 46-50 cover this case.

Lemma 46 *If f is not weakly positive, then the family $\{f\}|_{0,1}$ implements either* XOR *or $\bar{x}+\bar{y}$.*

Proof: Let $C = (\bar{x}_1 + \cdots + \bar{x}_p + y_1 + \cdots + y_q)$ be a maxterm in f with more than one negation i.e. $p \geq 2$. Substituting a 1 in place of variables $\bar{x}_3, \bar{x}_4, \ldots, \bar{x}_p$, a 0 in place of variables $y_1, y_2, \ldots, y_q$, and existentially quantifying over all variables not in C, we get a function f' such that $(\bar{x}_1 + \bar{x}_2)$ is a maxterm in f'.

By definition of maxterm, f' must be satisfied whenever $x_1 \oplus x_2 = 1$. Now if f' is also satisfied when $x_1 = x_2 = 0$, we get the function $\bar{x}_1 + \bar{x}_2$, else we get the function XOR(x_1, x_2). ■

Lemma 47 *The function (family)* $\{$XOR$\}$ *can implement the function* REP.

Proof: To implement REP(x, y), we include the constraints XOR(x, z) and XOR(y, z) for an auxiliary variable z. ■

Lemma 48 *Let g be a non-affine function. Then the function family $\{g,$* REP*,* XOR$\}|_{0,1}$ *can either implement the function $(x+\bar{y})$ or $(\bar{x}+\bar{y})$.*

Proof: Since g is non-affine, we essentially have the following situation for three satisfying assignments s_1, s_2 and s_3 for g.

									$g()$
s_1	00...0	00...0	00...0	00...0	11...1	11...1	11...1	11...1	1
s_2	00...0	00...0	11...1	11...1	00...0	00...0	11...1	11...1	1
s_3	00...0	11...1	00...0	11...1	00...0	11...1	00...0	11...1	1
$s_1 \oplus s_2 \oplus s_3$	00...0	11...1	11...1	00...0	11...1	00...0	00...0	11...1	0
	00...0	xx...x	yy...y	zz...z	$\bar{z}\bar{z}$...$\bar{z}$	$\bar{y}\bar{y}$...$\bar{y}$	$\bar{x}\bar{x}$...$\bar{x}$	11...1	

Fixing the above variables to 0's and 1's as shown in the last row, and assigning replicated copies of three variables x, y and z (and their negations using XOR), we get a function $h(x, y, z)$ with the following truth-table :

x \ yz	00	01	11	10
0	1	B	1	A
1	C	1	D	0

Figure 5.1: Truth-table of the function $h(x, y, z)$

The undetermined values in the table are indicated by the parameters A, B, C and D. The following analysis shows that for every possible value of these parameters, we can indeed implement an OR function using the constants 0 and 1.

$$\begin{aligned} A = 0 &\implies \exists\, x\, h(y,z) = y + \bar{z} \\ A = 1, B = 0 &\implies h(0,y,z) = \bar{y} + z \\ A = 1, B = 1, C = 0 &\implies h(x,0,z) = x + \bar{z} \\ A = 1, B = 1, C = 1, D = 0 &\implies h(x,y,1) = \bar{x} + \bar{z} \\ A = 1, B = 1, C = 1, D = 1 &\implies h(1,y,z) = y + \bar{z} \end{aligned}$$

■

Lemma 49 *The family* $\{x + \bar{y}, \text{XOR}\}$ *implements the function* $\bar{x} + \bar{y}$.

Proof: To implement $\bar{x} + \bar{y}$, we create an auxiliary variable x'. We now add two constraints, namely $x' + \bar{y}$, and $\text{XOR}(x, x')$. Clearly, all constraints are satisfied only if $\bar{x} + \bar{y}$ is satisfied. ■

We now summarize the effect of Lemmas 46-49 formally.

Lemma 50 *If* $\mathcal{F} \not\subset \mathcal{F}_i$ *for any* $i \in \{1, 2, 3, 4\}$*, then* $\mathcal{F}|_{0,1}$ *implements the function* $\bar{x} + \bar{y}$.

Proof: Lemma 46 yields that $\mathcal{F}|_{0,1}$ can either implement $\bar{x} + \bar{y}$, in which case we are done, or it can implement XOR. By Lemma 47 this implies $\mathcal{F}|_{0,1}$ can also implement REP. Lemma 48 and non-affineness of $\mathcal{F}$ imply in turn that in this case, $\mathcal{F}|_{0,1}$ can either implement $\bar{x} + \bar{y}$ or it implements $x + \bar{y}$. In the former case we are done, and in the latter we use Lemma 49 and the fact that $\mathcal{F}|_{0,1}$ can implement both XOR and $x + \bar{y}$ to conclude that it can implement $\bar{x} + \bar{y}$. ■

We now turn to the case in which all functions are 0-valid, and show that either we can implement $\bar{x}+\bar{y}$, or implement a 1. If the former occurs, we are done, and if the latter, we can apply Lemma 50.

Case B : All functions are 0-valid.

Lemma 51 *If f is 0-valid and not weakly positive, then the family $\{f\}|_0$ either implements $\bar{x}_1 + \cdots + \bar{x}_k$ for some $k \geq 2$ or it implements $x + \bar{y}$ or* REP.

Proof: Let $C = (\bar{x}_1 + \cdots + \bar{x}_p + y_1 + \cdots + y_q)$ be a maxterm in f with more than one negation i.e. $p \geq 2$ (such a maxterm exists since f is not weakly positive). Substituting a 0 in place of variables $y_1, y_2, \ldots, y_q$, and existentially quantifying over all variables not in C, we get a function g such that $(\bar{x}_1 + \bar{x}_2 + \cdots + \bar{x}_p)$ is a maxterm in g. Consider an unsatisfying assignment s for g with the smallest number of 1's and let k denote the number of 1's in s; we know $k > 0$ since the original function f is 0-valid. WLOG assume that s assigns value 1 to the variables $x_1, x_2, \ldots, x_k$ and 0's to the remaining variables. It is easy to see that by fixing the variables $x_{k+1}, x_{k+2}, \ldots, x_p$ to 0, we get a function $g' = (\bar{x}_1 + \bar{x}_2 + \cdots + \bar{x}_k)$. If $k > 1$, then this implements the function $(\bar{x}_1 + \cdots + \bar{x}_k)$ and we are done.

Otherwise $k = 1$, i.e. there exists an unsatisfying assignment s which assigns value 1 to exactly one of the x_i's, say x_1. Now consider a satisfying assignment s' which assigns 1 to x_1 and has a minimum number of 1's among all assignments which assign 1 to x_1. The existence of such an assignment easily follows from C being a maxterm in g. W.l.o.g. assume that $s' = 1^i 0^{p-i}$. Thus the function g looks as follows:

	x_1	x_2	$x_3 ... x_i$	$x_{i+1} ... x_p$	$g()$
s_1	0	0	00...0	00...0	1
s_2	1	0	00...0	00...0	0
$s' = s_3$	1	1	11...1	00...0	1
s_4	0	1	__..._	00...0	?

Existential quantification over the variables $x_3, x_4, \ldots, x_i$ and fixing the variables x_{i+1} through x_p to 0 yields a function g' which is either $(x_1 + \bar{x}_2)$ or REP(x_1, x_2). The lemma follows. ■

If we can implement $x + \bar{y}$, then the following lemma shows that we can essentially implement a 1, and thus we can reduce to the previous case.

Lemma 52 *If* MAXONE($\mathcal{F} \cup \{x + \bar{y}\}$) *is* α*-approximable for some function* α*, then so is* MAXONE($\mathcal{F} \cup \{1\}$).

Proof: Given an instance I of MAXONE($\mathcal{F} \cup \{1\}$) we construct an instance I' of MAXONE($\mathcal{F} \cup \{x + \bar{y}\}$) as follows: The variable set of I' is the same as that of I. Every constraint from $\mathcal{F}$ in I is also included in I'. The only remaining constraints are of the form $x_i = 1$ for some variables x_i (imposed by the function 1). We simulate this constraint in I' with $n - 1$ constraints of the form $x_i + \bar{x}_j$ for every $j \in \{1, \ldots, n\}$, $j \neq i$. Every non-zero solution to the resulting instance I' is also a solution to I, since the solution must have $x_i = 1$ or else every $x_j = 0$. Thus the resulting instance of MAXONE($\mathcal{F} \cup \{x + \bar{y}\}$) has the same objective function and the same feasible space and is hence at least as hard as the original problem. ■

Now by Lemma 51 the only remaining subcase is if we can implement REP. The following two lemmas show that in this case we can either implement $\bar{x} + \bar{y}$ or $x + \bar{y}$. If we can do the former, we are done, and if the latter, we can use $x + \bar{y}$ to implement a 1, and reduce to the previous case.

Lemma 53 *Let* f *be a non-affine function. Then there exist two satisfying assignments* s_1 *and* s_2 *such that* $s_1 \oplus s_2$ *is not a satisfying assignment for* f.

Proof: See, for example, Schaefer [95]. ■

Lemma 54 *If* f *is 0-valid function and non-affine, then* MAXONE($\{f, \text{REP}\}$) *implements either the function* $(\bar{x} + \bar{y})$ *or the function* $(x + \bar{y})$.

Proof: Using Lemma 53 and the fact that f is 0-valid, we essentially have the following situation:

					$g()$
	00...0	00...0	00...0	00...0	1
s_1	00...0	00...0	11...1	11...1	1
s_2	00...0	11...1	00...0	11...1	1
$s_1 \oplus s_2$	00...0	11...1	11...1	00...0	0
	00...0	$xx...x$	$yy...y$	$zz...z$	

Fixing the above variables to 0's as shown in the last row, and assigning replicated copies of three variables x, y and z, we get a function $h(x, y, z)$ with the following truth-table :

x \ yz	00	01	11	10
0	1	B	1	A
1	C	1	D	0

Figure 5.2: Truth-table of the function $h(x, y, z)$

The lemma now follows using an analysis identical to the one used in Lemma 48. ■

We are now ready to prove the final theorem for this section.

Theorem 28 *If $\mathcal{F} \not\subset \mathcal{F}_i$ for any $i \in \{1, 2, 3, 4\}$, then* MAXONE$(\mathcal{F})$ *is poly-APX-hard.*

Proof: As usual it suffices to consider the weighted problem. We will show that either $\mathcal{F}$ implements $\bar{x}_1 + \cdots + \bar{x}_k$ for some $k \geq 2$, or that $\mathcal{F}$ implements $x + \bar{y}$ and $\mathcal{F}|_1$ implements $\bar{x} + \bar{y}$. The theorem follows from an application of Lemma 52, which shows that MAXONE$(\mathcal{F})$ is as hard to approximate as MAXONE$(\mathcal{F}|_1)$, which in turn is poly-APX hard.

By Lemma 44 we have that $\mathcal{F}$ implements $\mathcal{F}|_0$. Furthermore, if $\mathcal{F}$ is not 0-valid, then $\mathcal{F}$ implements $\mathcal{F}|_{0,1}$ which in turn implements $\bar{x} + \bar{y}$ (by Lemma 50). Hence we are left with the case where $\mathcal{F}$ is 0-valid. In this case, by Lemma 51, $\mathcal{F}|_0$ (and hence $\mathcal{F}$) implements $\bar{x}_1 + \cdots + \bar{x}_k$ or $x + \bar{y}$ or REP. In the first case we are done. In the second case we get the second possibility from above since $\mathcal{F}|_1$ implements $\mathcal{F}|_{0,1}$ and $\mathcal{F}|_{0,1}$ implements $\bar{x} + \bar{y}$ (from Lemma 50 again). Finally if

$\mathcal{F}|_0$ can implement REP, Lemma 54 can be applied to claim that either $\mathcal{F}$ implements $\bar{x} + \bar{y}$ or $\mathcal{F}|_0$ implements $x + \bar{y}$. If $\mathcal{F}$ implements $\bar{x} + \bar{y}$ then we are done, else it implements $x + \bar{y}$. Again we apply Lemma 50 to conclude that in this case also $\mathcal{F}$ implements $x + \bar{y}$ and $\mathcal{F}|_1$ implements $\bar{x} + \bar{y}$.

■

Chapter 6

The Approximability of Graph Coloring

6.1 Introduction *

Given a graph G, a *legal coloring* of G is an assignment of a color to each vertex of G, such that no edge of G is connecting two vertices that are assigned the same color. The chromatic-number of G, denoted $\chi(G)$ is the minimum number of colors necessary for a legal coloring of G.

Given as input a graph G, one may want to come up with a legal coloring of G with the fewest possible colors. A polynomial-time algorithm that achieves coloring with a set of colors whose size is not vastly larger than the optimum ($\chi(G)$) has many practical applications. As an example consider the following computational problem: given a set of tasks to perform, where some of the tasks are pair-wise conflicting (say they cannot be carried out at the same time or at the same place), find a partition of the set of tasks such that no set of the partition contains two conflicting tasks. This problem is equivalent to the chromatic number problem: consider the graph whose vertices are all tasks, where two vertices are connected if the corresponding tasks are conflicting. A coloring of this graph is a non-conflicting partition of the set of tasks.

Coloring a graph G with the minimum $\chi(G)$ colors was shown to be NP-hard by Karp [63]; the results there imply that coloring a 3-colorable graph with 3 colors is NP-hard (this implies the same hardness result for k-colorable graph for any $k \geq 3$).

On the other hand, Blum [19, 20], following earlier work by Wigderson [98] provided a polynomial-time algorithm which, on input a 3-colorable n-vertex graph, finds a legal coloring using at most $O(n^{3/8} \log^{\frac{8}{5}} n)$ colors. More recently, Karger, Motwani and Sudan, designed a randomized polynomial time algorithm which uses no more than $O(n^{1/4} \log n)$. Their techniques also apply to k-colorable graphs where the guarantee is $O(n^{1-\frac{3}{(k+1)}})$ colors. In general, the best known performance guarantee is due to an algorithm by Halldórsson [49] that colors a graph G with $O(\frac{n(\log\log n)^2}{\log^3 n}) \cdot \chi(G)$ colors.

These results leave a huge gap between the achievable coloring and the apparently intractable. In this chapter, we slightly narrow this gap by strengthening Karp's hardness result. In particular, we show that separating between the case where $\chi(G) \leq 3$ and the case where $\chi(G) \geq 5$ is NP-hard. This implies that given a 3-colorable graph G, it is NP-hard to color G with 4 colors. An immediate

* This chapter is based on joint work with Nati Linial and Muli Safra [66].

corollary of this result is that for *any* fixed $k \geq 3$, it is NP-Hard to color a k-colorable graph with $k + 2\lfloor \frac{k}{3} \rfloor - 1$ colors.

Some results on the hardness of approximate coloring were obtained over the years. Garey and Johnson [40] have used composition operations on graphs to show that if, for every k, there exists a polynomial time procedure to color a k-colorable graph with fewer than $2k - 6$ colors then P=NP. However, they do not specify an integer k for which it is NP-hard to color a k-colorable graph with fewer than $2k - 6$ colors. Furthermore, their result clearly says nothing about the hardness of coloring a 3-colorable graph. Linial and Vazirani [78] have used squares of graphs to investigate the best ratio of approximating the chromatic number in polynomial time. They give evidence that this ratio, for n-vertex graphs is either below $\log^{1+o(1)} n$ or above $n^{\Omega(1)}$.

Building on the hardness of approximation results discovered through the PCP-based characterization of NP, Lund and Yannakakis [82] showed that for certain two constants $0 < c_1 < c_2 < 1$, it is NP-hard to color an n^{c_1}-colorable graph with n^{c_2} colors. The authors go on to prove that for every constant h there exists a constant c_h such that it is NP-hard to color a c_h-colorable graph by $c_h \cdot h$ colors. However, c_h depends on h and is relatively large in comparison; this is again not applicable to small fixed values of the chromatic number. More recently, Feige and Kilian [38] have shown a $(n^{1-\epsilon})$-hardness for the general graph coloring problem.

The remainder of this chapter is organized as follows. In Sections 6.2 and 6.3, we present a simple proof to both the theorems of Lund and Yannakakis [82] concerning the hardness of approximating the chromatic number. In Section 6.4 we prove that, given a graph G, it is NP-Hard to distinguish between the case where $\chi(G) \leq 3$ and the case where $\chi(G) \geq 5$, and therefore, it is NP-hard to color a 3-colorable graph using only 4 colors.

6.2 Hardness of Approximating the Chromatic Number when the Value May be Large

In this section, we give a simple proof that chromatic number is hard to approximates to within a factor of n^{ϵ}, while the value of the chromatic number itself may be $n^{\epsilon'}$ for some constant $0 < \epsilon' < 1$.

Notation

We use $\omega(G)$ to denote the size of the largest clique in a graph G. The chromatic number of G is denoted by $\chi(G)$, while its clique-cover number is denoted by $\bar{\chi}(G)$. We use $\bar{G}$ to denote the complement graph of G. Since any clique-cover of a graph G (a partition of G into cliques) is a legal coloring of the graph $\bar{G}$ (a partition of $\bar{G}$ into independents sets), it follows that $\bar{\chi}(G) = \chi(\bar{G})$.

The Starting Point

We start with an r-partite graph G that results from the reduction of [37], so it either has an r-clique (the maximum possible) or every clique of G is of small size.

Pictorially, one can think of the graph G as consisting of r rows, each being an independent set, and having some edges connecting vertices in different rows. Hence a clique of size r in G has exactly one representative in each row. Note that each row of G may have different number of vertices.

Let q be the maximum number of vertices in any row of G, and assume that either $\omega(G) = r$ or $\omega(G) < \frac{r}{q^\epsilon}$ for some absolute constant ϵ, $0 < \epsilon < 1$. For any $q = o(r)$, separating between these two cases is NP-hard [37, 8, 7, 90].

Our Approach

Given such a graph G with $q = r^\delta$ for some constant $\delta, 0 < \delta < 1$, we construct a graph H with rq' vertices for some $q' = (rq)^{O(1)}$ such that $\bar{\chi}(H) = q'$ when $\omega(G) = r$ and $\bar{\chi}(H) > q' \cdot q^\epsilon$ otherwise. Since $\bar{\chi}(H) = \chi(\bar{H})$, our result follows from the construction of such a graph H.

The graph H can be intuitively described as a transformation of G which makes H symmetric under certain rotations, so that an r-clique has several symmetric images which together cover H completely. On the other hand, the transformation also ensures that the size of the largest clique in H is at most that of G.

The graph H, like G, is an r-partite graph, and is constructed so that for any clique C in H there are $q' - 1$ additional cliques in H, that together cover all the vertices in all the rows in which C has a representative. It will be shown below that $\omega(G) = \omega(H)$. Therefore, if $\omega(G) = r$ then

$\bar{\chi}(H) = q'$. On the other hand, if $\omega(G) < \frac{r}{q^\epsilon}$ then by simply dividing the number of vertices in H by the size of the largest clique in H, we obtain the claimed lower-bound on $\bar{\chi}(H)$.

The Structure of H

The graph H is an r-partite graph; each row of H corresponds to a distinct row of G. A row of H consists of q' vertices (arranged in q' columns, denoted $0, ..., q'-1$). We consider a mapping (to be specified below) of the vertices in each row of G to the vertices in the corresponding row of H. We call the vertex v' of H, to which a vertex v of G is mapped, the *image vertex* of v.

We now describe how the edge set of H is constructed. For every edge (u, v) in G, we have an edge (u', v') in H where u' and v' are the images of u and v respectively. We refer to such an edge as a *direct* edge.

In addition, the edge set of H is extended to include all the *rotations* of the above edges as follows: If H contains an edge connecting the ith vertex in row k to the jth vertex in row l, then we also add to H edges connecting the $(i + m \bmod q')$th vertex in row k to the $(j + m \bmod q')$th vertex in row l for every $m \in \{1, .., q'-1\}$.

In order to fully describe H, it now only remains to describe the image function, which maps each vertex v of G to a unique vertex in H. However, from the above description alone we can conclude the following:

Lemma 55 *If $\omega(G) = r$ then $\bar{\chi}(H) = q'$.*

Proof: The graph H is *symmetric under rotation*; i.e., let $s^j(H)$ be the graph that results by rotating all the rows of H by j columns to the right (in a wraparound manner), then for every $j \in \{0, .., q'\}$, $s^j(H) = H$. Now, given a clique C of size r in G, consider the set C' which consists of the images in H of the vertices in C. Clearly, C' is a clique of H, and due to the symmetry of H under rotation, C' has $q'-1$ *rotational images* that together cover all the vertices of H. ■

The Image Function

For an integer $k > 0$ we denote by $[k]$ the set $\{0, ..., k-1\}$. As a preliminary for the description of the image function, from G to H, we start off with a simple lemma, showing the existence

of an injection T, of some special structure, from any set of size n into the range $[m]$, for some $m = n^{O(1)}$.

Lemma 56 *For every positive integer n, there exists a function $T: [n] \rightarrow [m]$ where $m = \Omega(n^5)$, such that for every distinct multiset $\{i_1, i_2, i_3\}$, $i_1, i_2, i_3 \in [n]$, the sum $T(i_1)+T(i_2)+T(i_3)$ mod m as well as its subsums (that is, $T(i_1) + T(i_2)$ mod m and $T(i_1)$) are distinct.*

Proof: We use an inductive argument to show the existence of a mapping which satisfies the property that for every distinct multiset $\{i_1, i_2, i_3\}$, $i_1, i_2, i_3 \in [n]$, the sum $T(i_1) + T(i_2) + T(i_3)$ mod m is distinct. Suppose that $T(0)$, ..., $T(l-1)$ have already been selected from $[m]$ such that for $0 \leq i_1, i_2, i_3 \leq l-1$, all the sums $T(i_1) + T(i_2) + T(i_3)$ mod m are distinct. An element $y \in [m]$ cannot be chosen for $T(l)$ if and only if there are $0 \leq i_1, i_2, i_3, i_4, i_5 \leq l-1$ such that

$$T(i_1) + T(i_2) + T(i_3) \equiv T(i_4) + T(i_5) + y \pmod{m}.$$

Therefore, at most l^5 elements y are ineligible at any step and if $m = \Omega(n^5)$, then the process can be carried through to yield the desired mapping.

Now observe that the mapping T shown to exist above must also satisfy the property that for every distinct multiset of size two, namely $\{i_1, i_2\}$, the sum $T(i_1) + T(i_2)$ mod m is distinct. This follows because if there exists two distinct mulisets of size two, namely $\{i_1, i_2\}$ and $\{j_1, j_2\}$ such that

$$T(i_1) + T(i_2) \equiv T(j_1) + T(j_2) \pmod{m}$$

then we can simply add the same element y to both multisets and get a contradiction to the property proven above that the sums of all triplets are distinct. It is clearly the case that the mapping T is an injection. ■

The Reduction

Let T be the injection shown to exist in Lemma 56 for the domain of all vertices of G, i.e., $n = rq$, and let $q' = m$, the size of the range of T. Hence,

$$T\colon \{0, ..., rq - 1\} \to \{0, ..., q' - 1\}.$$

The *image* of a vertex v in the ith row of G is the vertex of H which is in the ith row and $T(v)$ column of H.

We now show that our transformation of graph G into the graph H preserves the clique number, that is, G and H have the same clique number.

Lemma 57 $\omega(G) = \omega(H)$.

Note that this immediately yields our goal concerning the chromatic number of $\bar{H}$ since if $\omega(G) = r$ then $\bar{\chi}(H) = q'$ by Lemma 55. Otherwise, $\omega(G) < \frac{r}{q^\epsilon}$ and then simply counting the number of vertices of H and dividing by the size of the largest clique in H (using Lemma 57), we get

$$\bar{\chi}(H) > \frac{rq'}{\frac{r}{q^\epsilon}} = q' \cdot q^\epsilon.$$

Proof (of Lemma 57):

$\omega(H) \geq \omega(G)$: The image of a clique in G forms a clique in H.

$\omega(H) \leq \omega(G)$: We say that an edge (u, v) in G is the *origin* of an edge e in H if e is obtained via the images of u and v or by rotating the edge so obtained.

Using the special property of the image function T, we first show that every edge in H has a unique origin in G. Suppose, to the contrary, that an edge in H has two origins in G, namely the edges (u_1, v_1) and (u_2, v_2). Then we must have

$$T(u_1) - T(v_1) \equiv T(u_2) - T(v_2) \pmod{q'}$$

and it follows that

$$T(u_1) + T(v_2) \equiv T(u_2) + T(v_1) \pmod{q'}$$

which contradicts the property of T.

Given a clique C' in H, consider the origins of the edges connecting the vertices of C' between themselves. We claim that these origin edges are related to *consistent* vertex origins in the graph G in the following sense: there exists a mapping f, from every vertex $v' \in C'$, to a vertex $v \in G$, such that the origin of the edge (v'_1, v'_2), for $v'_1, v'_2 \in C'$, is the edge $(f(v'_1), f(v'_2))$ in G. Once this is proven, it follows immediately that the consistent vertex origins of C' form a clique in G, which implies our claim.

Assume, by way of contradiction, that the origins of the edges connecting vertices in C' are not consistent; then there would have been a triangle $\{u', v', w'\}$ such that the origin of the edge (u', v') is (u_1, v) and the origin of the edge (u', w') is (u_2, w), where $u_1 \neq u_2$. We show that such a triangle cannot exist in H.

Let, (v_1, w_1) be the origin in G of the edge (v', w'), and observe that:

$$\begin{aligned} T(u_1) - T(v) &\equiv \text{col}(u') - \text{col}(v') \pmod{q'} \\ T(v_1) - T(w_1) &\equiv \text{col}(v') - \text{col}(w') \pmod{q'} \\ T(w) - T(u_2) &\equiv \text{col}(w') - \text{col}(u') \pmod{q)'} \end{aligned}$$

where $\text{col}(x)$, for a vertex x of H, denotes the column of x in H. Combining the above equivalence relationships, we get

$$T(u_1) - T(v) + T(v_1) - T(w_1) + T(w) - T(u_2) \equiv 0 \pmod{q'}$$

and therefore

$$T(u_1) + T(v_1) + T(w) \equiv T(v) + T(w_1) + T(u_2) \pmod{q'}$$

which by the special property of T, and the fact that the origins of the edges connecting vertices u', v' and w' are in three distinct parts of G, implies that $u_1 = u_2$ (as well as that $v_1 = v$ and $w_1 = w$).

Hence we conclude that $\omega(H) \leq \omega(G)$. ■

6.3 Hardness of Approximating the Chromatic Number when the Value is (a Large) Constant

We now sketch how the argument in the previous section can be extended to eliminate the dependency of q' on r and thereby show that for any constant h there exists a constant c such that it is NP-hard to determine whether $\chi(G) \leq c$ or $\chi(G) \geq c \cdot h$.

We start with the graph G as described earlier, however, we now assume that q is a constant. Furthermore, the injection T which was previously defined over all the vertices in G, is now defined only over the vertices in any part of G, that is, the domain of T is restricted to $[q]$. As a result, several vertices may now map to the same column in H and thus an edge in H may not have a unique origin.

We will use the mapping T to transform G into a $(k \cdot r)$-partite graph H for some integer $1 < k < q$, such that if $\omega(G) = r$ then $\bar{\chi}(H) = q'$, and otherwise $\bar{\chi}(H) > \frac{q' \cdot q^{\epsilon}}{2}$. This transformation is better described through an intermediate graph G' where the graph G' is also a $(k \cdot r)$-partite graph. For every row of G, we include a block of k rows in G' such that the jth row in the block corresponding to the row i of G is simply the ith row shifted by j columns to the right in a wraparound manner. Thus each vertex of G has k copies in the graph G'. For every edge (u, v) in G, we insert an edge between every copy of vertex u and every copy of vertex v in G'. While doing so, we assume that each vertex of G is connected to itself. Thus all the k copies of any vertex form a clique in G'. It is easy to see that $\omega(G') = k \cdot \omega(G)$.

Once the graph G' is constructed, we transform it into the graph H in similar manner as before by applying the mapping T to the vertices in each row of G'. The edge set of H is also constructed in a similar manner. For every edge (u, v) in G', we have an edge (u', v') in H where u' and v' are the images of u and v respectively. We extend this edge set by including all the rotations of these edges. Let v be the kth vertex in the jth row of G, and let x be the vertex which is the copy of v in the ith row of the jth block of G', then the *image* of x is the vertex of H which is in the ith row of the jth block and $T(k + i \bmod q)$ column of H.

As noted earlier, a consequence of the fact that the domain of T is now restricted to $[q]$ is that an edge (u', v') in H may now have multiple origins in G'. However, the following claim shows

that this can happen only when u' and v' are in the same column in H.

Claim 1 *Every edge (u', v') in H such that $col(u') \neq col(v')$ has a unique origin edge in G'.*

Proof: Consider an edge (u', v') in H which has at least two distinct origins edge and let (x_1, y_1) and (x_2, y_2) be any two such distinct origin edges. Thus we must have

$$T(x_1) - T(y_1) \equiv T(x_2) - T(y_2) \pmod{q'}.$$

By the special property of T, we must have $\{x_1, y_2\} = \{x_2, y_1\}$. But since (x_1, y_1) and (x_2, y_2) are distinct edges, either $x_1 \neq x_2$ or $y_1 \neq y_2$. So it must be the case that $x_1 = y_1$ and $x_2 = y_2$. This immediately implies $col(u') = T(x_1) = T(y_1) = col(v')$. ■

We will now relate the clique numbers of the graphs G and H. Unlike the previous section where we obtained matching upper and lower bounds on $\omega(H)$ in terms of $\omega(G)$, we now characterize a range of values where $\omega(H)$ may lie.

Lemma 58 $k \cdot \omega(G) \leq \omega(H) \leq k \cdot \omega(G) + r$.

This characterization immediately yields our goal concerning the chromatic number of $\bar{H}$ since, if $\omega(G) = r$, then H contains a clique C' of size $k \cdot r$ which along with its $q' - 1$ rotational images covers all the vertices in H and thus $\bar{\chi}(H) = q'$. Otherwise, $\omega(G) < \frac{r}{q^\epsilon}$ and therefore,

$$\omega(H) < \frac{k \cdot r}{q^\epsilon} + r\,.$$

By choosing $k = \lceil q^\epsilon \rceil$, we get $\omega(H) < 2\frac{k \cdot r}{q^\epsilon}$. Now simply dividing the total number of vertices in H by the size of the largest clique in H, we get $\bar{\chi}(H) > \frac{q' \cdot q^\epsilon}{2}$.

Proof (of Lemma 58):

$\omega(H) \geq k \cdot \omega(G)$: The image of a clique C in G' forms a clique in H and $\omega(G') = k\omega(G)$.

$\omega(H) \leq k \cdot \omega(G) + r$: To see this, consider any clique C' in H. Ignore any blocks in H where the clique C' contains at most one representative. Thus we restrict ourselves to a subset $C'' \subseteq C'$ such that in every block of H, C'' either has zero or at least two representatives. Clearly, $|C'| \leq |C''| + r$ because C' can have precisely one representative in at most r blocks. Let b denote

the number of blocks in H such that C'' contains at least two representatives in them. We will show that $\omega(G) \geq b$.

Consider now an edge connecting two vertices $u', v' \in C''$ such that u' and v' are in the same block of H. We claim that it must be the case that these two vertices are in different columns and thus the edge (u', v') has a unique origin in G'. This follows rather easily from the observation that for every edge (u, v) in G' such that u and v are in the same block (and thus they are the copies of the same vertex x in G) $T(u) \neq T(v)$. We further use this observation to define a labeling L for each edge $e = (u', v')$ where u', v' are in the same block in H. We define $L(e) = x$ if the edge (u, v) is the origin of the edge (u', v') and u', v' correspond to two different copies of the vertex x in G.

We now show that for any edge $e_1 = (u'_1, w'_1)$ within a block of H, and an edge $e_2 = (u'_2, w'_2)$ also within a block of H, where u'_1, u'_2, w'_1, w'_2 form a 4-clique in H, it must be the case that $L(e_1)$ is connected to $L(e_2)$ in G. Therefore, the labels of the edges connecting vertices (within the same block) in C'' form a clique in G, which implies our claim that $\omega(G) \geq b$.

Since u'_1 and w'_1 are in different columns, as well as u'_2 and w'_2, we can assume, without loss of generality, that u'_1 and u'_2 are in different columns (as well as w'_1 and w'_2). Hence, the origin in G' of the edge (u'_1, u'_2) in H is uniquely defined, say (z_1, y_2); we show that z_1 is a copy of $L(e_1)$ and y_2 is a copy of $L(e_2)$ in G', which are connected in G', hence $L(e_1)$ is connected to $L(e_2)$ in G.

Let (x_1, y_1) be the origin in G' of the edge (u'_1, w'_1), we show that it must be the case that $x_1 = z_1$. A similar argument shows that the origin of e_2 is consistent with the origin of (u'_1, u'_2).

We break the argument into two cases:

- w'_1 and u'_2 are in different columns. The same argument as the triangle argument from the last section applies.

- w'_1 and u'_2 are in the same column. It must be the case that

$$T(x_1) - T(y_1) \equiv T(z_1) - T(y_2) \pmod{q'}.$$

Using the special property of T, we know that $\{x_1, y_2\} = \{z_1, y_1\}$. Since $T(x_1) \neq T(y_2)$, it

must be the case that $x_1 = z_1$.

■

6.4 Hardness of Approximating the Chromatic Number for Small Values

In this section we describe our proof that distinguishing between the case that the chromatic number of a graph is at most 3 and the case that it is at least 5 is NP-hard.

The Starting Point

As before, we start with an r-partite graph $G = (V, E)$ that either has an r-clique or every clique of G contains less than $\frac{r}{2}$ vertices. Separating between these two cases is NP-hard [37, 8].

Our Approach

Given such a graph G, we construct a graph H such that $\bar{\chi}(H) = 3$ when $\omega(G) = r$ and $\bar{\chi}(H) \geq 5$ when $\omega(G) < \frac{r}{2}$. Since the chromatic number of the graph $\bar{H}$ is equal to the clique cover number of H, our result follows from the construction of such a graph H.

For the purpose of this reduction, we assume that every vertex of G is connected to itself, and that every vertex is connected to at least one vertex in each row of G. It is easy to see that given the graph G, it can always be transformed into a graph G' such that G' satisfies both these assumptions and $\omega(G') = r$ iff $\omega(G) = r$ and $\omega(G') < \frac{r}{2}$ otherwise.

6.4.1 The Structure of H

The graph H is a multi-partite graph; the rows of H (partite sets) are divided into r *blocks*, one for each row (partite set) of G. The ith block of H consists of $O(q_i)$ rows (more precisely, $5q_i - 7$ rows), where q_i is the number of vertices in the ith row of G. Therefore, the number of rows in H is at most $O(rq)$, where q is the maximum number of vertices in any row of G.

Each of the rows in the ith block of H consists of 3 vertices (arranged in 3 columns, denoted 0, 1 and 2) and is associated with a ordered 3-way partition of the set of vertices of the ith row of

G (however, not all possible such partitions will be used in constructing the graph H). Thus each vertex in a row in the ith block of H is labeled by a single subset of the vertices of the ith row of G, which we refer to as the *label set* of that vertex. These labels, as we will see next, determine the edge set of H.

A vertex of H, labeled by a set of vertices $X \subset V$, is connected to a vertex labeled by the set $Y \subset V$, if there exists a vertex $u \in X$ and a vertex $v \in Y$ such that u and v are connected in G. (Note that due to our assumption that each vertex of G is connected to itself, two vertices in the same block of H whose label sets have a non-empty intersection are connected in H.) The edges inserted in this manner are referred to as *direct* edges.

In addition, the edge set of H is extended to include all the *rotations* of the above edges as follows: If H contains a direct edge connecting the ith vertex in row k to the jth vertex in row l, then we also add to H edges connecting the $(i+1 \bmod 3)$th vertex in row k to the $(j+1 \bmod 3)$th vertex in row l, and similarly between the $(i+2 \bmod 3)$th and $(j+2 \bmod 3)$th vertices in these rows.

For conciseness, a row whose vertices 0, 1 and 2 have labels X, Y and Z respectively, is said to have a *row label* of the form:

$$X \qquad Y \qquad Z$$

In order to fully describe H, it now only remains to describe the ordered partitions associated with each of its rows. However, from the above description alone we can conclude the following.

Claim 2 *Let S be a subset of the rows of H, and let C_0 be a clique that has one representative in each row in S. Then the set of rows S is coverable by 3 cliques.*

Proof: The graph H is symmetric under rotation; i.e., let $s(H)$ be the graph that results by rotating all rows of H by 1 to the right (in a wraparound manner), then $s(H) = s(s(H)) = H$. Therefore, C_0 has two rotational images, C_1 and C_2, that together cover all vertices in all rows in S. ■

The Ordered Partitions

Let $Q = \{v_1, ..., v_q\}$ denote the set of vertices in the ith row of G, and for each $j \in [1..q-1]$, let $X_j = \{v_{j+1}, ..., v_q\}$, $Y_j = \{v_1, ..., v_j\}$ and $Z_j = X_j \cup Y_{j-1}$ $(= Q - \{v_j\})$. We now describe how the ith block of rows in H is constructed:

- The first row is the trivial partition putting all the vertices in one set:

$$Q \qquad \emptyset \qquad \emptyset\,.$$

- For each $j, 1 \leq j < q$, H contains one row with labeling given by

$$X_j \qquad Y_j \qquad \emptyset\,,$$

  and another row that corresponds to the following ordering of the same partition :

$$Y_j \qquad X_j \qquad \emptyset\,.$$

- In addition, for each j, $1 < j < q$, H contains two rows whose labeling corresponds to a partition that singles out the unique vertex which is not included in Z_j :

$$Z_j \qquad \{v_j\} \qquad \emptyset\,,$$

  and

$$\{v_j\} \qquad Z_j \qquad \emptyset\,.$$

- And finally, for each j, $1 < j < q$, H contains a single row with labeling

$$X_{j+1} \qquad Y_j \qquad \{v_j\}\,.$$

6.4.2 A Clique of Size r in G Implies a 3-Coloring of $\bar{H}$

The following lemma is a rather immediate consequence of the construction of the edge set of H and claim 2.

Lemma 59 *If $\omega(G) = r$ then $\bar{\chi}(H) = 3$.*

Proof: Let $C = \{v_1, ..., v_r\}$ be the set of vertices forming a clique of size r in G. Clearly, each row of H contains one vertex whose labeling contains some $v_i \in C$; let C_0 be the set of those vertices in H. By the construction of the edge set of H, C_0 constitutes a clique with a representative in every row of H. Using Claim 2, we conclude that H is coverable by 3 cliques. ■

6.4.3 A 4-Coloring of $\bar{H}$ Implies a Large Clique in G

Our objective now is to show that $\bar{\chi}(H) \leq 4$ implies $\omega(G) = r$. We will show that if $\bar{\chi}(H) \leq 4$ then $\omega(G) \geq \frac{r}{2}$, which, by the constraint on the values taken by $\omega(G)$, implies that $\omega(G)$ must be r.[1]

Assume H has a 4-clique cover. We use such a cover to identify a set of vertices of G, say $S_{CLQ} = \{v_1, ..., v_r\}$, where each v_i belongs to a distinct row of G such that S_{CLQ} is a union of two cliques S_1 and S_2 in G. The construction of such a set S_{CLQ} is clearly sufficient to conclude that $\omega(G) \geq \frac{r}{2}$.

The Critical Cliques

Consider the ith block of rows in H. There are three cliques that contain a vertex in the row whose row label is of the form

[1] As an aside, it may be noted that by Lemma 59, this in turn implies that $\bar{\chi}(H) = 3$.

$$Q \qquad \emptyset \qquad \emptyset$$

where Q denotes the set of vertices in the ith row of G. To each of these cliques we assign a *shift*, which is either 0,1 or 2, according to whether the clique contains the first, second or the third vertex respectively.

Observe that the vertices in two rows (in different blocks) with row labels of the above form, are connected only if they appear in the same column. Hence a clique is assigned in this manner at most one shift value over all the blocks in H. However, one of the shift values may be assigned to two cliques. Nevertheless, we see that in such a case, we can replace all occurrences of one of the two cliques by the other in all rows of the above form. This follows from the structure of H and the assumption that any vertex in G is connected to at least one vertex in any row of G.

Therefore, we can assume from now on that all the rows of the above form are covered by precisely three cliques in the 4-clique cover of H. Let C_0, C_1 and C_2 denote these three cliques such that C_i is assigned shift value i in every block. We refer to these three cliques as the *critical* cliques while the remaining clique (which may be empty), denoted by C_N, is referred to as the *non-critical* clique.

The Voting Scheme

In any row of H, whose label is of the form

$$L_0 \qquad L_1 \qquad L_2,$$

we say that the critical clique C_s *votes for* L_i if C_s contains the vertex with label L_j where $i \equiv (j-s) \bmod 3$. That is, C_0 votes for the set labeling the vertex it contains, while C_1 and C_2 vote for the set labeling the vertex to the immediate left and right (in a wraparound manner) respectively,

of the vertex they contain in the above row. Thus in a row whose label has the form

$$Q \qquad \emptyset \qquad \emptyset\,,$$

each of C_0, C_1 and C_2 vote for Q.

Let us now consider a pair of rows in the same block of H which have the form

$$\begin{array}{lll} L_0 & L_1 & \emptyset \\ L_1 & L_0 & \emptyset \end{array}$$

and consider the votes that may be casted by the critical cliques in that pair. The following claims summarize some useful observations.

Claim 3 *In a row whose row label is of the form,*

$$L_0 \qquad L_1 \qquad \emptyset$$

no critical clique can vote for the empty set.

Proof : By definition, the critical clique C_i appears in the column i of the row whose row label is of the form

$$Q \qquad \emptyset \qquad \emptyset$$

Therefore, by our construction of the edge set of H, C_i can only appear in the columns i and $(i+1)$ mod 3 of the given row. Thus it can only vote for either the entry in column 0 or the entry in column 1 of the given row. The claim follows. ■

Claim 4 *In a pair of rows of H of the form*

$$\begin{array}{lll} L_0 & L_1 & \emptyset \\ L_1 & L_0 & \emptyset\,, \end{array}$$

the following two properties are always satisfied :

(a) a critical clique which appears in both rows either votes for L_0 in both rows or votes for L_1 in both rows, and

(b) two critical cliques which appear in both rows, either together vote for L_0 or together vote for L_1.

Proof : By Claim 3, we know that a critical clique only votes for L_0 or L_1 in either of the two rows. Since there are no vertical edges between the two rows, it cannot vote for L_0 in one row and L_1 in the other row of the pair. Thus property (a) follows. To see property (b), consider a critical clique, say C_i, which appears in both rows. By property (a), it either votes for L_0 or L_1 in both rows. Without loss of generality, assume it votes for L_0 in both rows. Observe now that the vertex which corresponds to the critical clique $C_{(i+1)\text{mod}3}$ voting for L_1 in the second row of the pair, is taken by C_i. Similarly, the vertex which corresponds to $C_{(i+2)\text{mod}3}$ voting for L_1 in the first row of the pair, is also taken by C_i. Thus if either $C_{(i+1)\text{mod}3}$ or $C_{(i+2)\text{mod}3}$ also appears in both rows of the pair, it must also vote for L_0. ■

We say that the label L_i is *elected* in the above pair of rows if the majority (i.e. two) of the critical cliques vote for L_i, $i = \{0, 1\}$. The following is a straightforward consequence of the preceding two claims.

Claim 5 *In a pair of rows of H of the form*

$$\begin{array}{lll} L_0 & L_1 & \emptyset \\ L_1 & L_0 & \emptyset\,, \end{array}$$

majority is always defined.

Proof : If all three critical cliques appear in this pair of rows, majority is clearly defined by Claim 4(a). On the other hand, if only two of the critical cliques appear in this pair of rows (i.e., the pair of rows is covered by only three cliques), then each of these two critical cliques appears in both the rows and therefore by Claim 4(b), both of them must vote for either L_0 or L_1. ■

For example, in Figure 6.1, C_0 votes for L_0 while C_1 and C_2 vote for L_1. Hence L_1 is elected in this pair of rows.

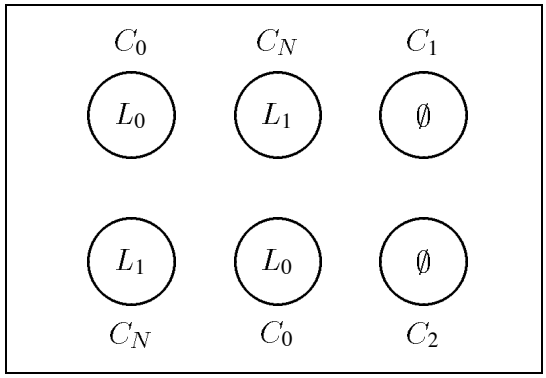

Figure 6.1: An example of critical cliques choosing L_1

Singling Out the Singletons

We have established so far that in every pair of rows which corresponds to two different orderings of a partition of the form $\langle L_0, L_1, \phi \rangle$, either L_0 or L_1 must get majority of the votes. Our next goal is to show that among all such pairs of rows in a block, there exists one such that the set elected is a singleton.

Lemma 60 *In every block B_i of rows in H, there exists a pair of rows corresponding to the partition of the form $\langle Q - \{v_j\}, \{v_j\}, \emptyset \rangle$ where $Q = \{v_1, ..., v_q\}$ is the set of vertices in the ith row of G, such that $\{v_j\}$ is elected by the majority of the critical cliques in this pair.*

Proof: Recall that we defined $X_j = \{v_{j+1}, ..., v_q\}$, $Y_j = \{v_1, ..., v_j\}$ and $Z_j = X_j \cup Y_{j-1}$, where $j \in [1..q-1]$. Now consider the sequence of pairs of rows with the row labels of the form:

$$\begin{array}{lll} X_j & Y_j & \emptyset \\ Y_j & X_j & \emptyset\,. \end{array}$$

If $Y_1 = \{v_1\}$ is elected in the first pair ($j = 1$), or $X_{q-1} = \{v_q\}$ is elected in the last pair ($j = q-1$), we are done. Otherwise, there must be a *switch point* $k \in [2..q-1]$ such that $X_{k-1}(= \{v_k, ..., v_q\})$ is elected in the $(k-1)$th pair and $Y_k(= \{v_1, ..., v_k\})$ is elected in the kth pair. We show that then it must be the case that $\{v_k\}$ is elected in the pair of rows with row label of the form:

$$\begin{array}{lll} Z_k & \{v_k\} & \emptyset \\ \{v_k\} & Z_k & \emptyset\,. \end{array}$$

Suppose by way of contradiction, the vertex with label Z_k is elected in the above pair of rows; we show that this makes it impossible to cover all three vertices in the row R whose row label is:

$$X_k \qquad Y_{k-1} \qquad \{v_k\}\,.$$

We argue that at most one of the critical cliques can cover a vertex in R; this clearly suffices as there is only one non-critical clique.

Consider the three pairs of rows which corresponds to the following three partitions:

- The partition just before the switch: $\langle X_{k-1}, Y_{k-1}, \emptyset \rangle$,
- The partition just after the switch: $\langle X_k, Y_k, \emptyset \rangle$, and
- The partition which singles out the vertex that causes the switch: $\langle Z_k, \{v_k\}, \emptyset \rangle$.

A critical clique C_i that contains a vertex in row R, votes for one of the three sets, $Y_{k-1}, \{v_k\}$ or X_k. Let W denote this set. Clearly, one of the above three partitions has the form $\langle W, Q - W, \emptyset \rangle$.

Let P be the pair of rows corresponding to this partition. By our assumption, the label $Q - W$ is elected in P. The critical clique C_i cannot vote for $Q - W$ in P since C_i votes for W in R and the edges connecting the vote for W in R to the votes for $Q - W$ in P do not exist in H (there are three direct edges connecting a row in P to R, however, none is a rotation of the edge that if existed in H were to connect W in R to $Q - W$ in P). Therefore, it must be the case that the two remaining critical cliques, namely $C_{(i+1) \bmod 3}$ and $C_{(i+2) \bmod 3}$, vote for $Q - W$ in P (otherwise $Q - W$ would not have been elected in P).

Let $j_1 \neq j_2 \in \{0, 1\}$ be the columns, of the vertices in the first and second rows of P respectively, that are labeled by $Q - W$. Consider the vertex in the first row and the $(j_1 + i) \bmod 3$ column of P, and the vertex in the second row and $(j_2 + i) \bmod 3$ column of P (if the critical clique C_i were to vote for $Q - W$, it would cover at least one of these two vertices). These two vertices cannot be contained in the critical cliques $C_{(i+1) \bmod 3}$ or $C_{(i+2) \bmod 3}$ as this would mean these cliques do not vote for $Q - W$, which then could not have been elected. Therefore, these two vertices must be contained in the remaining non-critical clique C_N (see figure 2). Thus C_N contains vertices in both column i and column $(i + 1) \bmod 3$ (this follows because $j_1 \neq j_2 \in \{0, 1\}$).

Now if another critical clique, say $C_{i'}$ ($i' \neq i$), appears in row R voting for some set W', there exists a different pair of rows, say P', such that it corresponds to a partition of the form $\langle W', Q - W', \emptyset \rangle$ and by our assumption, the label $Q - W'$ is elected in P'. By applying the same argument as before, we can conclude that C_N must contain a vertex in column i' and a vertex in column $(i' + 1) \bmod 3$ of this pair. This means C_N contains vertices in all three columns in the pairs P and P'. However, taking into account the edges connecting pairs P and P', this contradicts the following simple claim:

Claim 6 *In any block of H, consider a pair of rows, say P_1, with labels of the form:*

$$\begin{array}{lll} L_0 + S & L_1 & \emptyset \\ L_1 & L_0 + S & \emptyset \end{array},$$

and a clique C in H that contains vertex i, for $i \in \{0,1,2\}$, in the first row and vertex $(i+1)$ mod 3 in the second row (i.e., some shift of a choice that corresponds to label $L_0 + S$). Now consider the pair of rows, say P_2, with row labels of the form :

$$\begin{array}{lll} L_0 & L_1 + S & \emptyset \\ L_1 + S & L_0 & \emptyset\,. \end{array}$$

Then C cannot contain in this pair of rows a vertex in the $(i+2)$ mod 3 column.

Proof: Simply observe that the vertex i in the first row of P_1, is not connected to the vertex $(i+2)$ mod 3 in the first row of P_2 and similarly, the vertex $(i+1)$ mod 3 in the second row of P_1, is not connected to the vertex $(i+2)$ mod 3 in the second row of P_2. ■

Consequently, only one critical clique can appear in R and thus all the vertices of R could not have been covered by this clique cover. This is a contradiction. We therefore conclude that the vertex with label $\{v_k\}$ is elected in the pair of rows corresponding to the partition $\langle Z_k, \{v_k\}, \phi\rangle$. ■

The Singletons Form at most Two Cliques in G

Our final goal now is to show that the set of vertices formed by the singleton sets elected in each block (we associate exactly one singleton set with each block), is indeed the set S_{CLQ} we described earlier.

Lemma 61 *Let $S_{CLQ} = \{v_1, ..., v_r\}$ be a set of vertices of G, such that v_i belongs to the ith row of G, and $\{v_i\}$ is elected in the ith block of H; then S_{CLQ} is a union of two cliques in G.*

Proof: Let Q_i denote the set of vertices in the ith row of G, where $i \in \{1, ..., r\}$ and let P_i denote a pair of rows of the form below :

$$\begin{array}{lll} Q_i - \{v_i\} & \{v_i\} & \emptyset \\ \{v_i\} & Q_i - \{v_i\} & \emptyset \end{array}$$

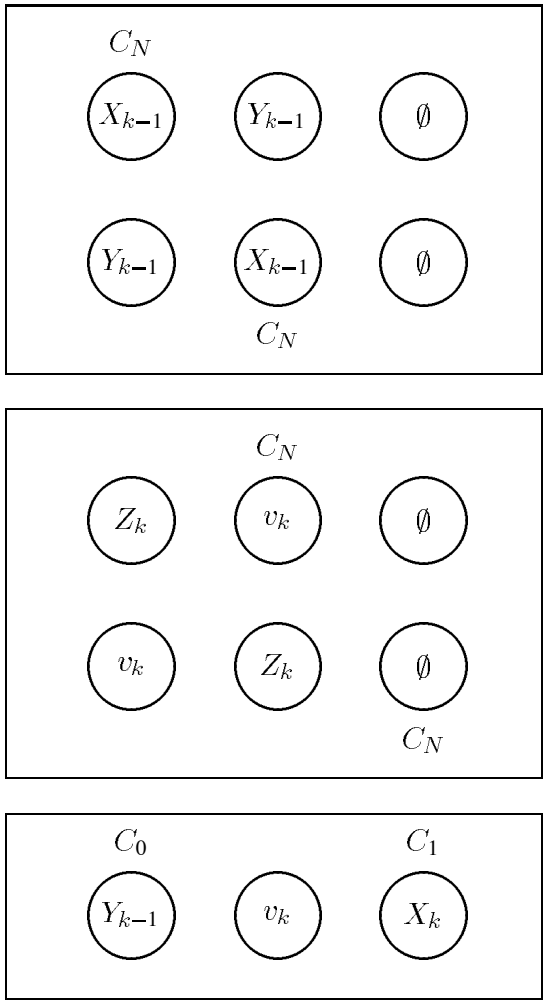

Figure 6.2: Critical cliques C_0 and C_1 vote for Y_{k-1} and v_k, respectively, in row r. This forces the non-critical clique C_N to cover 4 vertices, as indicated, which do not induce a clique in H.

where $v_i \in Q_i$. We say that C_N *holds* $\{v_i\}$ *with shift* $j \in \{0,1,2\}$ *in* P_i if C_N contains both the vertex in the $(j+1 \bmod 3)$th column of the first row, and the vertex in the jth column in the second row.

We define $S_1 \subseteq S_{CLQ}$ to be the set of vertices v_i such that, in the pair of rows P_i, C_N holds $\{v_i\}$ with some shift $j \in \{0,1,2\}$ and let $S_2 = S_{CLQ} - S_1$. We claim that S_1 and S_2 each form a clique in G.

Let us first look at the easier case which is that of S_2. In each P_i, such that $v_i \in S_2$, C_N does not hold $\{v_i\}$ with any shift j. Therefore, every critical clique that contains a vertex in P_i votes for $\{v_i\}$ and hence it must be the case that one of the critical cliques votes for $\{v_i\}$ in P_i and $\{v_{i'}\}$ in $P_{i'}$, and contains 3 vertices in these four rows. This is not possible unless v_i is connected in G to $v_{i'}$.

Now, regarding S_1, we are given $i \neq i'$ such that $v_i, v_{i'} \in S_1$, and such that C_N holds $\{v_i\}$ with shift j in P_i and it holds $\{v_{i'}\}$ with shift j' in $P_{i'}$. We show that, unless v_i is connected in G to $v_{i'}$, a 4-clique-cover, in which both v_i and $v_{i'}$ are elected, is not possible.

For clarity of exposition, let us represent the pair of rows P_i and $P_{i'}$ such that their first row is shifted one column to the left (note that we are not changing the edge set of H). Hence, we have P_i that looks like:

$$\begin{array}{lll} \{v_i\} & \emptyset & Q_i - \{v_i\} \\ \{v_i\} & Q_i - \{v_i\} & \emptyset \end{array}$$

and $P_{i'}$ that looks like:

$$\begin{array}{lll} \{v_{i'}\} & \emptyset & Q_{i'} - \{v_{i'}\} \\ \{v_{i'}\} & Q_{i'} - \{v_{i'}\} & \emptyset\,. \end{array}$$

The following claim is immediate now.

Claim 7 *Unless v_i is connected to $v_{i'}$ in G, H has no vertical edges connecting the first (second) row of P_i to the second (first) row of $P_{i'}$.* ■

By our assumption, C_N holds $\{v_i\}$ with some shift j in P_i and holds $\{v_{i'}\}$ with some shift j' in $P_{i'}$, i.e., C_N contains two vertices in the same column in both P_i and $P_{i'}$. If $j = j'$, by claim 7 we are done. Hence we assume, from now on, that $j \neq j'$; let $m \in \{0, 1, 2\}$ be such that $m \neq j$ and $m \neq j'$. Note that C_N does not contain any vertices in column m.

It must be the case that the critical cliques with shifts j' and m constitute the majority of votes in P_i, and the critical cliques with shifts j and m constitute the majority of votes in $P_{i'}$.

Since C_m votes for $\{v_i\}$ in P_i and $\{v_{i'}\}$ in $P_{i'}$, C_m contains vertices only in column m.

By claim 7, unless v_i is connected to $v_{i'}$, C_m either contains vertices only from the first row of P_i and the first row of $P_{i'}$ or the second row of P_i and second row of $P_{i'}$. However, clique C_j in P_i and clique C'_j in P_i can contain either the vertex in the first row in column m in P_i and the second row in column m in $P_{i'}$, respectively, or vice versa. This yields a contradiction. ■

6.4.4 Off with the 4th Clique

We can now conclude the following theorem:

Theorem 29 *Coloring a 3-colorable graph by 4 colors is NP-hard.*

Proof: From Lemma 61 we conclude that if $\bar{\chi}(H) \leq 4$ then $\omega(G) \geq \frac{r}{2}$, therefore $\omega(G) = r$. ■

The following is a rather straightforward corollary of theorem 29:

Corollary 10 *For any fixed k, it is NP-Hard to color a k-chromatic graph with at most $k + 2\lfloor \frac{k}{3} \rfloor - 1$ colors.*

Chapter 7

Conclusion

The last three decades have witnessed a continued progress in the design of approximation algorithms with provable performance guarantees. This progress has been complemented by a series of recent breakthroughs in hardness of approximation which have provided almost matching inapproximability results for many important optimization problems. Meanwhile, algorithmic research in approximation has witnessed another phenomenon whereby problem-specific techniques were often distilled into algorithmic paradigms which apply to entire classes of problems. The hardness of approximation research, on the other hand, has not seen a similar success in translating problem-specific results into general principles which uniformly apply to classes of problems.

Our work has developed frameworks whereby the hardness of approximation results for individual optimization problems could be effectively translated into a structural understanding of an entire class of optimization problems. For instance, our structure theorem translates the recently obtained hardness of approximation results for problems such as MAX 3-SAT and MAX CLIQUE into a structural characterization of approximation classes APX and poly-APX, respectively. Similarly, our study of constraint satisfaction-based maximization problems identified uniform transformations to show an approximation-preserving equivalence between well-understood individual problems on the one hand, and entire classes of problems on the other hand. These results represent new insights towards understanding the structure of the above-mentioned classes.

But this research represents only a relatively small step towards understanding the approximation behavior of optimization problems. Our work highlights a number of issues which deserve further investigation. For example, while the structure theorem yields a complete characterization for many approximation classes, the class PTAS still seems far from being well-understood. An important open problem is to identify natural complete problems for this class. Since the running time of a PTAS problem may have an arbitrary dependence on the error, it seems that any such completeness results would require reductions which impose weaker restrictions than the E-reductions. In particular, the PTAS reductions of Crescenzi and Trevisan [29] which allow the running time of reductions to depend on the error, may be an appropriate choice for this task.

Schaefer's framework of constraint satisfaction problems gave us a way to define two natural classes of maximization problems. Maximization problems built in this manner represent a well-behaved microcosm of NPO which captures many representative optimization problems. The

amenability of these classes to a unified analysis, in turn, provided us with a useful instrument to gain some formal insight into the approximation behavior of naturally-arising NP maximization problems. A natural complement to this work would be a study of minimization problems defined on a similar platform. Khanna, Sudan, and Trevisan [72] have initiated such a study and have obtained classification theorems which help unify many unresolved problems concerning the approximability of minimization problems. An interesting new line of research is to discover other ways of defining natural classes of optimization problems which are amenable to a formal analysis with respect to their approximation properties. While a collective body of formal results over many such well-behaved sub-classes of NPO may still not lead to a comprehensive understanding of the NPO as a whole, it could provide us with a much better understanding of NPO problems that arise in practice. An alternate direction for extending the above work is to consider constraints over non-boolean domains.

Finally, semidefinite programming based relaxations have resulted in improved performance guarantees for 3-coloring [65]. But these guarantees still require a polynomial number of colors for a 3-colorable graph. The huge gap between the upper and the lower bounds on the approximability remains virtually untouched by the modest progress described in this work. A resolution of this gap is perhaps one of the most fundamental open problems in approximation today. Any progress in merely changing the order of magnitude of this gap would represent a big step in our understanding of this problem.

Appendix A

Problem Definitions

A.1 SAT

INPUT : A collection C of disjunctive clauses of literals over a set of variables U.

QUESTION : Is there a truth assignment to U which satisfies all clauses in C ?

A.2 k-SAT

INPUT : A collection C of disjunctive clauses of literals over a set of variables U such that each clause has at most k literals.

QUESTION : Is there a truth assignment to U which satisfies all clauses in C ?

A.3 MAX k-SAT

INPUT : A collection C of disjunctive clauses of literals over a set of variables U such that each clause has at most k literals.

GOAL : Find a truth assignment to U which satisfies the largest number of clauses.

A.4 MAX CUT

INPUT : A graph $G = (V, E)$.

GOAL : Find a partition of V into disjoint sets S and T so as to maximize the number of edges in E with one end-point in S and the other in T.

A.5 s-t MIN CUT

INPUT : A directed graph $G = (V, E)$ and two vertices $s, t \in V$.

GOAL : Find a partition of V into disjoint sets $S \supseteq \{s\}$ and $T \supseteq \{t\}$ so as to minimize the number of edges in E going from S to T.

A.6 MAX CLIQUE

INPUT : A graph $G = (V, E)$.

GOAL : Find a largest size subset $V' \subseteq V$ such that any two vertices in V' are connected through an edge in E.

A.7 MAX INDEPENDENT SET

INPUT : A graph $G = (V, E)$.

GOAL : Find a largest size subset $V' \subseteq V$ such that no two vertices in V' are connected through an edge in E.

A.8 GRAPH COLORING

INPUT : A graph $G = (V, E)$.

GOAL : Find a coloring that uses minimum number of colors (A map $c : V \to [1..|V|]$ is a coloring if $c(u) \neq c(v)$ for all $(u, v) \in E$).

A.9 GRAPH k-COLORING

INPUT : A k-colorable graph $G = (V, E)$.

GOAL : Find a coloring of V minimizing the total number of colors.

A.10 TSP(1,2)

INPUT : A complete graph $G = (V, E)$ such that each edge in E has an assigned length of either one or two.

GOAL : Find a shortest cycle which visits all vertices in V.

A.11 MIN VERTEX COVER

INPUT : A graph $G = (V, E)$.

GOAL : Find a partition of V into disjoint sets $V_1, V_2, ..., V_k$ such that each V_i is an independent set and the total number of such sets is minimized.

A.12 MIN SET COVER

INPUT : A pair (U, C) where U is a finite set and C is a collection of subsets of U.

GOAL : Minimize $|C'|$ such that $C' \subseteq C$ and $\cup_{S \in C'} = \cup_{S \in C}$.

Bibliography

[1] P. Alimonti. New Local Search Approximation Techniques for Maximum Generalized Satisfiability Problems. In *Proceedings of the 2nd Italian Conference on Algorithms and Complexity* (1994), pp. 40–53.

[2] S. Arora. Probabilistic Checking of Proofs and Hardness of Approximation Problems. PhD Thesis, UC Berkeley (1994).

[3] S. Arora. Polynomial Time Approximation Scheme for Euclidean TSP and Other Geometric Problems. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science* (1996). To appear.

[4] S. Arora, L. Babai, J. Stern and Z. Sweedyk. The Hardness of Approximate Optima in Lattices, Codes, and Systems of Linear Equations. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science* (1996), pp. 724–733.

[5] S. Arora, D. Karger, and M. Karpinski. Polynomial Time Approximation Schemes for Dense Instances of NP-hard Problems. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing* (1995), pp. 284-293.

[6] S. Arora and C. Lund. Polynomial Time Approximation Scheme for Euclidean TSP and Other Geometric Problems. In *Approximation Algorithms for NP-hard Problems* (D. Hochbaum, Ed.), PWS Publishing, Boston (1996).

[7] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Hardness of Approximation Problems. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science* (1992), pp. 14–23.

[8] S. Arora and S. Safra. Probabilistic Checking of Proofs; a New Characterization of NP. In *Proceedings of the 33rd IEEE Symposium on Foundation of Computer Science* (1992), pp. 1–13.

[9] G. Ausiello, P. Crescenzi and M. Protasi. Approximate Solution of NP Optimization Problems. *Theoretical Computer Science*, 150 (1995), pp. 1–55.

[10] G. Ausiello and M. Protasi. Local Search, Reducibility, and Approximability of NP Optimization Problems. *Inform. Process. Lett.*, 54 (1995), pp. 73–79.

[11] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (1985), pp. 421–429.

[12] L. Babai. Transparent Proofs and Limits to Approximation. *Technical Report* TR-94-07, Dept. of Computer Science, University of Chicago (1994).

[13] B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41:153–180 (1994).

[14] I. Barland, P. G. Kolaitis and M. N. Thakur. Integer Programming as a Framework for Optimization and Approximability. In *Proceedings of the 11th Annual Conference on Structure in Complexity Theory* (1996).

[15] M. Bellare. Proof Checking and Approximation: Towards Tight Results. In *Sigact News* (Complexity Theory Column), Vol.27, No. 1 (1996).

[16] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs and non-approximability – towards tight results. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science* (1995), pp. 422–431.

[17] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing* (1993), pp. 294–304.

[18] P. Berman and M. Furer. Approximating Maximum Independent Set in Bounded Degree Graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (1993), pp. 365–371.

[19] A. Blum. An $O(n^{0.4})$ approximation algorithm for 3-coloring. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing* (1989), pp. 535–542.

[20] A. Blum. Some Tools for Approximate 3-Coloring. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science* (1990), pp. 554–562.

[21] H. L. Bodlaender. Some Classes of Graphs with Bounded Treewidth. *Bulletin of the European Association for Theoretical Computer Science* (1988), pp. 116–126.

[22] H. L. Bodlaender. A Linear Time Algorithm for Finding Tree-decompositions of Small Treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing* (1993), pp. 226–234.

[23] D.P. Bovet and P. Crescenzi. **Introduction to the Theory of Complexity.** Prentice-Hall, New York (1993).

[24] N. Chiba, T. Nishizeki, and N. Saito. Applications of the Planar Separator Theorem. *Journal of Information Processing*, 4:203–207 (1981).

[25] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing* (1971), pp. 151–158.

[26] N. Creignou. A Dichotomy Theorem for Maximum Generalized Satisfiability Problems. *Journal of Computer and System Sciences*, 51:3: 511–522 (1995).

[27] N. Creignou and M. Hermann. Complexity of Generalized Satisfiability Problems. *Research report* 1994-2, Group de recherche en algorithmique et logique, Caen (1994).

[28] P. Crescenzi and A. Panconesi. Completeness in approximation classes. *Information and Computation*, vol. 93 (1991), pp. 241–262.

[29] P. Crescenzi and L. Trevisan. On approximation scheme preserving reducibility and its applications. Lecture Notes in Computer Science, vol. 880 (1994), pp. 330-341.

[30] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. In *Proceedings of the 1st Annual International Conference on Computing and Combinatorics*, Lecture Notes in Comput. Sci. 959, pp. 539–548. Springer-Verlag (1995).

[31] P. Crescenzi, R. Silvestri, and L. Trevisan. To Weight or not to Weight: Where is the Question? In *Proceedings of the 4th Israel Symp. on Theory of Computing and Systems* (1996), pp. 68–77.

[32] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis, The complexity of multiway cuts. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pp. 241–251 (1992).

[33] K. Edwards, The complexity of colouring problems, Theoretical Comput. Sci., 43:337–343 (1986).

[34] R. Fagin. Generalized First-Order Spectra and Polynomial-time Recognizable Sets. In Richard Karp (ed.), **Complexity of Computer Computations**, AMS (1974).

[35] T. Feder and M. Vardi. Monotone monadic SNP and constraint satisfaction. In *Proceedings of the 25th ACM Symposium on Theory of Computing* (1993).

[36] U. Feige. A Threshold of ln n for Approximating Set Cover. In *Proceedings of the 28th ACM Symposium on Theory of Computing* (1996) , pp. 314–318.

[37] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science* (1991), pp. 2–12.

[38] U. Feige and J. Kilian. Zero Knowledge and the Chromatic Number. In *Proceedings of the 11th Annual Conference on Structure in Complexity Theory* (1996) , pp. 278–287.

[39] A. Frieze and M. Jerrum, Improved approximation algorithms for MAX k-CUT and MAX BISECTION. In *Proceedings of the 4th International Conference on Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science 920, pp. 1–13. Springer-Verlag (1995).

[40] M. R. Garey and D. S. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM*, vol. 23 (1976), pp. 43-49

[41] Michael R. Garey and David S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W. H. Freeman (1979).

[42] N. Garg, H. Saran, and V. Vazirani. Finding separator cuts in planar graphs within twice the optimal. In *Proceedings of the 35th Annual ACM Symposium on Foundations of Computer Science* (1994), pp. 14–23.

[43] N. Garg, V. V. Vazirani, and M. Yannakakis, Approximate max-flow min-(multi)cut theorems and their applications. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (1993), pp. 698–707.

[44] M. X. Goemans and D. P. Williamson, A General Approximation Technique for Constrained Forest Problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 307–316.

[45] M. X. Goemans and D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42, pp. 1115–1145 (1995).

[46] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (1985), pp. 291–304.

[47] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45:1563–1581 (1966).

[48] M. Grigni, E. Koustsoupias, and C. Papadimitriou. An Approximation Scheme for Planar Graph TSP. In *Proceedings of the 36th Annual ACM Symposium on Foundations of Computer Science* (1995), pp. 640–645.

[49] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Inform. Process. Lett.*, vol. 45 (1993), pp. 19–23

[50] M. M. Halldorsson and J. Radhakrishnan. Improved Approximations of Independent Sets in Bounded-Degree Graphs. In *Proceedings of the 4th Scandanavian Workshop on Algorithm Theory* (1994), pp. 194–206.

[51] J. Hartmanis and L. Berman. On isomorphisms and density of NP and other complete sets. In *Proceedings of the 8th ACM Symp. on Theory of Computing* (1976), pp. 30–40.

[52] J. Håstad. Clique is Hard to Approximate within $n^{1-\epsilon}$. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science* (1996). To appear.

[53] Dorit S. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discrete Applied Mathematics*, vol. 6 (1982), pp. 243–254.

[54] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz, and R. E. Stearns. Approximation Schemes using L-reductions. In *Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science* (1994), pp. 342-353.

[55] N. Immerman. Descriptive Complexity: a Logician's Approach to Computation. In *Notices of the American Mathematical Society*, 42(10): 1127–1133 (1995).

[56] N. Karmarkar and R.M. Karp. An Efficient Approximation Scheme For The One-Dimensional Bin Packing Problem. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science* (1982), pp. 312–320.

[57] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations* (R.E. Miller and J.W. Thatcher, Eds.), pp. 85–103, Plenum, New York (1972).

[58] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences*, vol. 9 (1974), pp. 256-278.

[59] D. S. Johnson. The Tale of the Second Prover. In the NP-completeness Column: An On-going Guide. *Journal of Algorithms*, vol. 13 (1992), pp. 502–524.

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

[61] V. Kann. On the Approximability of NP-complete Optimization Problems. Ph.D. Thesis, Department of Numerical Analysis and Computing Science. Royal Institute of Technology, Stockholm, Sweden (1992).

[62] V. Kann, S. Khanna, A. Panconesi, and J. Lagergren. On the Hardness of Approximating MAX k-CUT and its Dual. In *Proceedings of the 4th Israel Symp. on Theory of Computing and Systems* (1996), pp. 61–67.

[63] R. M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pp. 85–103. Plenum Press (1972).

[64] D. Karger, R. Motwani and G.D.S. Ramkumar. On approximating the longest path in a graph. In *Proceedings of the Third Workshop on Algorithms and Data Structures* (1993), pp. 421–432.

[65] D. Karger, R. Motwani and M. Sudan. Approximate Graph Coloring by Semidefinite Programming. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science* (1994), pp. 2–13.

[66] S. Khanna, N. Linial, and S. Safra. On the Hardness of Approximating the Chromatic Number. In *Proceedings of the 2nd Israel Symp. on Theory of Computing and Systems* (1993), pp. 250–260.

[67] S. Khanna, R. Motwani, M. Sudan, and U.V. Vazirani. On Syntactic versus Computational Views of Approximability. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science* (1994), pp. 819–830.

[68] S. Khanna and R. Motwani. Towards a Syntactic Characterization of PTAS. In *Proceedings of the 28th ACM Symposium on Theory of Computing* (1996) , pp. 329–337.

[69] S. Khanna, R. Motwani, and S. Vishwanathan. Approximating MAX SNP Problems via Semi-Definite Programming. *In preparation* (1996).

[70] S. Khanna and M. Sudan. The Optimization Complexity of Constraint Satisfaction Problems. *Stanford University Technical Note*, STAN-CS-TN-96-29 (1996).

[71] S. Khanna, M. Sudan and D.P. Williamson. The Optimization Complexity of Structure Maximization Problems. *In preparation* (1996).

[72] S. Khanna, M. Sudan and L. Trevisan. Constraint Satisfaction: The Approximability of Minimization Problems. *In preparation* (1996).

[73] Phokion G. Kolaitis and Madhukar N. Thakur. Approximation Properties of NP Minimization Classes. *JCSS*, 50(3):391–411 (1995).

[74] Phokion G. Kolaitis and Madhukar N. Thakur. Logical Definability of NP Optimization Problems. *Information and Computation*, 115(2):321–353 (1994).

[75] R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:1, pp. 155–171 (1975).

[76] L. A. Levin. Universal sorting problems. *Problems of Information Transmission*, 9, pp. 265–266 (1973).

[77] D. Lichtenstein. Planar Formulae and their Uses. *SIAM Journal on Computing*, 11:329–343 (1980).

[78] N. Linial and U. Vazirani. Graph Products and Chromatic Numbers. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science* (1989), pp. 124–128.

[79] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Mathematics*, 36:177–189 (1979).

[80] R.J. Lipton and R.E. Tarjan. Applications of Planar Separator Theorem. *SIAM Journal on Computing*, 9:615–627 (1980).

[81] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. *20th International Colloqium on Automata, Languages and Programming*, pp. 40–51 (1993).

[82] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41:960-981 (1994).

[83] R. Motwani. Lecture Notes on Approximation Algorithms. Report No. STAN-CS-92-1435, Department of Computer Science, Stanford University (1992).

[84] R. Motwani and P. Raghavan. **Randomized Algorithms.** Cambridge University Press (1995).

[85] A. Panconesi and D. Ranjan. Quantifiers and Approximation. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing* (1990), pp. 446–456.

[86] C.H. Papadimitriou. **Computational Complexity.** Addison-Wesley (1994).

[87] C. H. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. *Journal of Computer and System Sciences*, vol. 43 (1991), pp. 425–440.

[88] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, vol. 18 (1993), pp. 1–11.

[89] E. Petrank. The Hardness of Approximation: Gap Location. *Computational Complexity*, vol. 4 (1994), pp. 133–157.

[90] S. Phillips and S. Safra. PCP and tighter bounds for approximating MAX-SNP. Manuscript (1992).

[91] S. Poljak and Z. Tuza, The max-cut problem – a survey. In L.Lovasz and P.Seymour editors, *Special Year on Combinatorial Optimization*, DIMACS series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society (1993).

[92] P. Raghavan and C.D. Thompson. Randomized Rounding. *Combinatorica*, 7 (1987), pp. 365–374.

[93] N. Robertson and P. D. Seymour. Graph Minors. II. Algorithmic Aspects of Treewidth. *Journal of Algorithms*, 7:309–322 (1986).

[94] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22:115–124 (1975).

[95] T. Schaefer. The complexity of satisfiability problems In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing* (1978), pp. 216–226.

[96] W. Fernandez de la Vega and G.S. Lueker. Bin Packing can be solved within $1 + \epsilon$ in Linear Time, *Combinatorica*, 1:349–355 (1981).

[97] M. Sudan. Efficient Checking of Polynomials and Proofs, and the Hardness of Approximation Problems. PhD Thesis, UC Berkeley (1992).

[98] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, vol. 30, No. 4 (1983), pp. 729–735

[99] M. Yannakakis. The analysis of local search problems and their heuristics. In *Proceedings of the 7th Annual Symposium of Theoretical Aspects of Computer Science* (1990), pp. 298–311.