Polynomial Time Approximation Schemes for Dense Instances of $\mathcal{NP}$-Hard Problems*

(Extended Version)

Sanjeev Arora† David Karger‡ Marek Karpinski§

Abstract

We present a unified framework for designing polynomial time approximation schemes (PTASs) for "dense" instances of many $\mathcal{NP}$-hard optimization problems, including maximum cut, graph bisection, graph separation, minimum k-way cut with and without specified terminals, and maximum 3-satisfiability. By dense graphs we mean graphs with minimum degree $\Omega(n)$, although our algorithms solve most of these problems so long as the average degree is $\Omega(n)$. Denseness for non-graph problems is defined similarly. The unified framework begins with the idea of *exhaustive sampling:* picking a small random set of vertices, guessing where they go on the optimum solution, and then using their placement to determine the placement of everything else. The approach then develops into a PTAS for approximating certain *smooth* integer programs where the objective function and the constraints are "dense" polynomials of constant degree.

*To appear in J. Comput. System Sci., 1998.

†Princeton University. Supported by an NSF CAREER Award NSF CCR-9502747 and an Alfred Sloan Fellowship. email: arora@cs.princeton.edu. URL: http://www.cs.princeton.edu/~arora

‡MIT Laboratory for Computer Science. Work done at AT&T Bell Laboratories. email: karger@lcs.mit.edu URL: http://theory.lcs.mit.edu/~karger

§University of Bonn. Supported in part by the International Computer Science Institute, Berkeley, California, by the DFG Grant KA 673/4-1, ESPRIT BR Grants 7097, 21726, and EC-US 030, and by the Max-Planck Research Prize. Email: marek@cs.bonn.edu , URL: http://theory.cs.uni-bonn.de/~marek.

1 Introduction

Approximation algorithms, whenever they can be found, are a way to deal with the $\mathcal{NP}$-hardness of optimization problems. Ideally, they should run in polynomial time and have a small *approximation ratio*, which is the worst-case ratio of the value of the solution returned by the algorithm to the value of the optimum solution. (This definition is for minimization problems; for maximization problems the ratio is inverted so that it is always at least 1.)

Optimization problems seem to be approximable to different degrees (see [Shm94] for a survey). We know that unless $\mathcal{P} = \mathcal{NP}$, problems such as CLIQUE [FGL+91, AS92, ALM+92] and CHROMATIC NUMBER [LY93] cannot be approximated even to within a factor of n^{δ} in polynomial time, for some fixed $\delta > 0$. (More recently, Håstad [H96] showed that if SAT does not have randomized polynomial-time algorithms, then CLIQUE cannot be approximated to within a factor $n^{1-\delta}$, for every $\delta > 0$.) Others problems, such as those related to graph separators [LR88], have algorithms with approximation ratios close to $O(\log n)$. No inapproximability results are known for them. MAX-$\mathcal{SNP}$ problems, such as MAX-CUT or MAX-3-SAT, can be approximated to within some fixed constant factor but no better [PY91, ALM+92]. Only a few problems, such as KNAPSACK [S75] and BIN PACKING [FL81], are known to have *polynomial time approximation schemes (PTASs).*

A PTAS is an algorithm that, for every fixed $\epsilon > 0$, achieves an approximation ratio of $1+\epsilon$ in time that is polynomial in the input size (but could grow very fast with $1/\epsilon$, such as $O(n^{1/\epsilon})$). A PTAS thus allows us to trade off approximation accuracy for running time. (In the previous definition, if the running time is polynomial in $1/\epsilon$ as well, then we have a *fully polynomial time approximation scheme*. These are known to exist for a few problems [GJ79, DFK91, KLM89].)

Unfortunately, recent results ([ALM+92]) show that if $\mathcal{P} \neq \mathcal{NP}$, then PTASs do not exist for many $\mathcal{NP}$-hard problems. In particular, this is is true for every *MAX-$\mathcal{SNP}$-hard* problem. (The class of MAX-$\mathcal{SNP}$-hard problems includes VERTEX COVER, MAX-3-SAT, MAX-CUT, METRIC TSP, MULTIWAY CUTS, and many others [PY91].)

Note that the inapproximability results mentioned above, like all $\mathcal{NP}$-

hardness results, rule out approximation only on worst case instances of the problem. They do not rule out the existence of algorithms (heuristics) that do well on *most* instances. This observation is the starting point of our research.

This paper gives PTASs for a large class of $\mathcal{NP}$-hard problems when the problem instance is *dense*. The definition of *denseness* depends on the problem; for example, dense graphs are graphs with $\Omega(n^2)$ edges while dense 3-SAT formulas are those with $\Omega(n^3)$ clauses. Note that almost all graphs (asymptotically speaking) are dense, as are almost all 3-SAT instances.

The design of many (but not all) of our PTAS's relies on the observation that many optimization problems can be phrased as *nonlinear* integer programs in which the objective function is a low degree polynomial. For dense problems, the optimum value of the objective function is quite large. Thus, to achieve a *multiplicative* approximation for dense instances it suffices to achieve an *additive* approximation for the nonlinear integer programming problem. We design such an additive approximation algorithm (see Sections 1.2 and 1.3).

In the remainder of this introduction, we describe the problems we solve and sketch our solution techniques.

1.1 Applicable Optimization Problems

We now describe the problems to which we apply our techniques. The reader will note that the problems span a broad spectrum. Some, like maximum cut and maximum k-satisfiability, are MAX-$\mathcal{SNP}$-complete. Thus they do not have PTASs on general (that is, non-dense) instances [ALM+92], but they can all be approximated within some constant factor in polynomial time [PY91]. Others, like graph bisection and separation, do not currently have any algorithms with approximation ratios better than $O(\log n)$ on general instances. It is an open problem whether they are hard to approximate.

MAX-CUT: Partition the vertices of an undirected graph into two groups so as to maximize the number of edges with exactly one endpoint in each group. An algorithm in [GW94] achieves an approximation ratio of 1.13 for the problem.

MAX-DICUT: The directed version of the MAX-CUT problem. An algorithm in [FG95] (improving [GW94]) achieves an approximation ratio of 1.15.

MAX-HYPERCUT(d): A generalization of MAX-CUT to hypergraphs of dimension d; an edge is considered cut if it has at least one endpoint on each side.

SEPARATOR: Partition the vertices of a graph into two groups, each with at least 1/3 of the vertices, so as to minimize the number of edges with exactly one endpoint in each group. An algorithm in [LR88] achieves approximation ratio $O(\log n)$ (though it may produce a $1/4 : 3/4$ separator instead of a $1/3 : 2/3$ separator).

BISECTION: Partition the vertices of an undirected graph into two equal halves so as to minimize the number of edges with exactly one endpoint in each half. Some algorithms, for example using eigenvalues [B87] or simulated annealing [JS93] do well on certain random graphs (see also [BCLS84]). For worst-case inputs, no true approximation algorithms are known. Some known "bisection approximators" (based upon techniques of [LR88]) yield separators whose capacity is within a factor $O(\log n)$ of the capacity of the optimum bisection. Our algorithm gives an exact bisection.

MAX-k-SAT: Given a conjunctive normal form formula with k variables per clause, find a true-false assignment to the variables making the maximum possible number of clauses true. An algorithm in [Yan92] achieves an approximation ratio of 1.33 for the problem. Improved algorithms have since been given for MAX-3-SAT; an approximation ratio of $8/7 + \epsilon$ is achieved in [KZ97]. It also known that achieving an approximation ratio of $8/7 - \epsilon$ is N P-hard [H97].

MIN-k-CUT: Given an n-vertex graph, remove a minimum set of edges that partitions the graph into k connected components. Saran and Vazirani [SV91] gave a $(2-2/k)$-approximation algorithm. The variant *k-terminal cut* problem specifies k vertices that must all be disconnected from each other by the removal of the edges. Dalhaus et al. [DJP+94] give an algorithm that achieves an approximation ratio of $(2 - 2/k)$.

DENSE-k-SUBGRAPH: Given a graph, find a subset of k vertices that induces a graph with the most edges. This problem was studied in [KP93], where

an approximation algorithm with ratio $n^{7/18}$ was presented.

3-COLORING: Color the vertices of a graph with 3 colors such that no two adjacent vertices have the same color. Application of our techniques to this problem yields a result already shown in [Edw86].

MAX-$\mathcal{SNP}$: The class of "constant factor approximable" problems defined in [PY91].

We now define a natural notion of *dense instance* for each problem. (The definition of dense instances for the class *MAX-$\mathcal{SNP}$* appears in Section 4.4, where we also describe a PTAS for them.) Exact optimization on dense instances is $\mathcal{NP}$-hard for all problems except MIN-k-CUT and 3-COLORING (see Section 7).

Definition 1.1. A graph is *δ-dense* if it has $\delta n^2/2$ edges. It is *everywhere-δ-dense* if the minimum degree is δn. We abbreviate $\Omega(1)$-dense as *dense* and everywhere-$\Omega(1)$-dense as *everywhere-dense.* Thus everywhere-dense implies dense, but not vice versa. Similarly, a k-SAT formula is dense if it has $\Omega(n^k)$ clauses, and a dimension-d hypergraph if it has $\Omega(n^d)$ edges.

Theorem 1.2. *There are PTASs for everywhere-dense instances of BISECTION and SEPARATOR.*

Theorem 1.3. *There are PTASs for dense instances of the following problems: MAX-CUT, MAX-DICUT, MAX-k-SAT for any constant k, DENSE-k-SUBGRAPH for $k = \Omega(n)$, MAX-HYPERCUT(d) for constant d, and any MAX-$\mathcal{SNP}$ problem.*

Theorem 1.4. *Exact algorithms exist on everywhere-dense graphs for MIN-k-CUT when $k = o(n)$ and for 3-COLORING.*

Remark. The 3-COLORING result is not new—see [Edw86]—but does follow from a direct application of our general technique

1.2 Our Methods

Our heuristics are based upon two main ideas: *exhaustive sampling* and its use in approximation of *polynomial integer programs.* We discuss these ideas in the

context of the maximum cut problem (MAX-CUT), one of the problems to which our techniques apply.

The goal in MAX-CUT is to partition the vertices of a given graph into two groups—called the *left* and *right* sides—so as to maximize the number of edges with an endpoint on each side. Notice that in the optimum solution, every vertex has the majority of its neighbors on the opposite side of the partition (else, it would improve the cut to move the vertex to the other side). Thus, if we knew where the neighbors of each vertex lay, we would know where to put each vertex.

This argument may seem circular, but the circularity can be broken (in dense graphs) by the following *exhaustive sampling* approach. Suppose we take a sample of $O(\log n)$ vertices. By exhaustively trying all possible (i.e., $2^{O(\log n)}$) placements of the vertices in the sample, we will eventually guess where each vertex of the sample belongs in the optimum cut. Since there are $2^{O(\log n)} = n^{O(1)}$ possibilities, we can afford to try every one of them in polynomial time. So assume we have partitioned the sampled vertices correctly according to the optimal cut. Now consider some unsampled vertex. If a majority of its neighbors belong on the right side of the optimum cut, then we expect that a majority of its sampled neighbors will be from the right side of the optimum cut. This suggests the following scheme: put each unsampled vertex on the side opposite the majority of its sampled neighbors.

This scheme works well for vertices whose opposite-side neighbors significantly outnumber their same-side neighbors. More problematic are vertices for which the neighbors split evenly between the two sides; sampling will not typically give us confidence about the majority side. This brings us to the second major idea of our paper: approximately solving nonlinear integer programs. Define a variable x_i for vertex i which is 1 if the vertex is on the right side of a cut and 0 otherwise. Then finding a maximum cut corresponds to finding a 0-1 assignment that maximizes the following function (where E is the edge set of the graph):

$$\sum_i x_i \left(\sum_{(i,j) \in E} (1 - x_j) \right).$$

To see this, note that the formula counts, for every vertex i on the right side of the cut, the number of edges leading from it to neighbors j on the left side of the cut.

Of course, solving even *linear* integer programs is N P-complete, and the above program involves a *quadratic* objective function. However, we show that exhaustive sampling can be used to approximately maximize such functions, and more generally, to approximately solve integer programs in which the constraints and objective involve low-degree polynomials instead of linear functions. We state our main approximation result in the next section.

Most of our approximation algorithms are more properly viewed as algorithms that compute an *additive* approximation (see Section 1.3). For example, our algorithm for MAX-CUT computes, for *every* graph, a cut of capacity at least $OPT - \epsilon n^2$, where ϵ is any desired constant. Such an approximation is also within a small *multiplicative* factor of the optimum in a dense graph (i.e., one with $\Omega(n^2)$ edges) because $OPT = \Omega(n^2)$ for such graphs (this follows from our earlier observation that in an optimum cut, every vertex is on the opposite side from a majority of its neighbors). However, our algorithms for BISECTION and SEPARATOR are not additive approximation algorithms.

1.3 Smooth Integer Programs

Many existing approximation algorithms for N P-hard problems are based on representation of the problem as a linear integer program (LIP). All problems in N P have such formulations since solving LIPs is N P-complete. Many problems have natural formuations as LIPs that give insight into their structure and lead to approximation algorithms. But formulation as a LIP masks the true nature of many other problems—in particular, an approximately optimum solution to the LIP may correspond to a far from optimum solution to the original optimization problem. A more natural formulation involves *nonlinear* integer program in which the objective function is a low degree polynomial. Most of our PTAS's for dense problems are derived from such a representation. We solve a general class of optimization problems in which the objective function and the constraints are polynomials.

Definition 1.5. A *polynomial integer program* (or PIP) is of the form

$$\text{maximize} \quad p_0(x_1, \ldots, x_n) \tag{1}$$

$$\text{subject to} \quad l_i \leq p_i(x) \leq u_i \quad (i = 1, \ldots, m) \tag{2}$$

$$x_i \in \{0, 1\} \qquad \forall i \leq n \tag{3}$$

where $p_0, \ldots, p_m$ are polynomials. (The PIP could involve minimization instead of maximization.)

When all p_i have degree at most d, we call this program a *degree d PIP*.

Since they subsume integer programs, it is clear that solving PIPs is N P-hard. One might hope to define a more tractable class by eliminating the integrality requirement, but this accomplishes nothing since the 0–1 integrality of x_i can be enforced by the quadratic constraint $x_i(x_i - 1) = 0$.

We now describe a class of PIPs that are easy to approximate.

Definition 1.6. An n-variate, degree-d polynomial has *smoothness c* if the absolute value of each coefficient of each degree i monomial (term) is at most $c \cdot n^{d-i}$.

Remark. The reader should think of c and d as being fixed constants, and n as being allowed to grow. We call the resulting family of polynomials a family of *c-smooth degree d polynomials.*

Definition 1.7. A *c-smooth degree-d PIP* is a PIP in which the objective function and constraints are c-smooth polynomials with degree at most d.

Smooth integer programs can represent many combinatorial problems in a natural way. We illustrate this using MAX-CUT as an example.

Example 1.8. A degree-2 polynomial with smoothness c has the form

$$\sum a_{ij} x_i x_j + \sum b_i x_i + d$$

where each $|a_{ij}| \leq c, |b_i| \leq cn, |d| \leq cn^2$.

We show how to formulate MAX-CUT on the graph $G = (V, E)$ using a 2-smooth integer program. Define a variable x_i for each vertex v_i. Then, assign $0, 1$ values to the x_i (in other words, find a cut) so as to maximize

$$\sum_{\{i,j\} \in E} (x_i(1 - x_j) + x_j(1 - x_i)).$$

(Notice that an edge $\{i,j\}$ contributes 1 to the sum when $x_i \neq x_j$ and 0 otherwise. Thus the sum is equal to the cut value.) Expanding the sum shows that the coefficients of the quadratic terms are 0 and -2 while the coefficients of the linear terms are at most n.

Now we can state our general theorem about approximation of smooth integer programs.

Definition 1.9. A solution a is said to *satisfy* a constraint $l_i \leq p_i(x) \leq u_i$ *to within an additive error* δ if $l_i - \delta \leq p_i(a) \leq u_i + \delta$.

Theorem 1.10. *There is a randomized polynomial-time algorithm that approximately solves smooth PIPs, in the following sense. Given a feasible c-smooth degree d PIP with n variables, objective function p_0 and K constraints, the algorithm finds a 0/1 solution z satisfying*

$$p_0(z_1, \ldots, z_n) \geq OPT - \epsilon n^d,$$

where OPT is the optimum of the PIP. This solution z also satisfies each degree d' constraint to within an additive factor of $\epsilon n^{d'}$ for $d' > 1$, and satisfies each linear constraint to within an additive error of $O(\epsilon\sqrt{n \log n})$.

The running time of the algorithm is $O((dKn^d)^t)$, where $t = 4c^2e^2d^2/\epsilon^2 = O(1/\epsilon^2)$.

The algorithm can be derandomized (i.e., made deterministic), while increasing the running time by only a polynomial factor.

Remark. The statement of the theorem can be stronger: the input PIP does not need to be feasible, but only approximately feasible (that is, there must be a point that satisfies each degree d' constraint to within an additive error $\epsilon' n^{d'}$ for some $\epsilon' < \epsilon/2$.)

Theorem 1.10 underlies almost all of our PTASs. However, our PTASs for BISECTION and MIN-k-CUT require some additional ideas since an additive approximation is not good enough.

1.4 Related Work

There are known examples of problems that are seemingly easier to approximate in dense graphs than in general graphs. For instance, in graphs with degree exceeding $n/2$, one can find Hamiltonian cycles [P̀76] and approximate the number of perfect matchings [JS89]. In everywhere-dense graphs it is easy to approximate the values of the Tutte polynomial and, as a special case, to estimate the reliability of a network [AFW94].

Independent of our work, Fernandez de la Vega [FdlV94] developed a PTAS for everywhere-dense MAX-CUT using exhaustive sampling principles similar to ours. After sampling and guessing, Fernandez de la Vega replaces our linear-programming solution with a greedy placement procedure. While this procedure is significantly simpler than ours (at least conceptually; the running time is still dominated by the exhaustive sampling procedure and is similar to ours), it is not obvious (and is an interesting open question) whether the procedure can generalize to the other problems we have listed.

Edwards [Edw86] shows how to 3-color a 3-colorable everywhere-dense graph in polynomial time. Our sampling approach gives an alternative algorithm.

A random-sampling based approach related to ours also appears in [KP92].

In the last section of the paper (Section 8) we describe some results related to our work that have been discovered since the conference presentation of the current paper.

1.5 Paper Organization

In Section 2 we give details of the main ideas of our approach, *exhaustive sampling* and *transforming polynomial constraints into linear constraints*, already sketched in Sections 1.2 and 1.3. We continue to use MAX-CUT as a motivating example.

In Section 3 we generalize these techniques to derive our (additive error) approximation algorithm for *any* smooth polynomial integer program (PIP). In Section 4, we use these PIPs to approximate most of the problems listed in Section 1.1. Solving BISECTION and SEPARATOR requires some additional exhaustive sampling ideas that are explained in Section 5. In Section 6, we

describe some problems that can be solved purely by exhaustive sampling, with no recourse to PIPs. Finally, in Section 7, we confirm that all of the problems we are approximating are still N P-complete when restricted to dense instances, demonstrating that an exact solution is unlikely.

2 Our Techniques: An overview

In this section we introduce our two major techniques, *exhaustive sampling* and *reducing degree d constraints to linear constraints* (in an approximate sense) to give a PTAS for dense MAX-CUT.

First we express MAX-CUT as a quadratic integer program as follows. Let the 0/1 vector x be the characteristic vector of a cut, i.e., $x_i = 1$ iff i is on the right side. Let $N(i)$ be the set of neighbors of vertex i, and let

$$r_i(x) = \sum_{j \in N(i)} (1 - x_j)$$

be the linear function denoting the number of number of neighbors of i that are on the left side of cut x. Then

$$\begin{aligned} \text{MAX-CUT} \;=\; & \max \textstyle\sum_i x_i \cdot r_i(x) \\ & \text{s.t. } x_i \in \{0,1\} \qquad \forall i \end{aligned}$$

The above formulation looks a lot like an integer linear program, for which numerous approximation techniques are known. Unfortunately, the "coefficients" $r_i(x)$ in the objective function are not constants—the program is actually a *quadratic* program. However, exhaustive sampling lets us estimate the value these coefficients have in the optimum solution. We arrive at our approximation in three steps:

1. Using exhaustive sampling, we estimate the values of $r_i(a)$ at the optimum solution $a = (a_1, \ldots, a_n)$. See Section 2.1.

2. We replace each function r_i by the corresponding estimate of $r_i(a)$. This turns the quadratic program into a linear (integer) program. We show that optimum of this linear integer program is near-optimum for the quadratic program. See Section 2.2.

3. We solve the fractional relaxation of the linear integer program, and use *randomized rounding* to convert the solution into an integer one. We show that this does not dramatically change the solution value. See Section 2.3.

A comment on notation: Throughout the paper, we will use $a \pm b$ where a and b are real numbers, as a shorthand for the interval $[a-b, a+b]$.

2.1 Estimating Coefficients

We begin by using exhaustive sampling to estimate the values $r_i(a)$ at the optimum solution a. Let a be the optimum cut and let $\rho_i = r_i(a)$. Then a is the solution to the following integer linear program:

$$\begin{aligned} \text{MAX-CUT} = \quad & \max \textstyle\sum_i x_i \cdot \rho_i \\ & \text{s.t. } x_i \in \{0,1\} \qquad \forall i \\ & r_i(x) = \rho_i \qquad \forall i \end{aligned}$$

Of course, the usefulness of this observation is unclear, since we don't know the values ρ_i. We show, however, that it is possible to compute an additive error *estimate* of the ρ_i in polynomial time, in other words, a set of numbers e_i such that

$$\rho_i - \epsilon n \leq e_i \leq \rho_i + \epsilon n \qquad \forall i. \tag{4}$$

This can be done using our *exhaustive sampling* approach. We take a random sample of $O(\log n)$ vertices. By exhaustively trying all possible (i.e., $2^{O(\log n)} = n^{O(1)}$) placements of the vertices in the sample, we will eventually guess a placement wherein each vertex is placed as it would be in the optimum cut. So we can assume that we have "guessed" the values a_j in the optimum cut for all the sampled vertices j. Now consider any unsampled vertex i. If it has $|N(i)| = \Omega(n)$ neighbors, then with high probability, $\Theta(\log n)$ of its neighbors are part of the random sample (*high probability* means probability $1 - n^{-\Omega(1)}$). A moment's thought shows that these neighbors form a uniform random sample from $N(i)$. Hence by examining the fraction of sampled neighbors on the left hand

side of the cut (namely, neighbors for which $a_j = 0$) we can obtain an estimate of $r_i(a)/|N(i)|$ that is correct to within a small additive factor. This follows from the following sampling lemma.

Lemma 2.1 (Sampling Lemma). *Let (a_i) be a sequence of n numbers, each with absolute value at most M. Let $f > 0$ be any number. If we choose a multiset of $s = g \log n$ of the a_i at random (with replacement), then their sum q satisfies*

$$\frac{qn}{s} \in \sum_i a_i \pm nM\sqrt{\frac{f}{g}}$$

with probability at least $1 - n^{-f}$.

Proof. Let $s = g \log n$. For $j = 1, \ldots, s$ let X_j be the random variable denoting the number picked in the j^{th} draw. Since the numbers are drawn with replacement, the values X_j are identically distributed, and

$$E[X_j] = \frac{1}{n}\sum_{j=1}^{n} a_j.$$

Since $|X_j| \leq M$ by hypothesis, the lemma now follows from the standard Höffding bound [H64]. □

For MAX-CUT, our goal is to estimate the values ρ_i of the form $\sum_{j \in N(i)}(1 - a_j)$. First, if any $|N(i)| \leq \epsilon n/10$, we use the estimate 0 for ρ_i. To estimate ρ_i for the remaining i, we randomly choose (with replacement) $g \log n$ indices with $g = O(1/\epsilon^3)$, and "guess" their values by exhaustively trying all possibilities. Since each $a_j = 0$ or 1, we can take $M = 1$ in the Sampling Lemma. The Sampling Lemma shows that for each i, the probability is at least $1 - 1/n^2$ that the following happen (i) $\Omega(\log n/\epsilon^2)$ of the sampled vertices lie in $N(i)$ (note that the conditional distribution of these vertices is uniform) and (ii) the estimate for ρ_i produced using this sample is accurate to within ϵn.

2.2 Linearizing the Quadratic Integer Program

Now we use the coefficient estimates to define an integer linear program whose solutions are near-optima for MAX-CUT. Given the estimates e_i just derived

for the values ρ_i, we write the following linear integer program. Note that it is feasible, since a satisfies it (assuming our sampling step in the previous section worked).

$$\begin{aligned} \text{NEW-OPT} = \quad & \max \ \textstyle\sum_i x_i \cdot e_i \\ & \text{s.t. } x_i \in \{0,1\} \qquad \forall i \\ & e_i - \epsilon n \leq r_i(x) \leq e_i - \epsilon n \qquad \forall i \end{aligned} \tag{5}$$

(Recall that each $r_i(x)$ is a linear function of x, so the given constraints are linear constraints.)

We claim that the optimum solution z to this integer linear program is near-optimum for MAX-CUT. This can be seen as follows:

$$\begin{aligned} \sum z_i r_i(z) &\geq \sum z_i(e_i - \epsilon n) && \text{By the constraints (5)} \\ &\geq \sum z_i e_i - \epsilon n^2 \\ &\geq \sum a_i e_i - \epsilon n^2 && \text{Since } z \text{ is integer optimum} \\ &\geq \sum a_i(\rho_i - \epsilon n) - \epsilon n^2 && \text{from (4)} \\ &\geq \sum a_i \rho_i - 2\epsilon n^2 \\ &= \text{MAX-CUT} - 2\epsilon n^2 \end{aligned}$$

In other words, the optimum of the integer program is a near-optimum solution to MAX-CUT.

2.3 Approximating the Linear Integer Program

Of course, we cannot exactly solve the integer linear program just derived. But we can compute an approximate solution to it as follows. We relax the integrality constraints, allowing $0 \leq x_i \leq 1$. We use linear programming to obtain the fractional optimum, say $y \in [0,1]^n$, and then use *randomized rounding* to convert the fractional solution to an integral one of roughly the same value. The key lemma is the following:

Lemma 2.2 (Randomized Rounding). *If c and f are positive integers and $0 < \epsilon < 1$, then the following is true for any integers $n \geq 0$. Let $y = (y_i)$ be a vector of n variables, $0 \leq y_i \leq 1$, that satisfies a certain linear constraint $a^T y = b$, where each $|a_i| \leq c$. Construct a vector $z = (z_i)$ randomly by setting $z_i = 1$ with probability y_i and 0 with probability $1 - y_i$. Then with probability at least $1 - n^{-f}$, we have*

$$a^T z \in b \pm c\sqrt{fn \ln n}$$

We can apply this lemma to our problem as follows. Give our fractional solution y, let us apply randomized rounding as in the lemma to yield an integral solution z. We claim that with high probability,

$$r_i(z) \in r_i(y) \pm O(\sqrt{n} \ln n) \tag{6}$$

$$\sum z_i r_i(y) \in \sum y_i r_i(y) \pm O(n^{3/2} \ln n) \tag{7}$$

Specifically, to derive Equations (6) and (7) from Lemma 2.2, note that each $r_i(x)$ is a linear function with 0–1 coefficients and that each $r_i(y)$ is at most n.

We use these equations as follows. The analysis of the previous section showed that the integral optimum of our derived linear program was near the maximum cut value, so the fractional optimum y can only be better. That is,

$$\sum y_i r_i(y) \geq \text{MAX-CUT} - 2\epsilon n^2.$$

We now use our randomized rounding lemma. We have that

$$\begin{aligned}
\sum z_i r_i(z) &\geq \sum z_i (r_i(y) - O(\sqrt{n} \ln n)) && \text{From (6)} \\
&\geq \sum z_i r_i(y) - O(n^{3/2} \ln n) \\
&\geq \sum y_i r_i(y) - O(n^{3/2} \ln n) && \text{From (7)} \\
&\geq \text{MAX-CUT} - (2\epsilon + o(1))n^2
\end{aligned}$$

This finishes the overview of our algorithm for MAX-CUT.

3 Approximating Smooth Integer Programs

We now generalize the results of the previous section to handle arbitrary polynomial integer programs (PIPs). We describe an algorithm that computes ap-

proximate solutions to smooth PIPs of low degree, thus proving Theorem 1.10. We use the fact that smooth PIPs can be recursively decomposed into smooth lower-degree PIPs. This lets us apply ideas similar to those described in Section 2 for MAX-CUT. In a PIP the objective function and constraints are low degree polynomials (degree 2 in the case of MAX-CUT). We use exhaustive sampling to convert such polynomial integer programs into linear integer programs. Then we use the Raghavan-Thompson technique to approximately solve the linear integer program.

We will see shortly that we can assume without loss of generality that we are dealing with the feasibility version of a PIP—that is, we are given a feasible PIP and out goal is to find an approximately feasible integer solution. Our general algorithm has the same three elements as the one for MAX-CUT:

1. We show in Section 3.2 that we can relax the integrality conditions, since we can use randomized rounding to convert every feasible *fractional* solution of a PIP into a feasible integral solution.

2. In Section 3.3, we generalize the sampling theorem, which applies only to sums, to let us estimate the values of polynomials.

3. We show in Section 3.4 that we can use our estimates to convert degree d constraints into linear constraints without affecting feasibility.

We begin in Section 3.1 with some basic observations.

3.1 Basic Observations

We begin with a few basic observations that we will use at various times in the proof.

3.1.1 A Polynomial Decomposition

Our PIP algorithms are basically recursive generalizations of the approach for MAX-CUT. They rely on the following key observation that lets us decompose any polynomial into simpler polynomials:

Lemma 3.1. *A c-smooth polynomial p of degree d can be written as*

$$p(x) = t + \sum x_i p_i(x)$$

where t is a constant and each p_i is a c-smooth polynomial of degree $d-1$.

Proof. From each monomial term in the expansion of p, pull out one variable x_i. Group all monomials from which x_i was extracted into p_i. Every degree d' term in p_i corresponds to a degree $d'+1$ term in p, and thus has coefficient at most $cn^{d-(d'+1)} = cn^{(d-1)-d'}$. Thus, since p_i has degree (at most) $d-1$, it is a c-smooth degree $d-1$ polynomial. □

Remark. The above analysis also shows that we can express p *uniquely* as a sum

$$p(x) = t + \sum x_i p_i(x_i, \ldots, x_n),$$

that is, where each p_i depends only on variables with index i or greater.

The decomposition of a degree d polynomial into degree $d-1$ polynomials gives us a natural recursion through which we can generalize our quadratic programming techniques. By computing an estimate ρ_i of the value of $p_i(x)$ at the optimum solution, we replace the degree-d constraint p with a single constraint on $\sum x_i \rho_i$ together with a family of constraints on the values $p_i(x)$. We then recursively expand these degree $d-1$ constraints, continuing until all of our constraints are linear.

To estimate the values $p_i(x)$, we again rely on the expansion above: we expand p_i in terms of degree $d-2$ polynomials, writing $p_i(x) = \sum x_j p_{ij}(x)$, recursively estimate the p_{ij} values, and then use exhaustive sampling to estimate p based on the values of the p_{ij}.

After constructing the required linear integer program, we solve its fractional relaxation and use randomized rounding as before to transform the solution into an integral solution. To prove that randomized rounding works, we again use the decomposition—we show that each $p_i(x)$ is roughly preserved by rounding, and deduce that $\sum x_i p_i(x)$ is also preserved.

3.1.2 Reducing Optimization to Feasibility

We can reduce PIP optimization to the corresponding feasibility problem ("Is there a feasible solution such that the objective exceeds a given value?") using binary search in the usual way. This uses the fact that the optimum value of a PIP is not too large, as shown in the following lemma (which will also be useful later).

Lemma 3.2. *If $n > d$, then the absolute value of a c-smooth polynomial at any point in $[0,1]^n$ is at most $2cen^d$ (where $\ln e = 1$).*

Proof. For $0 \leq i \leq d$ the polynomial has at most $\binom{n+i}{i}$ terms of degree i, and each has a coefficient in $[-cn^{d-i}, cn^{d-i}]$. Thus an upper bound on the absolute value at any point in $[0,1]^n$ is

$$\begin{aligned}
\sum_{i=0}^{d} cn^{d-i} \cdot \binom{n+i}{i} &\leq \sum_{i=0}^{d} cn^{d-i} \cdot \binom{n+d}{i} \\
&\leq cn^d \sum_{i=0}^{d} \left(\frac{n+d}{n}\right)^i \frac{1}{i\,!} \\
&\leq cn^d e^{1+d/n}
\end{aligned}$$

which is at most $cen^d(1 + 2d/n) < 2cen^d$ for $n > 5d$. □

3.2 Rounding Fractional PIPs

We begin with the final step of our algorithm, rounding a fractional solution to an integral one. We present this section first since it is more straightforward than the following ones but conveys the same ideas. As we saw in Section 2.3, Raghavan and Thompson [RT87] show that given a fractional solution to a linear program, we can round it into an integer solution that is "almost as good." We rephrased their result in Lemma 2.2. We now modify the Raghavan-Thompson technique to show in Lemma 3.3 that a similar result is true for low degree polynomials. In other words, we show that the value of a c-smooth polynomial at a point in $[0,1]^n$ is not too different from its value at a nearby integral point obtained by randomized rounding.

Lemma 3.3 (Randomized Rounding for degree d polynomials). *Let p be a c-smooth degree-d polynomial. Given fractional values (y_i) such that $p(y_1, \ldots, y_n) = b$, suppose randomized rounding is performed on the y_i as in Lemma 2.2 to yield a $0,1$ vector (z_i). Then with probability at least $1 - n^{d-f}$, we have*

$$p(z_1, \ldots, z_n) \in \left[b \pm gdn^{d-\frac{1}{2}} \sqrt{\ln n} \right],$$

where $g = 2ce\sqrt{f}$.

Proof. We use induction on the degree. The case $d = 1$ follows from Lemma 2.2. Now assume we have proved the Lemma for all integers less than d, and p is a degree d polynomial. As argued in Section 3.1, we can express p as

$$p(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i \cdot p_i(x_1, \ldots, x_n) + t, \tag{8}$$

where t is a constant and p_i is a c-smooth polynomial of degree at most $d - 1$.

Let ρ_i denote the value $p_i(y_1, \ldots, y_n)$. Then

$$\begin{aligned} b &= p(y_1, \ldots, y_n) \\ &= t + \sum_{i=1}^{n} \rho_i \cdot y_i. \end{aligned}$$

Let $(z_1, \ldots, z_n) \in \{0,1\}^n$ be obtained by randomized rounding on $(y_1, \ldots, y_n)$. Our proof consists of noticing that with high probability, (z_i) satisfies both $b \approx \sum_i \rho_i z_i$ (by Lemma 2.2) and $\forall i \leq n : \rho_i \approx p_i(z_1, \ldots, z_n)$ (by induction for degree $d - 1$). Then we realize that any such (z_i) also satisfies $b \approx p(z_1, \ldots, z_n)$.

Let us formalize this idea. Note that $|\rho_i| \leq 2cen^{d-1}$ by Lemma 3.2. So we can apply Lemma 2.2 (replacing c by $2cen^{d-1}$). We find that with probability at least $1 - n^{-f}$ (recalling that the notation $a \pm b$ is shorthand for the interval $[a - b, a + b]$),

$$\sum_{i=1}^{n} \rho_i \cdot z_i \in b \pm gn^{d-1} \sqrt{n \ln n}. \tag{9}$$

Furthermore, the inductive hypothesis implies that for each $i \leq n$, the probability is at least $1 - n^{d-f-1}$ that

$$p_i(z_i, \ldots, z_n) \in \rho_i \pm g(d-1)n^{d-1-\frac{1}{2}}\sqrt{\ln n} \tag{10}$$

Hence we conclude that with probability at least $1 - n^{d-f} - n^{-f} \approx 1 - n^{d-f}$, the event mentioned in Condition (10) happens for each $i \leq n$, and so does the event mentioned in Condition (9). Of course, when all these events happen, we have:

$$\begin{aligned}
p(z_1, \ldots, z_n) &= t + \sum_{i=1}^{n} z_i \cdot p_i(z_1, \ldots, z_n) \\
&\in t + \sum z_i(\rho_i \pm g(d-1)n^{d-1-1/2}\sqrt{\ln n}) && \text{by (10)} \\
&\subseteq t + \sum z_i\rho_i \pm g(d-1)n^{d-1/2}\sqrt{\ln n} \\
&= b \pm gn^{d-1}\sqrt{n \ln n} \pm g(d-1)n^{d-1/2}\sqrt{\ln n} && \text{by (9)} \\
&= [b \pm gdn^{d-\frac{1}{2}}\sqrt{\ln n}]
\end{aligned}$$

Hence we have shown that $p(z_1, \ldots, z_n) \in [b \pm gdn^{d-\frac{1}{2}}\sqrt{\ln n}]$ with probability at least $1 - n^{d-f}$. □

3.3 Estimating the Value of a Polynomial

Having shown how to round a fractional solution to an integral one, we now show how to find an approximately optimal fractional solution by solving a linear program. As discussed above, our procedure for replacing the constraint on $p(x)$ by linear constraints requires estimating the values at the optimum a of the coefficients $p_i(a)$ in the expansion $p(a) = \sum a_i p_i(a)$. In this section, we show how this estimation can be accomplished by exhaustive sampling. We describe a procedure `Eval` in Figure 3.3 that can approximate the value of a c-smooth degree d polynomial $p(x_1, \ldots, x_n)$ on any unknown 0/1 vector $(a_1, \ldots, a_n)$, given only partial information about $(a_1, \ldots, a_n)$. The algorithm is given the values a_i for $O(\log n)$ randomly-chosen indices i, and outputs an estimate that, with high probability, lies in $p(a_1, \ldots, a_n) \pm \epsilon n^d$.

To simplify our exposition later we describe the procedure more generally as using a (multi)set of indices $S \subseteq \{1, \ldots, n\}$.

Algorithm `Eval`$(p, S, \{a_i : i \in S\})$

Input: polynomial p of degree at most d,
set of variables indices S,
a_i for $i \in S$.
Output: estimate for $p(a_1, \ldots, a_n)$.

if $deg(p) = 0$ (i.e., p is a constant) **then**
    return p
else
    write $p(x_1, \ldots, x_n) = t + \sum x_i p_i(x_1, \ldots, x_n)$
        where t is a constant and each p_i has degree at most $d-1$
    for each $i \in S$
        $e_i \leftarrow$`Eval`$(p_i, S, \{a_i : i \in S\})$
    return
        $$t + \frac{n}{|S|} \sum_{i \in S} a_i e_i$$

Figure 1: The approximate evaluation algorithm

Note that if $S = \{1, \ldots, n\}$ then the procedure returns $p(a_1, \ldots, a_n)$. We will show that in order to get an additive approximation of the type we are interested in, it suffices to choose S randomly and of size $O(\log n)$. We use the Sampling Lemma (2.1) as the base case in our inductive proof of the correctness of `Eval`.

Lemma 3.4. *Let p be a c-smooth polynomial of degree d in n variables x_i, and let $a_1, \ldots, a_n \in \{0, 1\}$. Let S be a set of $O(g \log n)$ indices chosen randomly (with replacement). Then with probability at least $1 - n^{d-f}$, set S is such that* `Eval`$(p, S, \{a_i : i \in S\})$ *returns a value in $p(a_1, \ldots, a_n) \pm \epsilon n^d$, where*

$$\epsilon = 4ce\sqrt{\frac{f}{g}}.$$

Proof. The proof is by induction on d. The case $d = 0$ is clear. For the inductive step let $\rho_i = p_i(a_i, \ldots, a_n)$, so we have

$$p(a_1, \ldots, a_n) = t + \sum_{i=1}^{n} a_i \cdot \rho_i \tag{11}$$

The intuition for why `Eval`'s output should approximate $p(a_1, \ldots, a_n)$ is as follows. Each p_i has degree at most $d - 1$, so the inductive hypothesis implies that $e_i \approx \rho_i$. Thus the output of eval is

$$\begin{aligned} t + \frac{n}{|S|} \cdot \sum_{i \in S} a_i \cdot e_i &\approx t + \frac{n}{|S|} \cdot \sum_{i \in S} a_i \cdot \rho_i && \text{(by the inductive hypothesis)} \\ &\approx t + \sum_{i} a_i \cdot \rho_i && \text{(by the Sampling Lemma)} \end{aligned}$$

It remains to fill in the details, and to deal with the complication that the errors in our recursive estimates of the ρ_i accumulate into the error for our estimate of $p(a_1, \ldots, a_n)$.

Our sample has size $g \log n$. By Lemma 3.2, each $|\rho_i| \leq 2ce n^{d-1}$. Hence the Sampling Lemma implies that with probability $1 - n^{-f}$ the set S is such that

$$\frac{n}{|S|} \sum_{i \in S} a_i \rho_i \in \sum_{i} a_i \rho_i \pm \left(2ce\sqrt{\frac{f}{g}}\right) n^d \tag{12}$$

Of course, we do not have the values ρ_i. However, we do have the values $e_i =$`Eval`$(p_i, S, \{a_i : i \in S\})$. To see the impact of using them instead, let ϵ_d

denote the smallest number such that for every c-smooth degree d polynomial p and point $a \in \{0,1\}^n$

$$\Pr[\texttt{Eval} \text{ computes an estimate within } p(a) \pm \epsilon_d n^d] \geq 1 - n^{d-f}$$

We get an recurrence for ϵ_d as follows. By definition, `Eval` estimates any particular ρ_i to within $\epsilon_{d-1} n^{d-1}$ with probability $1-n^{(d-1)-f}$. Thus all n values ρ_i are estimated to within this bound with probability $1-n\cdot n^{(d-1)-f} = 1-n^{d-f}$. Combining with (12), we conclude that with probability at least $1 - n^{d-f} - n^{-f} \approx 1 - n^{d-f}$, set S is such that the returned value

$$\begin{aligned}
t + \frac{n}{|S|}\sum_{i\in S} a_i \cdot e_i &\in t + \frac{n}{|S|}\sum_{i\in S} a_i \cdot \left(\rho_i \pm \epsilon_{d-1} n^{d-1}\right) \\
&\subseteq t + \left(\frac{n}{|S|}\sum_{i\in S} a_i \cdot \rho_i\right) \pm \epsilon_{d-1} n^{d-1}\frac{n}{|S|} \\
&\subseteq t + \left(\sum_i a_i \cdot \rho_i \pm 2ce\sqrt{\frac{f}{g}}n^d\right) \pm \frac{\epsilon_{d-1}}{|S|}n^d \qquad \text{by (12)} \\
&\subseteq t + \sum_i a_i \cdot \rho_i \pm \left(2ce\sqrt{\frac{f}{g}} + \frac{\epsilon_{d-1}}{|S|}\right) n^d \\
&= p(a_1, \ldots, a_n) \pm \left(2ce\sqrt{\frac{f}{g}} + \frac{\epsilon_{d-1}}{|S|}\right) n^d.
\end{aligned}$$

It follows that

$$\begin{aligned}
\epsilon_d &\leq 2ce\sqrt{\frac{f}{g}} + \frac{\epsilon_{d-1}}{|S|} \\
&\leq 2ce\sqrt{\frac{f}{g}}(1 + |S|^{-1} + \cdots + |S|^{-d}) \\
&\leq 4ce\sqrt{\frac{f}{g}}
\end{aligned}$$

for $|S| > 1$. □

Corollary 3.5. *With probability* $1-n^{d-f}$ *over the choice of* S*, the* `Eval` *procedure accurately estimates the values of* all *the polynomials arising from the decomposition of polynomial* p *(that is, estimates every degree-*d' *polynomial to within* $\epsilon_{d'} n^{d'}$*).*

Proof. This is implicit in the previous proof. Note that the decomposition of p is determined solely by p, independent of the value of the optimum solution a that we are estimating. □

3.4 Transforming Degree d Constraints to Linear Constraints

Using the estimates produced by Procedure `Eval` of Section 3.3 we can transform any polynomial constraint into a family of linear constraints, so that any feasible solution to the linear constraints will approximately satisfy the polynomial constraint as well. We use algorithm `Linearize` in Figure 3.4. Just like `Eval`, the inputs to this procedure contain partial information about some feasible solution vector $(a_1, \ldots, a_n) \in \{0,1\}^n$ to the input constraint.

A simple induction shows that the Procedure in Figure 3.4, when given a degree d constraint, outputs a set of at most $1 + n + \cdots + n^{d-1} = O(n^{d-1})$ linear constraints. The next two lemmas prove the correctness of this (probabilistic) reduction. The first shows that with high probability, the replacement equations are jointly feasible. The second shows any feasible solution will be almost feasible for the original constraint.

Lemma 3.6. *Let $f, g, c > 0$ be any constants. Let* `Linearize` *be given an error parameter $\epsilon = 4ce\sqrt{f/g}$ and a constraint involving a c-smooth polynomial of degree d. Let $(a_1, \ldots, a_n) \in \{0,1\}^n$ be a feasible solution to the constraint. If S is a random sample of $g \log n$ variables (picked with replacement), then with probability at least $1 - dn^{d-f}$ Procedure* `Linearize` *outputs a set of linear constraints that are satisfied by $(a_1, \ldots, a_n)$.*

Proof. Calling linearize with polynomial p results in numerous recursive call, each of which (besides making other recursive calls) outputs a constraint on some degree d' polynomial p'. The boundaries l_i and u_i for this constraint are determined by a call to `Eval` involving polynomial p'. Vector a satisfies this constraint so long as l_i and u_i satisfy $l_i \leq p(a) \leq u$, and this happens so long as $e_i \in p'(a) \pm \epsilon n^{d'}$. So `Linearize` does the right thing so long as every polynomial p' arising in the recursions is estimated accurately. But these polynomials are just

Algorithm `Linearize`("$L \leq p(x_1, \ldots, x_n) \leq U$", S, $\{a_i : i \in S\}$, ϵ)

Input: constraint involving polynomial p of degree d,
lower bound L and upper bound U
multiset of variable indices S
$a_i \in \{0, 1\}$ for each $i \in S$.
Error parameter $\epsilon > 0$.

Output: A set of linear constraints

if p is linear **then**
  output the input constraint "$L \leq p(x) \leq U$"

else
  write $p(x_1, \ldots, x_n) = t + \sum x_i p_i(x_1, \ldots, x_n)$
    where t is a constant and each p_i has degree at most $d - 1$
  for $i = 1$ to n
    $e_i \leftarrow$ `Eval`$(p_i, S, \{a_i : i \in S\})$
    $l_i \leftarrow e_i - \epsilon n^{d-1}$
    $u_i \leftarrow e_i + \epsilon n^{d-1}$
    `Linearize`("$l_i \leq p_i(x_1, \ldots, x_n) \leq u_i$", S, $\{a_i : i \in S\}$, ϵ)
  output the constraint
    "$L - \epsilon n^d \leq t + \sum x_i e_i \leq U + \epsilon n^d$"

Figure 2: Linearizing a Polynomial Constraint

the polynomials arising in the recursive decomposition of the original polynomial p, and are all encountered during a call to `Eval`(p). Corollary 3.5 says that all of these polynomials are estimated to within the desired bounds with probability $1 - n^{d-f}$. □

Now we show that in the linear system output by `Linearize`, every feasible solution is an approximate solution to the input constraint (note that this statement involves no probabilities).

The next lemma uses $[x, y] \pm a$, where $x < y$ and $a \geq 0$, as a shorthand for the interval $[x - a, y + a]$.

Lemma 3.7. *Every feasible solution $(y_i) \in [0,1]^n$ to the set of linear constraints output by* `Linearize` *satisfies the following (irrespective of what the set S is)*

$$p(y) \in [L, U] \pm 2d\epsilon n^d$$

Proof. By induction on d. The case $d = 1$ is clear, so consider the inductive step. Since by assumption y is feasible for the entire set of output constraints, it is feasible for the constraints output by each recursive call involving a polynomial p_i. It follows by the inductive hypothesis that for each i,

$$p_i(y_i, \ldots, y_n) \in [l_i, u_i] \pm 2(d-1)\epsilon n^{d-1}.$$

Substituting the values of l_i and u_i we get

$$p_i(y) \in e_i \pm (2d-1)\epsilon n^{d-1}$$

Therefore,

$$\begin{aligned} p(y) &= t + \sum y_i p_i(y_i, \ldots, y_n) \\ &\in t + \sum y_i (e_i \pm (2d-1)\epsilon n^{d-1}) \\ &\subseteq t + \left(\sum y_i e_i\right) \pm (2d-1)\epsilon n^d \\ &\subseteq [L, U] \pm \epsilon n^d \pm (2d-1)\epsilon n^d \qquad (13) \\ &\subseteq [L, U] \pm 2d\epsilon n^d \end{aligned}$$

where Equation (13) follows from the fact that y is feasible for the constraint that was output before recursion, namely

$$t + \sum x_i e_i \in [L, U] \pm \epsilon n^d.$$

Thus the inductive step is complete. □

3.5 Proof of the main theorem

The proof of Theorem 1.10 now follows easily. We have a feasible degree d PIP with K constraints, where $K = poly(n)$. Suppose $(a_1, \ldots, a_n)$ is some (unknown) feasible 0/1 solution and $\epsilon > 0$ is some tolerance parameter. We describe a probabilistic procedure that produces a 0/1 solution z that is approximately feasible. That is, if $L \leq p(x) \leq U$ was a degree d' constraint in the input, then with high probability z satisfies it within an additive error $\epsilon n^{d'}$, that is, $L - \epsilon n^{d'} \leq p(z) \leq U + \epsilon n^{d'}$. (We indicate below how to derandomize the procedure.)

Let $f > 0$ be such that $n^f \approx 2dKn^d$. We let $\epsilon' = \epsilon/2d$, and $g = 16c^2e^2fd^2/\epsilon^2 = O(/\epsilon^2)$. We pick a random multiset S of $O(g \log n)$ variables and guess (by exhaustive enumeration in $2^{g \log n}$ time) the values of a_i for each $i \in S$. Then we use Procedure `Linearize` with error parameter ϵ' to replace each degree d' constraint with $O(n^{d'-1})$ linear constraints, thus obtaining a linear system with $O(Kn^{d-1})$ constraints. Since $(a_1, \ldots, a_n)$ is a feasible solution to the PIP, Lemma 3.6 implies that the probability that the new system is feasible is at least $1 - dn^{d-f}K > 1/2$. We solve the linear system using a polynomial-time algorithm for linear programming in $O((Kn^{d-1})^3)$ time [K84]. Lemma 3.7 implies that the (fractional) solution thus obtained is also a feasible solution for the original PIP, except for an additive error $2d\epsilon' n^d$. Then we randomly round this fractional solution to a 0/1 solution; Lemma 3.3 implies that this increases the additive error by at most $O(n^{d-1/2} \ln n) = o(n^d)$.

Hence we have described a probabilistic procedure that, with probability at least 1/2, produces a 0/1 solution that is feasible for the PIP except for an additive error of $2\epsilon' dn^d = \epsilon n^d$.

The procedure explores $2^{g \log n}$ exhaustive sampling possibilities and for each spends at most $O((Kn^d)^3)$ time in generating the linear constraints and solving them for each guess. The randomized rounding can be done in nearly linear time. Noting that

$$2^{g \log n} = n^g = (n^f)^{4c^2e^2d^2/\epsilon^2} \approx (2Kn^d)^{4c^2e^2d^2/\epsilon^2},$$

we conclude that this term dominates the running time.

3.6 Derandomization

Derandomizing the algorithm in Theorem 1.10 involves derandomizing its components, Procedures `Eval` and `Linearize`. Those depend upon two randomized procedures: Randomized Rounding (used in Lemma 3.3) and the Sampling Lemma. Raghavan [Rag88] derandomized the former through the method of conditional probabilities. Derandomizations of the Sampling Lemma appear in [BR94] and [BGG93]. For example, instead of picking $s = O(\log n/\epsilon^2)$ variables independently, it suffices to pick the variables whose indices are encountered on a random walk of length $O(\log n/\epsilon^2)$ on a constant degree expander [Gil93]. For any fixed sampling experiment, the probably that such a walk works is $1/n^{O(1)}$. Hence our algorithm can deterministically go through all such walks (the number of such walks is $n^{O(1/\epsilon^2)}$ since the expander has $O(1)$ degree). One of the walks is guaranteed to work correctly for *all* of the $poly(n)$ sampling experiments that our algorithms is interested in.

4 Applications

In this section we use our theorem on approximating constant-degree smooth integer programs to construct PTASs for (dense instances of) many problems. Most applications require approximating quadratic programs. Approximating dense MAX-k-SAT requires approximating degree-k integer programs. In later sections we will obtain PTASs for graph bisection and minimum k-way cut. These will require some additional ideas, specifically, a different application of exhaustive sampling.

4.1 MAX-CUT, MAX-DICUT, MAX-HYPERCUT

Note that a δ-dense graph has at least δn^2 edges. Thus the capacity k of the maximum cut exceeds $\delta n^2/2$, since this is the expected size of a cut obtained by randomly assigning each vertex to one side of the graph or the other with equal probability. We already saw in Example 1.8 how to represent MAX-CUT using c-smooth quadratic integer programs with $c = O(1)$. Using the approximation

scheme for quadratic programs in Theorem 1.10, we can in time $n^{O(1/\epsilon^2\delta^2)}$ find a cut of value at least $c - \epsilon\delta n^2/2 \geq (1-\epsilon)k$, in other words a $(1-\epsilon)$ approximation to the maximum cut.

MAX-DICUT has a similar PTAS. Again, an expectation argument shows that the maximum cut in a δ-dense graph exceeds $\delta n^2/4$. The representation by a quadratic program is also similar; in the quadratic program for MAX-CUT in Example 1.8 just replace $(x_i(1-x_j) + x_j(1-x_i))$ in the objective function by $(1-x_i)x_j$.

The PTAS for dense MAX-HYPERCUT(d) is similarly obtained by modeling the problems as a smooth degree-d PIP. For a given edge (set of vertices) S, we use the term $1 - \prod_{i\in S} x_i - \prod_{i\in S}(1-x_i)$. This term is 1 if S is cut and zero otherwise.

4.2 DENSE-k-SUBGRAPH

Let $k \geq \alpha n$. If a graph is δ-dense, then a graph induced by a random subset of k vertices contains $\alpha^2\delta n^2/2$ edges on average. Hence the densest subgraph contains at least $\alpha^2\delta n^2/2$ edges.

We can express the DENSEST-k-SUBGRAPH as the optimum of the following quadratic program.

$$\begin{array}{ll} \textit{maximize} & \sum_{\{i,j\}\in E} x_i x_j \\ \textit{subject to} & x_i \in \{0,1\} \\ & \sum_{i=1}^{n} x_i = k \end{array}$$

Clearly this PIP is 1-smooth. From Theorem 1.10 we know how to find an approximately optimal $0,1$ vector x satisfying $\sum_{i=1}^{n} x_i \in [k \pm g\sqrt{n \ln n}]$. Now we move at most $g\sqrt{n \ln n}$ vertices in or out to get a subset of size k; this affects the number of edges included in the subgraph by at most $gn\sqrt{n \ln n} = o(n^2)$.

The reader may wonder if our algorithm for DENSEST-SUBGRAPH has any application to the CLIQUE problem. We do not see any connection. In fact, approximating CLIQUE in dense graphs is N P-hard (this follows from

the fact that approximating INDEPENDENT SET in degree-5 graphs is N P-hard [ALM+92]).

4.3 MAX-k-SAT

A standard "arithmetization" method can be used to represent MAX-k-SAT as a degree-k smooth IP. Let $y_1, \ldots, y_n$ be the variables and m be the number of clauses. Introduce 0–1 valued variables $x_1, \ldots, x_n$. For each i, $1 \leq i \leq n$, replace each unnegated occurrence of variable y_i by $1 - x_i$, each negated occurrence by x_i, eachlogical $\vee$ by multiplication (over integers), and for each clause subtract the resulting term from 1. This changes each clause into a degree-k polynomial. To give an example, the clause $y_1 \vee \neg y_2 \vee y_3$ is replaced by the term $1-(1-x_1)x_2(1-x_3)$. Now associate, in the obvious way, $0, 1$ assignments to the variables x_i with truth assignments to the boolean variables y_i. Clearly, an assignment of values to the x_i makes the term 1 if the corresponding assignment to the y_i makes the clause true, and 0 otherwise.

Let t_j be the term obtained in this way from the jth clause. The following degree-k program represents the MAX-k-SAT instance, and is smooth.

$$\begin{array}{ll} \text{maximize} & \sum_{j \leq m} t_j(x_1, \ldots, x_n) \\ \text{subject to} & x_i \in \{0,1\} \qquad \forall i \end{array}$$

Now suppose the number of clauses is $m \geq \delta n^k$. Let OPT be the maximum number of clauses that any assignment can satisfy. Since the number of clauses of size k is $m - O(n^{k-1})$, and a random assignment satisfies each of them with probability $1 - 2^{-k}$, we have

$$\text{OPT} \geq (1 - 2^{-k})(m - O(n^{k-1})).$$

By approximating our PIP we can in time $O(n^{2^{4k}/\epsilon^2})$ find an assignment that satisfies $\text{OPT} - \frac{\epsilon}{2^k} n^k \geq (1 - \epsilon)\text{OPT}$ clauses.

4.4 Dense MAX-$\mathcal{SNP}$

As pointed out in [PY91], problems such as MAX-CUT, MAX-k-SAT and MAX-HYPERCUT(d) lie in a class called MAX-$\mathcal{SNP}$, and actually in a subclass called

MAX-$\mathcal{SNP}_0$ in [Pap94]. MAX-$\mathcal{SNP}_0$ was defined using model theory, and it is unclear how to define denseness for MAX-$\mathcal{SNP}_0$ problems. In fact, problems such as vertex cover are in MAX-$\mathcal{SNP}$ only if the degree of the graph is bounded. In this section we give a plausible definition of denseness and show that under this definition, all dense MAX-$\mathcal{SNP}_0$ problems have a PTAS.

Let MAX-k-FUNCTION-SAT be the problem in which the input consists of m boolean functions $f_1, f_2, \ldots, f_m$ in n variables, and each f_i depends only upon k variables. The objective is to assign values to the variables so as to satisfy as many f_i as possible. As is well-known (see [Pap94], Theorem 13.8), a MAX-$\mathcal{SNP}_0$ problem can be viewed as a MAX-k-FUNCTION-SAT problem for some fixed integer k. (An alternative name for MAX-k-FUNCTION-SAT is "constraint satisfaction problems" [KSW97].)

We call an instance of a MAX-$\mathcal{SNP}_0$ problem *dense* if the instance of MAX-k-FUNCTION-SAT produced using it has $\Omega(n^k)$ functions. It is easily checked that our earlier definitions of denseness were sub-cases of this definition. Also, not all MAX-$\mathcal{SNP}$ problems have a dense version under this definition; for example vertex cover is excluded.

A slight modification of the k-SAT technique of Section 4.3 shows that MAX-k-FUNCTION-SAT can be represented by a smooth degree-k integer program, so it follows that dense MAX-k-FUNCTION-SAT has a PTAS.

5 BISECTION and SEPARATOR

In this section we describe a PTAS for BISECTION. Small modifications described at the end give a PTAS for SEPARATOR. Consider a graph with minimum degree δn for some $\delta > 0$ and let k denote the capacity of the minimum bisection. The PTAS consists of two different algorithms, one of which is a PTAS when $k \geq \alpha n^2$, and the other when $k < \alpha n^2$, where α is a certain small constant. For $k \geq \alpha n^2$, we use our PIP approximation algorithm to achieve an additive error at most $\epsilon \alpha n^2$, so that the capacity of the final bisection is at most $(1+\epsilon)k$. When $k \leq \alpha n^2$, such additive approximations do not suffice, and we give a different algorithm. The algorithm uses exhaustive sampling to identify vertices that have "many" neighbors on one side of the bisection, and places them on that

side. We show that this leaves only a small number of unplaced neighbors, whose placement can be done greedily without significantly disturbing the value of our solution.

5.1 Large Bisections

The algorithm for $k \geq \alpha n^2$ is essentially our algorithm for approximating smooth quadratic integer programs. We formulate graph bisection using the same quadratic program as for MAX-CUT (see Example 1.8), except we change "maximize" to "minimize," and add the constraint $\sum x_i = n/2$. Applying our main theorem gives us an assignment to the x_i that makes the objective function less than $k + \epsilon n^2 \leq k(1 + \epsilon/\alpha)$.

There is one small problem: this $0,1$ assignment might not induce a bisection, since it only approximately satisfies the constraint $\sum x_i = n/2$. However, randomized rounding does guarantee there will be $n/2 \pm (\sqrt{n} \log n)$ vertices on each side of the solution. Hence we need to move only $O(\sqrt{n} \log n)$ vertices from one side to another in order to balance the cut. This affects the bisection value by at most $O(n^{1.5} \log n) = o(n^2)$.

5.2 Small Bisections

The case $k \leq \alpha n^2$ is more difficult. We need the following lemma.

Lemma 5.1. *In a minimum bisection, there is one side whose every degree-d vertex has at most $d/2 + 1$ of its neighbors on the other side.*

Proof. If not, then we can reduce the bisection value by picking from each side a vertex that has more than the allowed neighbors on the other side, and switching them. □

Let L_{opt} and R_{opt} denote the sets of vertices on the two sides of a particular minimum bisection. Without loss of generality, we will assume that L_{opt} is the side referred to in Lemma 5.1.

We now give a bisection algorithm in Figure 5.2. For simplicity, we describe it as a randomized algorithm, although we can easily derandomize it using the techniques mentioned earlier. Recall that δ is the denseness of the problem.

1. Pick a set S of $O((\log n)/\delta)$ vertices at random.
2. For each possible partition of S into two sets (S_L, S_R), construct a partition (L, R) as follows.
   (a) Let T be the set of vertices that have more than 5/8 of their sampled neighbors in S_R.
   (b) Put T in R.
   (c) For each vertex $v \notin T$, define bias(v) as
   $$\#(\text{neighbors of } v \text{ not in } T) \quad - \quad \#(\text{neighbors of } v \text{ in } T).$$
   (d) Put the $n/2 - |T|$ vertices with the smallest bias into R.
3. Of all bisections found in the previous step, output the one with the smallest value.

Figure 3: The Bisection Algorithm

Now we prove the correctness of the algorithm. Since it exhaustively tries all possible partitions of the vertices in the sample S, it also tries the "correct" partition, which labels each of the vertices of S according to a minimum bisection (L_{opt}, R_{opt}) of the entire graph. From now on we call this partition (S_L, S_R) of S *special*. We will show that with high probability (over the choice of S) the special partition leads the algorithm to a near-optimum graph bisection.

Let T be the set constructed by the first step of the algorithm using the special partition. The next lemma describes some useful properties of T. Call a vertex *radical-right* if at least 3/4 of its neighbors are in R_{opt}. Note (from Lemma 5.1) that every radical-right vertex must be in R_{opt}.

Lemma 5.2. *With high probability (over the choice of S), T is a subset of R_{opt} and contains every radical right vertex.*

Proof. Let v be any vertex. Since its degree exceeds δn, the Sampling Lemma implies that with high probability a random sample of size $O((\log n)/\delta)$ contains $\Theta(\log n)$ neighbors of v. Conditioning on the number of neighbors in the sample, these neighbors form an unbiased sample of that many neighbors of v.

Suppose $v \in L_{opt}$, and so has fewer than $1/2$ of its neighbors in R_{opt}. Then an application of the Sampling Lemma shows that in a random sample of $\Theta(\log n)$ neighbors of v, the probability that more than $5/8$ of them are in R_{opt} is $n^{-\Omega(1)}$. Hence the probability that $v \in T$ is $n^{-\Omega(1)}$.

Now suppose $v \in R_{opt}$ has more than $3/4$ of its neighbors in R_{opt}. An application of the Sampling Lemma shows that in a random sample of $O(\log n)$ neighbors of v, the probability that less than $5/8$ of them are in R_{opt} is $n^{-\Omega(1)}$. Hence the probability that $v \in T$ is $1 - n^{-\Omega(1)}$. □

The next lemma says that with high probability, T has size close to $n/2$ and thus contains almost all of R_{opt}.

Lemma 5.3. *If T satisfies the two conditions in Lemma 5.2 then $|T| \geq \frac{n}{2} - \frac{4k}{\delta n}$.*

Proof. Every vertex in $R_{opt} - T$ must have $1/4$ of its neighbors in L_{opt}. Let $s = |R_{opt} - T| = n/2 - |T|$. Then the value of the minimum bisection is at least $s\delta n/4$, which by assumption is at most k. Hence $s \leq 4k/(\delta n)$. □

We can now show that with high probability the algorithm produces a bisection close to optimum.

Theorem 5.4. *Assuming $k < \alpha n^2$, with high probability (over the choice of S) the bisection produced by the special partition has value at most $k(1+\epsilon)$, where $\epsilon = 16\alpha^2/\delta^2$.*

Proof. We measure the cost of extending T to a particular set of half the vertices. For any set $U \subseteq \overline{T}$, let $d_{in}(U)$ be twice the number of edges with both endpoints in U, and let $d_{out}(T)$ be the number of edges with exactly one endpoint in T. Further, let $\text{bias}(U)$ be the sum of the biases of vertices in U. We claim that the capacity of the bisection whose one side is $T \cup U$ is

$$d_{out}(T) + \text{bias}(U) - d_{in}(U). \tag{14}$$

To see this, note that the expression starts by counting all edges leaving T. The bias term then subtracts the edges crossing from U to T while adding the edges crossing from U to the other side of the cut. The bias term also *incorrectly* adds (twice, once for each endpoint) the number of edges with both endpoints in U, which do not cross the cut; however, this quantity is subtracted by the $d_{in}(U)$ term, resulting in the correct quantity.

With high probability, the set T produced in the first phase satisfies the conditions in Lemma 5.2. Hence $T \subseteq R_{opt}$, and $s = n/2 - |T| \leq 4k/(\delta n)$. Let $U_{opt} = R_{opt} - T$ be the optimum set of s vertices extending T to R_{opt} and let U_{actual} be the set of s vertices that the algorithm actually picks to extend R.

Since U_{opt} minimizes Equation (14), we know $k = d_{out}(T) + \text{bias}(U_{opt}) - d_{in}(U_{opt})$. On the other hand, U_{actual} (since it includes the s vertices with the smallest bias) minimizes $\text{bias}(U)$, and thus also the expression $d_{out}(T) + \text{bias}(U)$. Thus the capacity of the bisection whose one side is $T \cup U_{actual}$ is at most $k + d_{in}(U_{opt}) - d_{in}(U_{actual})$, which is at most $k + s^2 \leq k + (4k/(\delta n))^2$. Since $k < \alpha n^2$ the capacity is at most $k(1 + 16\alpha^2/\delta^2)$. □

Corollary 5.5. *If a dense graph has bisection value $O(n)$, the optimum bisection can be found in polynomial time.*

Proof. From Lemma 5.3, there are $O(1)$ vertices in $R_{opt} - T$. These can be found by exhaustive search. □

Corollary 5.6. *There is a PTAS for the optimum separator of a dense graph.*

Proof. Guess the number k of vertices on the left side of the separator (by trying all $n/3$ possibilities) and replace $n/2$ by k in the previous discussion of bisection. □

If the minimum degree is not constrained, but the average degree is $\Omega(n)$, our PIP approach still works for large bisection values, but our other algorithm for small bisection values fails. In fact, as shown in Section 7, approximating the minimum bisection for ϵ-dense graphs is no easier than approximating it on general graphs.

6 Algorithms without PIPs

Occasionally, exhaustive sampling is sufficient to solve or approximately solve a dense problem without recourse to PIPs. This was shown for MAX-CUT by Fernandez de la Vega [FdlV94]. We have also seen this for the case of (small) bisections. Here we describe two other problems, multiway cuts and 3-coloring. (The latter was already solved by Edwards [Edw86].)

6.1 MIN-k-CUT

First we consider the k-terminal cut problem. Let δn denote the minimum degree. Let OPT be the capacity of the optimum cut. Note that $OPT \leq kn$, since kn is an upperbound on the capacity of a cut in which $k-1$ of the terminals form singleton groups, while all other vertices form the remaining group. The algorithm relies on the following lemma.

Lemma 6.1. *When $k = o(n)$, there is a k-cut of capacity $(1+o(1))OPT$ that has a special form: at most $2/\delta$ groups of size $\Omega(n)$, and all other groups of size 1 (containing only the terminals).*

Proof. Suppose $S_1, S_2, \ldots, S_k$ (sorted in decreasing order by size) are the k groups in the optimum cut. Let an *ordinary* vertex be one that in the optimum cut has at most $1/4$ of its $\geq \delta n$ neighbors in groups different from its own. Note that there are at most $4k/\delta$ vertices that are not ordinary.

Let t be the number of groups with at least $\delta n/2$ vertices; note that $t \leq 2/\delta$. Consider the modified cut in which all nonterminals from $S_{t+1}, \ldots, S_k$ are moved to one of $S_1, S_2, \ldots, S_t$. We show that the capacity of this cut is at most $(1+o(1))$ times the original capacity; this will prove the lemma.

Since each of $S_{t+1}, \ldots, S_k$ has size less than $\delta n/2$, they can only contain vertices that are not ordinary. Thus they together contain at most $4k/\delta$ vertices. Furthermore, the capacity of the cut is at least

$$\sum_{i=t+1}^{k} \left(|S_i| \frac{\delta n}{2} - \binom{|S_i|}{2}\right).$$

Since $|S_i| = O(k) = o(n)$, the second term is $o(\cdot)$ of the first term. Now consider moving all nonterminals from $S_{t+1}, \ldots, S_k$ to $S_1, \ldots, S_t$; this increases the capacity of the cut by at most

$$\sum_{i=t+1}^{k} \binom{|S_i|}{2},$$

which —as already noted— is $o(\cdot)$ of $\sum_{i=t+1}^{k}(|S_i|\,\delta n/2)$. □

Now we describe the algorithm. Imagine fixing a k-cut of the type described in Lemma 6.1 Let an *ordinary* vertex be one that has at most $1/4$ of its $\geq \delta n$ neighbors in groups different from its own. Let s be the number of nodes that are not ordinary. Clearly, $OPT(1 + o(1)) \geq s\delta n/4$, thus implying

$$s \leq 4k(1 + o(1))/\delta = o(n).$$

First we exhaustively try all $k^{2/\delta}$ ways of picking terminals that will go into non-singleton groups. One of these ways will be "correct." For each of the placements, we try placing the nonterminals using exhaustive sampling. We pick a random sample of $O(\log n)$ vertices and by exhaustively trying all partitions of it (just as in the other algorithms), we can identify for each vertex the group (if one exists) that contains more than $2/3$ of its sampled neighbors, and assign it to that group. The Sampling Lemma shows that this fails to place or misplaces only vertices that are not ordinary, i.e., at most s vertices. Now place each of the remaining vertices in the group that contains a plurality of its neighbors. This gives almost the desired cut, except it may misplace s vertices. Furthermore, if a misplaced vertex contributed x to the correct cut (i.e., x of its neighbors were in another group) then it contributes at most $x + s - 1$ to our cut. Thus the cost of our cut is at most

$$OPT(1 + o(1)) + \binom{s}{2}.$$

Since $OPT(1+o(1)) \geq s\delta n/4$, and $s = o(n)$, the cut has capacity $OPT(1+o(1))$.

A similar PTAS can be designed for the problem without terminals, where the goal is simply to find the best partition into k nonempty groups of vertices.

6.2 3-COLORING

Random sampling also gives a scheme for 3-coloring dense 3-colorable graphs. Since this result replicates that of Edwards [Edw86], details are omitted. Let the colors be $0, 1, 2$. Initially, make all vertices "uncolored." As before, we pick a random sample of $O((\log n))/\epsilon^2)$ vertices and guess their colors. Let us focus on the correct guess. With high probability, at least one neighbor of every vertex is sampled and colored. Now observe that if a vertex has colored neighbors of two different colors, its color is determined. So long as such a vertex exists, color it. When we finish, each remaining uncolored vertex v_i has a neighbor with color c_i and no neighbors of other colors. Now set up an instance of 2-SAT as follows. Assign a variable x_i to vertex v_i that is true if v_i has color $c_i + 1 \pmod 3$ in the optimal coloring, and false otherwise (i.e., v_i has color $c_i - 1 \pmod 3$). For each edge (v_i, v_j), add constraints on the variables x_i and x_j that prevent v_i and v_j from having the same color. Solve the 2-SAT instance in polynomial time, and extract an assignment of colors from the assignment to the variable x_i.

Note that the algorithm can be derandomized easily.

7 $\mathcal{NP}$-completeness results

Thus far, we have described PTASs for dense instances of many $\mathcal{NP}$-hard problems. Now we show that computing optimal solutions in all these cases remains $\mathcal{NP}$-hard, justifying the search for approximation schemes.

For MAX-$\mathcal{SNP}$ type problems, it is usually easy to reduce non-dense instances to ϵ-dense instances. MAX-CUT provides a good example. Suppose OPT is the optimum value of the MAX-CUT problem on a graph $G = (V, E)$. We add a (disjoint) complete graph on n vertices to G. The new graph has $E + \binom{n}{2}$ edges, and is 1/2-dense. Furthermore, the new optimum of MAX-CUT is $OPT + \binom{n}{2}/2$. Thus exact optimization on the new instance is no easier than exact optimization on the old instance. A similar idea works for MAX-k-SAT, MAX-DICUT, etc.

Everywhere–dense BISECTION is also $\mathcal{NP}$-hard. In fact, the standard reduction that shows the $\mathcal{NP}$-completeness of BISECTION produces such instances.

It starts from instances of MAX-50-50-CUT that have constant degree — this is a known N P-hard restriction of MAX-50-50-CUT — and complements the graph to turn it into an instance of BISECTION. The resulting instance is everywhere-ϵ-dense.

Now we indicate two problems for which denseness does not seem to help in designing PTASs: dense instances of BISECTION and everywhere-dense instances of MIN-VERTEX-COVER. We show that if the first problem has a PTAS that then there is a PTAS for general instances of BISECTION (designing such a PTAS is a famous open problem). This follows from the following reduction: Given any instance of BISECTION with n vertices, add to it two disjoint cliques of size $2n$ each. The resulting instance is 2/5-dense, but the capacity of the minimum bisection is unchanged.

Now we show that if there is a PTAS on everywhere-1/2-dense instances of MIN-VERTEX-COVER, then $\mathcal{P} = \mathcal{NP}$. We rely on the result of [PY91, ALM+92] that $\mathcal{P} = \mathcal{NP}$ if there is a PTAS for MIN-VERTEX-COVER on the following simple family of graphs: each of the n vertices has degree at most 5, and the smallest vertex cover has size at least $n/2$. Notice that given such a simple graph we can add a clique on n vertices to the graph and put a complete bipartite graph between the original vertices and the new vertices. This raises the degree of every vertex to n (so the graph becomes everywhere-1/2-dense) and raises the size of the minimum vertex cover by exactly n. Thus a PTAS on the resulting instance is a PTAS on the original instance.

8 Conclusion

We suspect that our technique of approximately reducing quadratic programs to linear programs might be useful in non-dense instances of problems. Of course, the exhaustive random sampling that underlies our work no longer suffices, since an additive approximation is not good enough in that case. But some other approximation method could plausibly replace it. If such an approximation method can be found, it would probably also improve performance on dense instances, by removing the error due to the sampling lemma. Note that the error introduced by the Raghavan-Thompson technique in our approximation algorithm is much

smaller than that introduced by the sampling step (the former error is an additive term of $O(n^{1.5} \log n)$ in the case of quadratic programs; the latter error is $\Omega(n^2)$).

Does a good approximation algorithm exist for general BISECTION? What about an inapproximability result? Our results suggest how *not* to try to prove inapproximability results. Recall that the standard way to prove the $\mathcal{NP}$-completeness of BISECTION uses the fact that 50-50 MAX-CUT is just BISECTION on the complementary graph. Since 50-50 MAX-CUT on degree 5 graphs is MAX-$\mathcal{SNP}$ hard (and therefore has no PTAS), one is tempted to try to use this connection to prove the MAX-$\mathcal{SNP}$-hardness of BISECTION. This naive idea does not work, since the complementary graph of a degree 5 graph is a dense graph in which the minimum bisection has capacity $\Omega(n^2)$. This capacity swamps the gap (in the capacity of the optimum cut) of $\Theta(n)$ present in the instance of 50-50 MAX-CUT, so the the MAX-$\mathcal{SNP}$-hardness of BISECTION does not follow. Of course, now we know an inherent reason why such approaches are unlikely to succeed: BISECTION has a PTAS on dense graphs.

To conclude, we mention some recent research that extends or improves our work. Arora, Frieze, and Kaplan [AFK96] extend our exhaustive sampling idea to design additive approximation schemes for problems in which feasible solutions are permutations (such as the 0-1 *Quadratic Assignment* problem). Frieze and Kannan [FK96] and independently, Goldreich, Goldwasser, and Ron [GGR96] showed that our techniques apply because of certain *regularity* properties in dense graphs, and used this observation to design *linear* time additive approximation schemes for most of the problems we have considered here. Frieze and Kannan also point out connections to constructive versions of Szemeredi's Regularity Lemma.

Acknowledgedments

We thank the anonymous referee for detailed comments that significantly improved the presentation.

References

[AFK96] S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *37th Annual Symposium on Foundations of Computer Science*, pages 21–30, IEEE Computer Society Press, 1996.

[AFW94] N. Alon, A. Frieze, and D. Welsh. Polynomial time randomized approximation schemes for the tutte polynomial of dense graphs. In *Proceedings of the* 35^{th} *Annual Symposium on the Foundations of Computer Science* [IEE94], pages 24–35.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the* 33^{rd} *Annual Symposium on the Foundations of Computer Science* [IEE92], pages 14–23.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *JACM* 45(1):70–122, 1998. Prelim. version in *Proceedings of the* 33^{rd} *Annual Symposium on the Foundations of Computer Science* [IEE92], pages 2–13.

[BCLS84] T. Bui, S. Chaudhuri, T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. In *Proceedings of the* 25^{th} *Annual Symposium on the Foundations of Computer Science*, pages 181–192. IEEE, IEEE Computer Society Press, 1984.

[BGG93] M. Bellare, O. Goldreich, and S. Goldwasser. Randomness in interactive proofs. *Computational Complexity*, 3:319–354, 1993. Abstract in IEEE FOCS 1990.

[B87] R. B. Boppana. Eigenvalues and graph bisection: an average-case analysis *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 280-285, IEEE Computer Society Press, 1987.

[BR94] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proceedings of the* 35^{th} *Annual Symposium on the Foundations of Computer Science* [IEE94], pages 276–287.

[DFK91] M. E. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38:1–17, 1991.

[DJP+94] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994. A preliminary version appeared in ACM STOC 1992.

[Edw86] K. Edwards. The complexity of colouring problems on dense graphs. *Theoretical Computer Science*, 43:337–343, 1986.

[FG95] U. Feige and M. X. Goemans. Approximating the value of 2-prover proof systems, with applications to max-2sat and max-dicut. In *Proceedings of theIsraeli Symposium on Theoretical Computer Science*, pages 182–189, 1995.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the* 32^{nd} *Annual Symposium on the Foundations of Computer Science*, pages 2–12. IEEE, IEEE Computer Society Press, October 1991.

[FdlV94] W. Fernandez de la Vega. MAXCUT has a randomized approximation scheme in dense graphs. *Random Structures & Algorithms*, vol. 8(3):187-198, 1996. Prelim. manuscript, October 1994.

[FL81] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*:1(4), 349–355, 1981.

[FK96] A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *Proceedings of the* 37^{th} *Annual Symposium on the Foundations of Computer Science*, pages 12–20. IEEE, IEEE Computer Society Press, October 1996.

[GGR96] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. In *37th Annual Symposium on Foundations of Computer Science*, pages 339–348, IEEE Computer Society Press, 1996.

[Gil93] D. Gillman. A Chernoff bound for random walks on expanders. In *Proceedings of the* 34^{th} *Annual Symposium on the Foundations of Computer Science* [IEE93], pages 680–691.

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

[GW94] M. X. Goemans and D. P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *JACM* 42(6): 1115-1145, 1995. Extended abstract in *Proceedings of the* 26^{th} *ACM Symposium on Theory of Computing*, pages 422–431. ACM, ACM Press, May 1994.

[H96] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Proceedings of the* 37^{th} *Annual Symposium on the Foundations of Computer Science*, pages 627–636. IEEE Computer Society Press, 1996.

[H97] J. Håstad. Some optimal inapproximability results. *Proceedings of the* 28^{th} *ACM Symposium on Theory of Computing* pages 1–10, 1997.

[H64] W. Höffding. Probability inequalities for sums of bounded random variables. *Journal of the American Stastical Association*, 58(301):13–30, 1964.

[IEE92] IEEE. *Proceedings of the* 33^{rd} *Annual Symposium on the Foundations of Computer Science.* IEEE Computer Society Press, October 1992.

[IEE93] IEEE. *Proceedings of the* 34^{th} *Annual Symposium on the Foundations of Computer Science.* IEEE Computer Society Press, November 1993.

[IEE94] IEEE. *Proceedings of the* 35^{th} *Annual Symposium on the Foundations of Computer Science.* IEEE Computer Society Press, November 1994.

[IK75] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subsets problems. *Journal of the ACM*, 22(4):463–468, 1975.

[JS89] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.

[JS93] M. Jerrum and G. B. Sorkin. Simulated annealing for graph bisection. In *Proceedings of the* 34^{th} *Annual Symposium on the Foundations of Computer Science* [IEE93], pages 94–103.

[KZ97] H. Karloff and U. Zwick. A $7/8 - \epsilon$ approximation for MAX-3SAT? In *Proceedings of the* 37^{th} *Annual Symposium on the Foundations of Computer Science* 1997.

[K84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373-395, 1984.

[KK82] N. Karmakar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the* 23^{rd} *Annual Symposium on the Foundations of Computer Science*, pages 312–320. IEEE, IEEE Computer Society Press, 1982.

[KLM89] R. M. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.

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

[KP92] E. Koutsoupias and C. H. Papadimitriou. On the greedy heuristic for satisfiability. *Information Processing Letters*, 43(1):53–55, 1992.

[KP93] G. Kortsarz and D. Peleg. On choosing a dense subgraph. In *Proceedings of the* 34^{th} *Annual Symposium on the Foundations of Computer Science* [IEE93], pages 692–701.

[LR88] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the* 29^{th} *Annual Symposium on the Foundations of Computer Science*, pages 422–431. IEEE, IEEE Computer Society Press, October 1988.

[LY93] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proceedings of the* 25^{th} *ACM Symposium on Theory of Computing*, pages 286–293. ACM, ACM Press, May 1993.

[P̀76] L. Pòsa. Hamiltonian circuits in random graphs. *Discrete Mathematics*, 14:359–364, 1976.

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

[PY91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *JCSS*, 43(3):425–440, 1991. A preliminary version appeared in STOC 1988.

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximate packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–43, October 1988.

[RT87] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[S75] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22(1):115–124, 1975.

[SV91] H. Saran and V. V. Vazirani. Finding k-cuts within twice the optimal. In IEEE, editor, *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 743–751, San Juan, Porto Rico, October 1991. IEEE Computer Society Press.

[Shm94] D. Shmoys. Computing near-optimal solutions to combinatorial optimization problems. In *Combinatorial Optimization*, DIMACS Series

in Discrete Math and Theoretical Computer Science, (W. Cook, L. Lovasz, and P.D. Seymour, eds.) AMS, 1995, pages 355-397.

[Yan92] M. Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of the* 3^{rd} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9. ACM-SIAM, January 1992.