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

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 sources, and maximum 3-satisfiability. Dense graphs for us are graphs with minimum degree $\Theta(n)$, although some of our algorithms work so long as the graph is dense "on average". (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 is a "dense" polynomial of constant degree.

1 Introduction

Approximation algorithms, whenever they can be found, are a practical way to deal with the $\mathcal{NP}$-hardness of optimization problems. Ideally, they should run in polynomial time and have a small *approximation ratio*—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 definition is inverted so that the ratio is always at least 1.)

The approximation properties of different problems vary a great deal (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 within a factor of n^δ in polynomial time. Others problems, such as those related to graph separators [LR88], have algorithms with approximation ratios close to $O(\log n)$ and no non-trivial lower bounds. Still others, such as the maximum cut problem, can be approximated to within some fixed constant factor [GW94]. Only a few problems, such as bin packing [KK82] and knapsack problems [IK75], are known to have *polynomial time approximation schemes (PTASs).*

A PTAS gives, for any fixed $\epsilon > 0$, a polynomial time algorithm with approximation ratio $1 + \epsilon$. A PTAS is a valuable approximation algorithm, since it allows us to trade off the accuracy of the approximation with the running time. (Note that the definition of a PTAS allows the algorithm's running-time to depend arbitrarily on ϵ.)

However, recent results show that unless $\mathcal{P} = \mathcal{NP}$, PTASs do not exist for many $\mathcal{NP}$-hard problems, including all MAX-SNP-hard problems such as vertex cover, maximum 3-satisfiability, maximum cut, metric TSP, and multiway cuts (see [ALM+92, 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.

*Princeton University. arora@cs.princeton.edu

†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. marek@cs.bonn.edu

1.1 Our Results

This paper gives PTASs for a large class of $\mathcal{NP}$-hard problems on *dense* instances. Density is a property of the problem instance; 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 (in the probabilistic sense) are dense, as are almost all 3-SAT instances.

Our techniques apply, in a uniform way, to a broad spectrum of problems, which otherwise seem to have various degrees of hardness (at least on general—i.e., non-dense—instances). Some, like maximum cut and maximum k-satisfiability, are MAX-SNP-complete. Thus they do not have PTASs in general, but they can all be approximated within some constant factor in polynomial time [PY91]. Others, like graph bisection and separation, are not known to be approximable within a factor better than $O(\log n)$, but also are not known to be hard to approximate. It is notable that existing weak-approximation algorithms do not even give a bisection, but instead give a $1/3 : 2/3$ cut of the graph that approximates the minimum bisection in value. Our PTAS gives an exact bisection.

Most of our PTASs are instances of a general approximation algorithm for certain *smooth* integer programs of low degree, which might find other applications.

Two ideas underlie our general approach. To understand the first, consider the (undirected) maximum cut problem, in which the goal is to partition the vertices of a graph into two groups—called the *left* and *right* sides—so as to minimize the number of edges with an endpoint on each side. Notice that in the optimium 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. So assume we have partitioned the sampled vertices correctly according to the optimal cut. Now consider some unsampled vertex. With high probability, some of its neighbors were sampled (*high probability* means probability $1-n^{-\Omega(1)}$). Furthermore, 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: by examining the sampled neighbors of a vertex, we can estimate what fraction of its neighbors lie on each side of the optimum cut. The collection of these estimates—one per vertex—allows us to turn the classical quadratic program for MAX-CUT into an integer *linear* program whose solution approximates the solution to the quadratic program. This lets us find an approximate solution using the randomized rounding techniques of Raghavan and Thompson [RT87].

Generalizations of these ideas work for dense instances of the following problems:

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. A .878-approximation algorithm is given in [GW94].

MAX-DICUT: The directed version of the MAX-CUT problem. A .859-approximation algorithm is given in [FG95] (improving [GW94]).

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.

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. No good approximation algorithm exists that output the actual bisection. But some algorithms, including those using eigenvalues ([BH92]) or simulated annealing ([JS93]) do well on certain random graphs (see also [BCLS84]).

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 crossing edges. An algorithm in [LR88] achieves approximation

ratio $O(\log n)$.

MAX-k-SAT: Given a k-CNF formula, find a true-false assignment to the variables making the maximum possible number of clauses true. An algorithm in [Yan92] approximates it within a factor 3/4. This has recently been improved to 0.758 ([GW94]).

MIN-k-CUT: Given an n-vertex graph with k *source* vertices, partition the graph vertices into k groups such that (i) each group contains one source and (ii) the number of edges with end-points in different groups is minimized. A $(2 - 1/k)$-approximation is given in [DJP+92].

DENSE-k-SUBGRAPH: Given a graph, find a subset of k vertices that induces a graph with the most possible edges. This problem was studied in [KP93], where an approximation algorithm with ratio $n^{2/5}$ was presented.

3-COLORING: Color the vertices of a graph with 3 colors such that no two adjacent vertices have the same color.

Exact optimization is $\mathcal{NP}$-hard for each of these problems, typically by a reduction from the non-dense to the dense case. We now define a natural notion of *dense instance* for each problem. Exact optimization remains $\mathcal{NP}$-hard on dense instances for all of them except MIN-k-CUT and 3-COLORING.

Definition 1.1 *A graph is* ϵ-dense *if it has* $\epsilon n^2/2$ *edges. It is* everywhere-ϵ-dense *if the minimum degree is* ϵn. *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 *Everywhere-dense instances of all the problems listed above have PTASs.*

Theorem 1.3 *Dense (and thus everywhere-dense) instances of the following problems have PTASs: MAX-CUT, MAX-DICUT, MAX-k-SAT for any constant k, MAX-k-CUT for* $k = o(n)$ *, DENSE-k-SUBGRAPH for* $k = \Omega(n)$*, and MAX-HYPERCUT(d).*

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

Note: There are stronger forms of some of the above results that we omit from this abstract. To give an example, we can solve BISECTION and SEPARATOR on dense graphs *exactly* when the objective is $O(n)$. We also note that the 3-COLORING result is not new—see [Edw86].

As mentioned earlier, the PTASs share common design principles, and are quite similar to the MAX-CUT algorithm outlined above. A better unifying framework turns out to be a general approximation algorithm for low-degree integer programs with a certain smoothness condition. Most of the above PTASs are subcases of this general algorithm, though BISECTION and MIN-k-CUT require additional ideas. Further, in Section 4.1, we will give a plausible definition of denseness for the class *MAX-SNP* defined in [PY91]. Our algorithm for approximating low-degree integer programs gives a PTAS for all dense MAX-SNP problems.

Definition 1.5 *A* smooth degree-d integer program *has the form*

$$\begin{array}{lll} \textit{maximize} & p(x_1, \ldots, x_n) & \\ \textit{subject to} & x_i \in \{0,1\} & \forall i \leq n \end{array} \qquad (1)$$

where $p(x_1, \ldots, x_n)$ *is a degree-d polynomial in which the coefficient of each degree-i monomial (term) is* $O(n^{d-i})$. *The program could involve minimization instead of maximization.*

Smooth integer programs represent many combinatorial problems in a natural way.

Example 1 A smooth degree-2 integer program has the form

$$\begin{array}{lll} \text{maximize} & \sum a_{ij} x_i x_j + \sum b_i x_i + c & \\ \text{subject to} & x_i \in \{0,1\} & \forall i \leq n \end{array}$$

where each $a_{ij} = O(1)$, $b_i = O(n)$, $c = O(n^2)$.

We show how to represent MAX-CUT on the graph $G = (V, E)$. Define a variable x_i for each vertex v_i. Then, assign 0, 1 values to the x_i so as to maximize

$$\frac{1}{2} \sum_{\{i,j\} \in E} (x_i(1 - x_j) + x_j(1 - x_i)).$$

(Notice, an edge $\{i, j\}$ contributes 1 to the sum when $x_i \neq x_j$ and 0 otherwise.) Expanding the sum shows that the coefficients of the quadratic terms are 0 and ± 1 while the coefficients of the linear terms are $O(n)$.

Theorem 1.6 *Let OPT be the maximum value of the objective function in the IP in Equation (1). For each fixed $\epsilon > 0$ there is a polynomial-time algorithm that produces a $0,1$ assignment for the x_i satisfying*

$$p(x_1, \ldots, x_n) \geq OPT - \epsilon n^d.$$

For minimization problems the solution satisfies

$$p(x_1, \ldots, x_n) \leq OPT + \epsilon n^d.$$

Related Work

There are known examples of problems which are seemingly easier to approximate in dense graphs than in general graphs. For instance, in graphs with degree more than $n/2$, the following problems are solved: finding Hamiltonian cycles [Po76] and approximating 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].

Vega [dlV94] has independently developed a PTAS for everywhere-dense MAX-CUT using principles similar to ours; however, his algorithm does not appear to 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.

The exhaustive sampling approach also appears, in a different context, in [KPa92].

2 Approximating Smooth IPs

This section describes the proof of Theorem 1.6. For simplicity, we describe the proof for the case of quadratic programs, and then merely outline a proof for the general case.

The proof uses two lemmas. The first is a standard fact about estimating the sum of n numbers using random sampling.

Lemma 2.1 (Sampling Lemma) *Let p be the sum of n numbers $a_1, \ldots, a_n$, each $O(1)$. When we pick a random subset of $s = O(\log n/\epsilon^2)$ numbers and compute their sum q, with high probability qn/s lies in the range $[p - \epsilon n, p + \epsilon n]$.*

In other words, we can sample to estimate the sum to within an *additive* error of ϵn.

The next lemma, due to Raghavan and Thompson [RT87] shows how to round approximate linear integer programs by solving the corresponding fractional program and then rounding the fractional solutions to integers.

Lemma 2.2 (Randomized Rounding) *Let $x = (x_i)$ be a vector of n variables, $0 \leq x_i \leq 1$, that satisfies certain linear constraints $a_i x = b_i$, where each $a_i = O(1)$. Construct y_i randomly by setting $y_i = 1$ with probability x_i and 0 otherwise. Then with high probability, $a_i y = b_i + O(\sqrt{n \log n})$.*

Now we state and prove Theorem 1.6 for quadratic programs. For simplicity we describe a randomized algorithm. Later we show how to derandomize it.

Theorem 2.3 *Suppose there is a $0,1$ solution to the quadratic integer program*

$$xAx + bx \geq c, \tag{2}$$

where x is a vector of n variables, A is an $n \times n$ matrix with entries $O(1)$, b is a vector of length n, with entries $O(n)$, and c is a constant. Then for any fixed ϵ, in time $n^{O(1/\epsilon^2)}$ we can find an assignment of $0,1$ values to x such that

$$xAx + bx \geq c - \epsilon n^2.$$

Proof: The main idea is to reduce the instance of quadratic programming to an instance of linear programming, and then use the Raghavan-Thompson technique to round the fractional solution to a $0,1$ solution. The reduction runs in time $n^{O(1/\epsilon^2)}$, and is meaningful only when $c > \epsilon n^2$, the nontrivial case of the theorem.

Denote by x^* some value for x that satisfies Equation (2). Rewrite the formula $xAx + bx$ as $\sum_i r_i x_i$, where $r(x) = xA + b$, so that $r_i = \sum x_j a_{ji} + b_i$. Suppose that we knew the value $r^* = x^* A + b$. Then consider the following set of linear equations:

$$\begin{aligned} xA + b &= r^* \\ r^* x \geq c \quad &, \quad 0 \leq x \leq 1 \end{aligned}$$

Note that the above system has a feasible $0,1$ solution, namely x^*. In polynomial time we can solve the system by linear programming, obtaining a fractional

solution x. Randomized rounding of this solution (Lemma 2.2) gives, with high probability, a $0,1$ solution y such that $ya_i + b_i = r_i^* + O(\sqrt{n \log n})$. Furthermore, since each $r_i^* = O(n)$, we know that with high probability we will have $r^* y \geq c - O(n) \cdot O(\sqrt{n \log n})$. Then

$$\begin{aligned} yAy + by &= (yA + b)y \\ &= r^* y - O(n^{3/2}\sqrt{\log n}) \\ &\geq c - O(n^{3/2}\sqrt{\log n}). \end{aligned}$$

Of course, this all depends on our assumption that the values r_i^* are available. We will shortly show that in polynomial time, it is possible to *estimate* these values, finding r_i such that $|r_i^* - r_i| < \epsilon n$. We show that the above idea works even with such estimates. Define a slightly different linear program:

$$\begin{aligned} \max \quad & rx \\ xA + b &\leq r + \epsilon n \\ xA + b &\geq r - \epsilon n \\ 0 \leq \quad & x \quad \leq 1. \end{aligned}$$

(For a vector v and scalar s, the notation $v+s$ denotes the vector with i^{th} component $v_i + s$.)

As before, x^* demonstrates that there is a feasible $0,1$ solution to this system for which the objective function is $rx^* = r^* x^* - (r^* - r)x^* \geq c - \epsilon n^2$. Again, solve the system by linear programming, and let x be the fractional solution thus obtained. Note $rx \geq rx^*$. Let $\delta = xA + b - r$, so that $|\delta_i| < \epsilon n$ and thus $|\delta y| \leq \epsilon n^2$. If we now proceed according to the randomized rounding scheme above, we will get y_i such that $ry \geq rx - O(n^{3/2}\sqrt{\log n})$. Thus,

$$\begin{aligned} yAy + by &= (yA + b)y \\ &= (yA + b - (xA + b))y + \delta y + ry \\ &\geq O(n^{3/2}\sqrt{\log n}) - \epsilon n^2 \\ &\quad +(c - \epsilon n^2 - O(n^{3/2}\sqrt{\log n})) \\ &\geq c - (2\epsilon + o(1))n^2 \end{aligned}$$

It remains to prove that we can estimate r^* to within the desired accuracy. To do so we give a randomized method to produce $n^{O(1/\epsilon^2)}$ estimates for r^*, one of which is accurate. This is good enough, since we can run the above algorithm for each estimate and choose the answer that works best.

The estimation method is a generalization of the sampling approach for MAX-CUT outlined in Section 1.1. Choose a set S of $k = O((\log n)/\epsilon^2)$ indices at random. Exhaustively go through each of the $2^k = n^{O(1/\epsilon^2)}$ ways of assigning values 0 or 1 to each variable whose index is in S. For each assignment, produce an estimate r of r^* by setting

$$r_i = b_i + \frac{n}{k} \sum_{j \in S} a_{ij} s_j$$

where s_j is the value assigned to the jth variable. Note that trying all possible assignments ensures that we try the "correct" assignment, namely, one in which $s_j = x_j^*$ for each $j \in S$. Call the estimate corresponding to this assignment the *special* estimate.

To finish the proof, it suffices to show that the special estimate approximates r^* with additive error ϵn, as desired. To do so, use the Sampling Lemma (2.1). Consider one sum $r_i^* = a_i x^* + b_i$. Since b_i is a constant, it suffices to estimate $\sum a_{ij} x_j^*$. By sampling and guessing values for $O(\epsilon^{-2} \log n)$ of the variables x_j^*, we determine the values of the same number of terms $a_{ij} x_j^*$ in the the sum for r_i^*. Since each $a_{ij} x_j^*$ is $O(1)$, the Sampling Lemma tells us that this sample of term values lets us estimate r_i^* to within ϵn with probability $1 - 1/2n$. We conclude that with probability at least $1 - n/2n = 1/2$, all n sums are estimated correctly. ■

The proof of Theorem 1.6 for integer programs of degree exceeding 2 goes via an induction on degree. Just as we randomly reduced (in an approximate sense) a quadratic program to a linear program, we can reduce degree-d programs to degree-$(d-1)$ programs.

2.1 Derandomization

Derandomizing the algorithm in Theorem 2.3 involves derandomizing its components: randomized rounding and the Sampling Lemma. Raghavan [R88] derandomized the former through the method of conditional probabilities. Derandomizations of the Sampling Lemma appear in [BR94] and [BGG93]. For example, instead of picking $s = O(\log n/\epsilon^2)$ vertices independently, it suffices to pick the vertices encountered on a random walk of length $O(\log n/\epsilon^2)$ on a constant degree expander [Gil93]. The number of such walks is $n^{O(1/\epsilon^2)}$, so our algorithm can deterministically go through all possible choices.

3 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. Obtaining PTASs for graph bisection and minimum k-way cut requires some additional ideas, specifically, a different application of the Sampling Lemma.

3.1 MAX-CUT, MAX-DICUT, MAX-HYPERCUT

Note that a δ-dense graph has at least δn^2 edges. Thus the capacity c 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 how to represent MAX-CUT using smooth quadratic integer programs with coefficient bound $O(1)$. Using the approximation scheme for quadratic programs in Theorem 2.3, we can in time $n^{O(1/\epsilon^2\delta^2)}$ find a cut of value at least $c - \epsilon\delta n^2/2 \geq (1-\epsilon)c$, in other words a $(1-\epsilon)$ approximation to the maximum cut.

MAX-DICUT has a similar PTAS. Again, an expected case 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 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 modelling the problems as a smooth degree-d IP.

3.2 MAX-k-SAT

We show how 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 occurence of variable y_i by $1 - x_i$, each negated occurence by x_i, the logical $\vee$ by multiplication (over integers), and for each clause subtract the resulting term from 1. Thus a clause changes 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.

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

Now suppose the number of clauses m is at least δn^k. Let OPT be the maximum number 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

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

Using our general theorem on approximating degree-k balanced programs with coefficient bound $O(1)$, we can in time $O(n^{2^k/\epsilon^2})$ find an assignment that satisfies $\text{OPT} - \frac{\epsilon}{2^k}n^k$ clauses, which is at least $(1-\epsilon)\text{OPT}$.

3.3 BISECTION and SEPARATOR

In this section we describe a PTAS for BISECTION; the PTAS for SEPARATOR is similar and is not described. Let the graph have minimum degree δn for $\delta > 0$ 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).

The algorithm for $k \geq \alpha n^2$ is essentially our algorithm for approximating smooth quadratic integer programs. Note that we can formulate graph bisection using the same quadratic program as for MAX-CUT (see Example 1), except we change "maximize" to "minimize," and add the constraint $\sum x_i = n/2$. Although smooth integer programs with added linear constraints were not previously discussed, they can clearly be solved the same way as before (i.e., solving linear programs and then using randomized rounding).

When the capacity of the minimum bisection, k, is at least αn^2, our algorithm for approximating integer programs 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 a slight problem, though: this $0,1$ assignment might not induce a bisection, since it only approximately satisfies the constraint $\sum x_i = n/2$. However, the error introduced by the randomized rounding (Lemma 2.2) is small: on a linear system that includes the (fractional) constraint $\sum x_i = n/2$, the $0/1$ values obtained after rounding satisfy $\sum x_i \in [n/2 \pm O(\sqrt{n}\log n)]$. 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 $O(n^{1.5}\log n) = o(n^2)$.

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

Lemma 3.1 *In a minimum bisection, there is one side whose every vertex has at most half its neighbors on the other side.*

Proof: If not, then we can reduce the cut value by picking from each side a vertex that has more than half its 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 3.1.

The algorithm is given in Figure 3.3. For simplicity, we describe it as a randomized algorithm, although we can easily derandomize it using the techniques mentioned earlier.

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.

Lemma 3.2 *With high probability (over the choice of S), T is a subset of R_{opt}, and contains every vertex that has more than $3/4$ of its neighbors in T (note that such a vertex must be in R_{opt}).*

1. Pick a set S of $O((\log n)/\delta^2)$ 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 neighbors in S_r.
   (b) Put T in R.
   (c) For each vertex $v \notin T$, define bias(v) as
   $$\#(\text{neighbors of } v \text{ not in } T) - \#(\text{neighbors of } v \text{ in } T).$$
   (d) Put the $n/2 - |T|$ vertices with the smallest bias into R.
3. Of all bisections, output the one with the smallest value.

Figure 1: The Bisection Algorithm

Proof: Let v be any vertex. Since its degree exceeds δn, the Sampling Lemma implies that with high probability a random sample of size $O((\log n)/\delta^2)$ contains $\Theta(\log n)$ 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 $1/poly(n)$. Hence the probability that $v \in T$ is $1/poly(n)$.

Now suppose v is 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 more than $5/8$ of them are in R_{opt} is $1 - 1/poly(n)$. Hence the probability that $v \in T$ is $1 - 1/poly(n)$. ■

The next lemma says that with high probability, T has size close to $n/2$.

Lemma 3.3 *If T satisfies the two conditions in Lemma 3.2 then $|T| \geq \frac{n}{2}(1 - \frac{8k}{\delta n^2})$.*

Proof: Every vertex in $R_{opt} - T$ must have $1/4$ of its neighbors in L_{opt}. Let $s = |R_{opt} - T| = n/2 -$

$|T|$. The value of the minimum bisection is at least $s\delta n/4$, which by assumption is at most k. Hence $s \leq 4k/(\delta n)$. ■

The following lemma shows that with high probability the algorithm produces a bisection close to optimum.

Theorem 3.4 *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: With high probability, the set T produced in the first phase satisfies the conditions in Lemma 3.2. Hence $T \subseteq R_{opt}$, and $s = n/2 - |T| \leq 4k/(\delta n)$.

For any set $U \subseteq \overline{T}$, $|U| = s$, 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 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

$$d_{out}(T) + \text{bias}(U) - d_{in}(U). \tag{3}$$

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.

Let $U^* = R_{opt} - T$ be the optimum way to extend T to R_{opt} and let U_{actual} be the set of s vertices that the algorithm picks to actually extend R.

Since U^* minimizes Equation (3), we know $k = d_{out}(T) + \text{bias}(U^*) - d_{in}(U^*)$. On the other hand, U_{actual} (since it includes the s vertices with the smallest bias) minimizes 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 $k + d_{in}(U^*) - d_{in}(U_{actual})$, which is at most $k + s^2 \leq k + (4k/(\delta n))^2$. Since $k < \alpha n^2$ the capacity is at most $k(1 + 16\alpha^2/\delta^2)$. ■

If the minimum degree is not constrained, but the average degree is $\Omega(n)$, our randomized rounding scheme still works for large bisection values, but our other algorithm for small bisection values fails. So the question of a PTAS for bisection on dense graphs remains open.

4 MIN-k-CUT

Let δn denote the minimum degree. Note that the optimum cut has value at most kn, since that is the capacity of a cut in which $k-1$ of the sources form singleton groups, while all other vertices form the remaining group. It follows that $O(k)$ vertices can have more than 1/4 of their $\Omega(n)$ neighbors in different groups from their own.

First suppose k is constant. Then, by picking a random sample of $O(\log n)$ vertices and doing an exhaustive search on it (just as in the other algorithms), identify for each vertex the group which contains more than 1/4 of its neighbors. The Sampling Lemma shows that this fails to place or misplaces only those vertices with more than 1/4 of their neighbors in a group other than their own, i.e. $O(k)$ vertices. Now try all $O(k^k)$ possible assignments of these $O(k)$ vertices to groups. One of them gives the optimum cut.

When k is large (but still $o(n)$) this approach still works. We claim that the minimum k-source cut has a special form: $O(1)$ groups of size $\Omega(n)$, and all other groups of size 1 (containing only the sources). It follows that exhaustive sampling as described above allows us to identify the non-singleton groups exactly.

To see that the claim is true, assume the minimum k-way cut contains a group of more than 1 but less than $\delta n/2$ vertices. Then each vertex in this group can have at most $\delta n/2$ neighbors within the group and must therefore have at least $\delta n/2$ neighbors outside the group. This implies that the value of this cut is at least $(\delta n/2)^2$. If $k = o(n)$, this contradicts the fact that the minimum cut is at most kn.

A similar result holds for the problem without sources, where the goal is simply to find the best partition into k nonempty groups of vertices.

4.1 Dense MAX-SNP

As pointed out in [PY91], problems such as MAX-CUT, MAX-k-SAT, MAX-HYPERCUT(d), etc. lie in a class called MAX-SNP (also called MAX-SNP$_0$ in [Pap94]). Owing to the model-theoretic nature of the definition of MAX-SNP, it is unclear how to define denseness for MAX-SNP problems. In fact, problems

such as vertex cover are in MAX-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-SNP problems have a PTAS.

Let MAX-k-FUNCTION-SAT be the problem in which the input consists of m boolean functions $f_1, f_2, \ldots$ 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's as possible. As is well-known (see [Pap94], Theorem 13.8), a MAX-SNP problem can be viewed as a MAX-k-FUNCTION-SAT problem for some fixed integer k.

We call an instance of a MAX-SNP 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 subcases of this definition. Also, not all MAX-SNP problems have a dense version under this definition; for example vertex cover is excluded.

A slight modification of the technique of Section 3.2 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 Conclusion

We suspect that our technique of approximately reducing quadratic programs to linear programs might be useful in nondense instances of problems. Of course, the exhaustive random sampling we use fails, 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 (an additive error of $O(n^{1.5} \log n)$) in our approximation algorithm is much smaller than that introduced by the sampling.

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 balanced MAX-CUT is just BISECTION on the complementary graph. Balanced MAX-CUT (like unrestricted MAX-CUT) is MAX-SNP hard, and therefore has no PTAS. However, the operation of taking the complement of a sparse graph yields a dense graph, for which we have just given approximation algorithms for MAX-CUT. Hence the MAX-SNP-hardness proof does not extend to BISECTION. Of course, now we know why that approach is unlikely to succeed: BISECTION has a PTAS on dense graphs.

References

[AFW94] N. Alon, A. Frieze, and D. Welsh. Polynomial time randomized approximation schemes for the tutte polynomial of dense graphs. In *Proc.* 35th *FOCS*, pages 24–35. IEEE, IEEE Computer Society Press, November 1994.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proc.* 33rd *FOCS*, pages 14–23. IEEE, October 1992.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. In *Proc. 33rd FOCS*, pages 2–13, IEEE, 1992.

[BCLS84] T. Bui, S. Chaudhuri, T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. In *Proc.* 25th *FOCS*, pages 181–192. IEEE, 1984.

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

[BH92] R.B. Boppana and M.M. Halldorsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32:180–196, 1992.

[BR94] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proc.* 35th *FOCS*, pages 276–287. IEEE, 1994.

[DJP+92] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiway cuts. In *Proc.* 24th *ACM STOC*, pages 241–251. ACM Press, May 1992.

[dlV94] W.F. de la Vega. MAXCUT has a randomized approximation scheme in dense graphs. manuscript, October 1994.

[Edw86] K. Edwards. The complexity of colour-

ing 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 *Proc. ISTCS95*, pages 182-189.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proc. 32nd FOCS*, pages 2–12, IEEE, 1991.

[Gil93] D. Gillman. A chernoff bound for random walks on expanders. In *Proc.* 34th *FOCS*, pages 680–691. IEEE, November 1993.

[GW94] M.X. Goemans and D.P. Williamson. .878-approximation algorithms for MAX CUT and MAX 2SAT. In *Proc.* 26th *STOC*, pages 422–431. ACM Press, May 1994.

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

[JS89] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6), 1989.

[JS93] M. Jerrum and G. B. Sorkin. Simulated annealing for graph bisection. In *Proc.* 34th *FOCS*, pages 94–103. IEEE, November 1993.

[KK82] N. Karmaker and R.M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc.* 23rd *FOCS*, pages 312–320. IEEE, 1982.

[KP93] G. Kortsarz and D. Peleg. On choosing a dense subgraph. In *Proc.* 34th *FOCS*, pages 692–701. IEEE, November 1993.

[KPa92] E. Koutsoupias and C.H. Papadimitriou. On the greedy heuristic for satisfiability. *IPL, 43* pages 53–55, 1992.

[LR88] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proc.* 29th *FOCS*, pages 422–431. IEEE, October 1988.

[LY93] C. Lund and M. Yannakakis. hardness of approximating minimization problems. In *Proc.* 25th *STOC*, pages 286–293. ACM, May 1993.

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

[PY91] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *JCSS*, 43:425–440, 1991. Preliminary Version in Proc. ACM STOC, 1988.

[Po76] L. Pósa. Hamiltonian circuits in random graphs. *Discrete Mathematics*, 14:359-364, 1976.

[R88] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximate packing integer programs. *JCSS*, 37(2):130–43, October 1988.

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

[Shm94] D. Shmoys. Computing near-optimal solutions to combinatorial optimization problems. To appear in *Dimacs Series in Discrete Math and Theoretical Computer Science*, 1994.

[Yan92] M. Yannakakis. On the approximation of maximum satisfiability. In *Proc.* 3rd *SODA*, pages 1–9. ACM-SIAM, January 1992.