Texts in Computational Complexity: Approximation Problems

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

January 28, 2006

Introduction

The notion of approximation is a natural one, and has arisen also in other disciplines. Its most common use is in references to quantities (e.g., the length of one meter is approximately forty inches), but it is also used when referring to qualities (e.g., an approximately correct account of a historical event). In the context of computation, the notion of approximation modifies computational tasks such as search and decision problems. (In fact, we have encountered it already as a modifier of counting problems; see [12, Text 15].)

Two major questions regarding approximation are (1) what is a "good" approximation, and (2) can it be found easier than finding an exact solution. The answer to the first question seems intimately related to the specific computational task at stake and to its role in the wider context (i.e., the higher level application): a good approximation is one that suffices for the intended application. Indeed, the importance of certain approximation problems is much more subjective than the importance of the corresponding optimization problems. This fact seems to stand in the way of attempts at providing a comprehensive theory of approximation problems (e.g., classes of natural approximation problems that are shown to be computationally equivalent).

Turning to the second question, we note that in numerous cases natural approximation problems seem to be significantly easier than the corresponding original ("exact") problems. On the other hand, in numerous other cases, natural approximation problems are computationally equivalent to the original problems. We shall exemplify both cases by reviewing some specific results, but regret not being able to provide any systematic classification.

Mimicking the two standard uses of the word *approximation*, we shall distinguish between approximation problems that are of the "search type" and problems that are have a clear "decisional" flavor. In the first case we shall refer to a function that assigns values to possible solutions (of a search problem); whereas in the second case we shall refer to distances between instances (of a decision problem). Needless to say, at times the same computational problem may be cast in both ways, but for most natural approximation problems one of the two frameworks is more appealing than the other.

Teaching note: Most of the results presented in this section refer to specific computational problems and (with one exception) are presented without a proof. In view of the complexity of the corresponding proofs and the merely illustrative role of these results, we recommend doing the same in class.

1 Search or Optimization

Many search problems involve a set of potential solutions (per each problem instance) such that different solutions are assigned different "values" (resp., "costs") by some "value" (resp., "cost") function. In such a case, one is interested in finding a solution of maximum value (resp., minimum cost). A corresponding approximation problem may refer to finding a solution of approximately maximum value (resp., approximately minimum cost), where the specification of the desired level of approximation is part of the problem's definition. Let us elaborate.

For concreteness, we focus on the case of a value that we wish to maximize. For greater flexibility, we allow the value of the solution to depend also on the instance itself. Thus, for a (polynomially bounded) binary relation R and a *value function* $f : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$, we consider the problem of finding solutions (with respect to R) that maximize the value of f. That is, given x (such that $R(x) \neq \emptyset$), the task is finding $y \in R(x)$ such that $f(x,y) = v_x$, where v_x is the maximum value of $f(x,y')$ over all $y' \in R(x)$. Typically, R is in $\mathcal{PC}$ and f is polynomial-time computable.[1] Indeed, without loss of generality, we may assume that for every x it holds that $R(x) = \{0,1\}^{\ell(|x|)}$ for some polynomial ℓ. Thus, the optimization problem is recast as *given x, find y such that $f(x,y) = v_x$, where $v_x = \max_{y' \in \{0,1\}^{\ell(|x|)}}\{f(x,y')\}$*.

We shall focus on relative approximation problems, where for some gap function $g : \{0,1\} \to \{r \in \mathbb{R} : r \geq 1\}$ the task is finding y such that $f(x,y) \geq v_x/g(x)$. Indeed, often the approximation factor is stated in terms of some structure of the input (e.g., the number of vertices in a graph) and not merely in terms of its length. Typically, g is polynomial-time computable. Approximation versions of minimization problems are defined analogously.

Definition 1 (g-factor approximation): *Let $f : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$, $\ell : \mathbb{N} \to \mathbb{N}$, and $g : \{0,1\} \to \{r \in \mathbb{R} : r \geq 1\}$.*

Maximization version: *The g-factor approximation of maximizing f (w.r.t ℓ) is the search problem R such that $R(x) = \{y \in \{0,1\}^{\ell(|x|)} : f(x,y) \geq v_x/g(x)\}$, where $v_x = \max_{y' \in \{0,1\}^{\ell(|x|)}}\{f(x,y')\}$.*

Minimization version: *The g-factor approximation of minimizing f (w.r.t ℓ) is the search problem R such that $R(x) = \{y \in \{0,1\}^{\ell(|x|)} : f(x,y) \leq g(x) \cdot c_x\}$, where $c_x = \min_{y' \in \{0,1\}^{\ell(|x|)}}\{f(x,y')\}$.*

We note that for numerous NP-complete optimization problems polynomial-time algorithms provide meaningful approximations. A few examples will be mentioned in Section 1.1. In contrast, for numerous other NP-complete optimization problems, natural approximation problems are computationally equivalent to the corresponding optimization problem. A few examples will be mentioned in Section 1.2, where we also introduce the notion of a *gap problem*, which is a promise problem (of the decision type) intended to capture the difficulty of the (approximate) search problem.

1.1 A few positive examples

Let us start with a trivial example. Considering a problem such as finding the maximum clique in a graph, we note that finding a linear factor approximation is trivial (i.e., given a graph $G = (V,E)$, we may output any vertex in V as a $|V|$-factor approximation of the maximum clique in G). A famous non-trivial example is presented next.

[1] In this case, we may assume without loss of generality that the function f depends only on the solution. This can be obtained by redefining the relation R such that each solution $y \in R(x)$ consists of a pair of the form (x,y'). Needless to say, this modification cannot be applied along with getting rid of R.

Proposition 2 (factor two approximation to `minimum Vertex Cover`): *There exists a polynomial-time approximation algorithm that given a graph $G = (V, E)$ outputs a vertex cover that is at most twice as large as the minimum vertex cover of G.*

We warn that an approximation algorithm for `minimum Vertex Cover` does not yield such an algorithm for the complementary problem (of `maximum Independent Set`). This phenomenon stands in contrast to the case of optimization, where an optimal solution for one problem (e.g., `minimum Vertex Cover`) yields an optimal solution for the complementary problem (`maximum Independent Set`).

Proof Sketch: The main observation is a connection between the set of maximal matchings and the set of vertex covers in a graph. Let M be any maximal matching in the graph $G = (V, E)$. Then, on one hand, the set of all vertices participating in M is a vertex cover of G, and, on the other hand, each vertex cover of G must contain at least one vertex of each edge of M. Thus, we can find the desired vertex cover by finding a maximal matching, which in turn can be found by a greedy algorithm. $\square$

Another example. An instance of the traveling salesman problem (`TSP`) consists of a matrix of distances between pairs of points and the task is finding a shortest tour that passes through all points. In general, no reasonable approximation is feasible for this problem (see Exercise 14), but here we consider two special cases in which the distances satisfies some natural constraints.

Theorem 3 (approximations to special cases of `TSP`): *Polynomial-time algorithms exists for the following two cases.*

1. *Providing a 1.5-factor approximation for the special case of TSP in which the distances satisfy the triangle inequality.*
2. *For every $\varepsilon > 1$, providing a $(1 + \varepsilon)$-factor approximation for the special case of Euclidean TSP* (i.e., the distances correspond to a k-dimensional Euclidean space, for some constant k (e.g., $k = 2$)).

A weaker version of Part 1 is given in Exercise 15. A detailed survey of Part 2 is provided in [1].

1.2 A few negative examples

Let us start again with a trivial example. Considering a problem such as finding the maximum clique in a graph, we note that given a graph $G = (V, E)$ finding a $(1+|V|^{-1})$-factor approximation of the maximum clique in G is as hard as finding a maximum clique in G. Indeed, this "result" is not really meaningful. In contrast, building on the PCP Theorem, one may prove that finding a $|V|^{1-o(1)}$-factor approximation of the maximum clique in G is as hard as finding a maximum clique in G. This follows from the fact that the approximation problem is NP-hard (cf. Theorem 5).

The statement of inapproximability results is made stronger by referring to a promise problem that consists of distinguishing instances of sufficiently far apart values. Such promise problems are called gap problems, and are typically stated with respect to two bounding functions $g_1, g_2 : \{0,1\} \to \mathbb{R}$ (which replace the gap g of Definition 1). Typically, g_1 and g_2 are polynomial-time computable.

Definition 4 (gap problem for approximation of f): *Let f be as in Definition 1 and $g_1, g_2 : \{0,1\} \to \mathbb{R}$.*

Maximization version: *For* $g_1 \geq g_2$*, the* gap_{g_1,g_2} problem of maximizing f *consists of distinguishing between* $\{x : v_x \geq g_1(x)\}$ *and* $\{x : v_x < g_2(x)\}$*, where* $v_x = \max_{y \in \{0,1\}^{\ell(|x|)}}\{f(x,y)\}$.

Minimization version: *For* $g_1 \leq g_2$*, the* gap_{g_1,g_2} problem of minimizing f *consists of distinguishing between* $\{x : c_x \leq g_1(x)\}$ *and* $\{x : c_x > g_2(x)\}$*, where* $c_x = \min_{y \in \{0,1\}^{\ell(|x|)}}\{f(x,y)\}$.

For example, the gap_{g_1,g_2} problem of maximizing the size of a clique in a graph consists of distinguishing between graphs G that have a clique of size $g_1(G)$ and graphs G that have no clique of size $g_2(G)$. In this case, we typically let $g_i(G)$ be a function of the number of vertices in $G=(V,E)$; that is, $g_i(G) = g'_i(|V|)$. Indeed, letting $\omega(G)$ denote the size of the largest clique in the graph G, we let $\texttt{gapClique}_{L,s}$ denote the gap problem of distinguishing between $\{G=(V,E) : \omega(G) \geq L(|V|)\}$ and $\{G=(V,E) : \omega(G) < s(|V|)\}$, where $L \geq s$. Using this terminology, we restate (and strengthen) the aforementioned $|V|^{1-o(1)}$-factor inapproximation of the maximum clique problem.

Theorem 5 *For some* $L(N) = N^{1-o(1)}$ *and* $s(N) = N^{o(1)}$*, it holds that* $\texttt{gapClique}_{L,s}$ *is NP-hard.*

As we shall show next, results of the type of Theorem 5 imply the hardness of a corresponding approximation problem; that is, the hardness of deciding a gap problem implies the hardness of a search problem that refers to an analogous factor of approximation.

Proposition 6 *Let* f, g_1, g_2 *be as in Definition 4 and suppose that these functions are polynomial-time computable. Then the* gap_{g_1,g_2} *problem of maximizing* f (resp., minimizing f) *is reducible to the* g_1/g_2*-factor* (resp., g_2/g_1-factor) *approximation of maximizing* f (resp., minimizing f).

Note that a reduction in the opposite direction does not necessarily exist (even in the case that the underlying optimization problem is self-reducible in some natural sense). Indeed, this is another difference between the current context (of approximation) and the context of optimization problems, where the search problem is reducible to a related decision problem.

Proof Sketch: We focus on the maximization version. On input x, we solve the gap_{g_1,g_2} problem, by making the query x, obtaining the answer y, and ruling that x has value exceeding $g_1(x)$ if and only if $f(x,y) \geq g_2(x)$. Recall that we need to analyze this reduction only on inputs that satisfy the promise. Thus, if $v_x \geq g_1(x)$ then the oracle must return a solution y that satisfies $f(x,y) \geq v_x/(g_1(x)/g_2(x))$, which implies that $f(x,y) \geq g_2(x)$. On the other hand, if $v_x < g_2(x)$ then $f(x,y) \leq v_x < g_2(x)$ holds for any possible solution y. $\square$

Additional examples. Let us consider $\texttt{gapVC}_{s,L}$, the gap_{g_s,g_L} problem of minimizing the vertex cover of a graph, where s and L are constants and $g_s(G) = s \cdot |V|$ (resp., $g_L(G) = L \cdot |V|$) for any graph $G=(V,E)$. Then, Proposition 2 implies (via Proposition 6) that, for every constant s, the problem $\texttt{gapVC}_{s,2s}$ is solvable in polynomial-time. In contrast, sufficiently narrowing the gap between the two thresholds yields an inapproximability result. In particular:

Theorem 7 *For some constants* $s < L$ (e.g., $s = 0.62$ and $L = 0.84$ will do)*, the problem* $\texttt{gapVC}_{s,L}$ *is NP-hard.*

As stated in [12, Text 16], the PCP Theorem and versions of it play a key role in establishing inapproximability results such as Theorems 5 and 7. In particular, recall the equivalence of the PCP Theorem itself and the NP-hardness of a gap problem concerning the maximization of the number of clauses that are satisfies in a given 3-CNF formula. Specifically, $\texttt{gapSAT}^3_\varepsilon$ was defined (in

[12, Text 16]) as the gap problem consisting of distinguishing between satisfiable 3-CNF formulae and 3-CNF formulae for which each truth assignment violates at least an ε fraction of the clauses. Although the aforementioned equivalence result does not specify the quantitative relation that underlies its qualitative assertion, when combined with the best known PCP construction, it does yield the best possible bound.

Theorem 8 *For every $v < 1/8$, the problem* $\mathtt{gapSAT}^3_v$ *is NP-hard.*

On the other hand, $\mathtt{gapSAT}^3_{1/8}$ is solvable in polynomial-time.

Sharp threshold. The aforementioned results regarding $\mathtt{gapSAT}^3_v$ exemplify a sharp threshold on the feasibly obtainable approximation factor. Another appealing example refers to the following maximization problem in which the instances are systems of linear equations over GF(2) and the task is finding an assignment that satisfies as many equations as possible. Note that by merely selecting an assignment at random, we expect to satisfy half of the equations. Also note that it is easy to determine whether there exists an assignment that satisfies all equations. Let $\mathtt{gapLin}_{L,s}$ denote the problem of distinguishing between systems in which one can satisfy at least an L fraction of the equations and systems in which one cannot satisfy an s fraction (or more) of the equations. Then, as just noted, $\mathtt{gapLin}_{L,0.5}$ is trivial and $\mathtt{gapLin}_{1,s}$ is feasible (for every $s < 1$). In contrast, moving both thresholds towards one another yields an NP-hard gap problem:

Theorem 9 *For every constant $\varepsilon > 0$, the problem* $\mathtt{gapLin}_{1-\varepsilon,0.5+\varepsilon}$ *is NP-hard.*

Gap location. Theorems 8 and 9 illustrate two opposite situations. In both cases there is an obvious upper-bound on the number of local conditions that can be satisfies (i.e., the number of clauses in the case of gapSAT and the number of equations in the case of gapLin). In one case (i.e., gapSAT) deciding whether an instance has a value that attains the upper-bound is NP-hard, whereas in the other case (i.e., gapLin) this decision problem is solvable in polynomial-time. Consequently, in the second case, the gap problem in which the higher threshold equals this upper-bound is easy (i.e., $\mathtt{gapLin}_{1,s}$ is in P), whereas in the first case the corresponding gap problem may be hard. Indeed, $\mathtt{gapSAT}_\varepsilon$ refers to distinguishing instances attaining the upper-bound from instances that have a value that is upper-bounded by a lower threshold (i.e., the $1-\varepsilon$ fraction), and in fact $\mathtt{gapSAT}_\varepsilon$ is NP-hard.

A final comment. All the aforementioned inapproximability results refer to approximation (resp., gap) problems that are relaxations of optimization problems in NP (i.e., the optimization problem is computational equivalent to a decision problem in $\mathcal{NP}$). In these cases, the NP-hardness of the approximation (resp., gap) problem implies that the corresponding optimization problem is reducible to the approximation (resp., gap) problem. In other words, in these cases nothing is gained by relaxing the original optimization problem, since the relaxed version remains just as hard.

2 Decision or Property Testing

A natural notion of relaxation for decision problems arises when considering the distance between instances, where a natural notion of distance is the Hamming distance (i.e., fraction of bits on which two strings disagree). Loosely speaking, this relaxation (called *property testing*) refers to distinguishing inputs that reside in a predetermined set S from inputs that are "relatively far" from any input that resides in the set. Two natural types of promise problems emerge (with respect to the predetermined set S and the Hamming distance between strings):

1. *Relaxed decision w.r.t a fixed distance*: Fixing a distance parameter δ, we consider the problem of distinguishing inputs in S from inputs in $\Gamma_\delta(S)$, where

$$\Gamma_\delta(S) \stackrel{\text{def}}{=} \{x : \forall z \in S \cap \{0,1\}^{|x|} \ \ \Delta(x,z) > \delta \cdot |x|\} \tag{1}$$

   and $\Delta(x_1 \cdots x_m, z_1 \cdots z_m) = |\{i : x_i \neq z_i\}|$ denotes the number of bits on which $x = x_1 \cdots x_m$ and $z = z_1 \cdots z_m$ disagree. Thus, here we consider a promise problem that is a restriction (or a special case) of the problem of deciding membership in S.

2. *Relaxed decision w.r.t a variable distance*: Here instances are pairs (x, δ), where x is as in Type 1 and δ is a distance parameter. The yes-instances are pairs (x, δ) such that $x \in S$, whereas (x, δ) is a no-instance if $x \in \Gamma_\delta(S)$.

We shall focus on Type 1 formulation, which seems to capture the essential question of whether or not these relaxations lower the complexity of the original decision problem. The study of Type 2 formulation refers to a relatively secondary question, which assumes a positive answer to the first question; that is, assuming that the relaxed form is easier than the original form, we ask how is the complexity of the problem affected by making the distance parameter smaller (which means making the relaxed problem "tighter" and ultimately equivalent to the original problem).

We note that for numerous NP-complete problems there exist natural (Type 1) relaxations that are solvable in polynomial-time. Actually, these algorithms run in *sub-linear* time (specifically polylogarithmic time), when given direct access to the input. A few examples will be presented in Section 2.2. As indicated in Section 2.2, this is not a generic phenomenon. We start by discussing several key definitional issues (see Section 2.1).

2.1 Definitional issues

Property testing is concerned not only with solving relaxed versions of NP-hard problems, but rather solving these problems (as well as problems in $\mathcal{P}$) in *sub-linear time*. Needless to say, such results assume a model of computation in which algorithms have direct access to bits in the (representation of the) input (see Definition 10).

Definition 10 (a direct access model – conventions): *An algorithm with* direct access to its input *is given its* main input *on a special input device that is accessed as an oracle. In addition, the algorithm is given the length of the input and possibly other parameters on an* secondary input device. *The complexity of such an algorithm is stated in terms of the length of its main input.*

Definition 11 (property testing for S): *The promise problem of distinguishing S from $\Gamma_\delta(S)$ is called* property testing for S (with respect to δ).

Recall that we say that a randomized algorithm solves a promise problem if it accepts every yes-instance (resp., rejects every no-instance) with probability at least 2/3. Thus, a (randomized) property testing for S accepts every input in S (resp., rejects every input in $\Gamma_\delta(S)$) with probability at least 2/3.

The question of representation. The specific representation of the input is of major concern in the current context. Firstly, the representation affects the distance measure (e.g., under an artificially padded input format, all strings will be deemed close to one another and property testing for S will become trivial). Secondly, since our focus is on sub-linear time algorithms, we cannot afford to transform the input from one natural format to another. Both issues will be clarified by the examples provided in Section 2.2.

The essential role of the promise. Recall that, for a fixed constant $\delta > 0$, we consider the promise problem of distinguishing S from $\Gamma_\delta(S)$. The promise means that all instances that are neither in S nor far from S (i.e., not in $\Gamma_\delta(S)$) are ignored, which is essential for sub-linear algorithms for natural problems. To demonstrate the point, we say that a set S is reasonably sensitive if for infinitely many $x \in \{0,1\}^* \cap S$ and every $I \subset [|x|]$ of cardinality $|x|/3$ there exists $z \in \{0,1\}^{|x|} \setminus S$ such that for every $i \in I$ it holds that $x_i = z_i$. We note that if S is reasonably sensitive then *deciding membership* in S requires linear time (even in the model of Definition 10). On the other hand, even for a reasonably sensitive set S, *property testing* for S (i.e., distinguishing S from $\Gamma_\delta(S)$) may be done in sub-linear (randomized) time. Consider, for example, the set S consisting of strings that have a majority of 1's. Then, S is reasonably sensitive, but the fraction of 1's in the input can be approximated in polylogarithmic time (which yields a property tester for S).

The essential role of randomization. Referring to the foregoing example, we note that randomization is essential for any sub-linear time algorithm that distinguishes this set S from, say, $\Gamma_{0.4}(S)$. In contrast, a sub-linear time deterministic algorithm cannot distinguish 1^n from any input that has 1's in each position probed by the algorithm on input 1^n. In general, on input x, a (sub-linear time) deterministic algorithm always reads the same bits of x and thus cannot distinguish x from any z that agrees with x on these bit locations.

Note that, in both cases, we are able to prove lower-bounds on the time complexity of algorithms. But these lower-bounds are actually information theoretic in nature and refer to the number of queries performed by these algorithms.

2.2 Two models for testing graph properties

In this section we consider the complexity of property testing for sets of graphs that are *closed under graph isomorphism*; such sets are called graph properties. In view of the foregoing comment, referring to the importance of representation, we consider two standard representations of graphs.

1. The adjacency matrix representation. Here a graph $G = ([N], E)$ is represented (in a slightly redundant form) by an N-by-N Boolean matrix $M_G = (m_{i,j})_{i,j\in[N]}$ such that $m_{i,j} = 1$ if and only if $\{i, j\} \in E$.

2. Bounded incidence-lists representation. For a fixed parameter d, a graph $G = ([N], E)$ of degree at most d is represented (in a slightly redundant form) by a mapping $\mu_G : [N] \times [d] \to [N] \cup \{\perp\}$ such that $\mu_G(u, i) = v$ if v is the i^{th} neighbor of u and $\mu_G(u, i) = \perp$ if v has less than i neighbors.

We stress that the aforementioned representations determine both the notion of distance between graphs and the type of queries performed by the algorithm. As we shall see, the difference between these two representations yields a big difference on the complexity of corresponding property testing problems.

Theorem 12 (property testing in the adjacency matrix representation): *For any fixed $\delta > 0$ and each of the following sets, there exists a polylogarithmic time randomized algorithm that solves the corresponding property testing problem.*

- *The set of k-colorable graphs, for every fixed $k \geq 2$.*
- *The set of graphs having a clique* (resp. independent set) *of density ρ, for every fixed $\rho > 0$.*
- *The set of N-vertex graphs having a cut*[2] *with at least $\rho \cdot N^2$ edges, for every fixed $\rho > 0$.*
- *The set of N-vertex graphs having a bisection*[2] *with at most $\rho \cdot N^2$ edges, for every fixed $\rho > 0$.*

In contrast, for some $\delta > 0$, there exists a graph property in $\mathcal{NP}$ for which property testing requires linear time.

The algorithms use a constant number of queries, which in turn is polynomial in the constant $1/\delta$, and their running time hides a constant that is exponential in their query complexity, except for the case of 2-colorability where the hidden constant is polynomial in $1/\delta$. Note that such dependencies seem essential, since setting $\delta = 1/N^2$ regains the original (non-relaxed) decision problems (which, with the exception of 2-colorability, are all NP-complete). Again, the lower-bound on the time complexity follows from a lower-bound on the query complexity. We note that the graph property for which this lower-bound is proved is not a natural one.

Theorem 12 exhibits a dichotomy graph properties for which property testing is possible by a constant number of queries and graph properties for which property testing requires a linear number of queries. A combinatorial characterization of the graph properties for which property testing is possible (in the adjacency matrix representation) using a constant number of queries is known.[3] We note that the constant in this characterization may depend arbitrarily on δ (and indeed, in some cases, it a function growing faster than a tower of $1/\delta$ exponents).

Turning back to Theorem 12, we note that the results regarding property testing for the sets corresponding to max-cut and min-bisection yield approximation algorithms with an additive error term (of δN^2). For dense graphs (i.e., N-vertex graphs having $\Omega(N^2)$ edges), this yields a constant factor approximation for the standard approximation problem (as in Definition 1). That is, for every constant $c > 1$, we obtain a c-factor approximation of maximizing the size of a cut (resp., minimizing the size of the bisection) in dense graphs. On the other hand, the result regarding clique yields a so called dual-approximation for maximum clique; that is, we approximate the minimum number of missing edges in the densest induced graph of a given size.

Indeed, Theorem 12 is meaningful only for dense graphs. The same holds, in general, for the adjacency matrix representation.[4] Also note that for sets S satisfying $\Gamma_\delta(S) = \emptyset$ under the adjacency matrix representation (e.g., the set of connected graphs) property testing is trivial.

We now turn to the bounded incidence-lists representation, which is relevant only for bounded degree graphs. The problems of max-cut, min-bisection and clique (as in Theorem 12) are trivial under this representation, but graph connectivity becomes non-trivial and the complexity of property testing for the set of bipartite graphs changes dramatically.

[2] A cut in a graph $G = ([N], E)$ is a partition $(S, [N] \setminus S)$ of the set of vertices and the edges of the cut are the edges with exactly one endpoint in S. A bisection is a cut of the graph to two parts of equal cardinality.

[3] Describing this fascinating result of Alon *et. al.* [4], which refers to the notion of regular partitions (introduced by Szemerédi), is beyond the scope of the current text.

[4] In this model, all N-vertex graphs having less than δN^2 edges may be accepted if and only if there exists such a (non-dense) graph in the predetermined set. This trivial algorithm is correct because if set S contains an N-vertex graph with less than δN^2 then $\Gamma_\delta(S) = \emptyset$.

Theorem 13 (property testing in the bounded incidence-lists representation): *The following assertions refer to representations of graphs of degree at most d.*

- *For any fixed d and $\delta > 0$, there exists a polylogarithmic time randomized algorithm that solves the property testing problem for the set of connected graphs of degree at most d.*
- *For any fixed d and $\delta > 0$, there exists a sub-linear randomized algorithm that solves the property testing problem for the set of bipartite graphs of degree at most d. Specifically, on input an N-vertex graph, the algorithm runs for $\widetilde{O}(\sqrt{N})$ time.*
- *For any fixed $d \geq 3$ and some $\delta > 0$, property testing for the set of N-vertex* (3-regular) *bipartite graphs requires $\Omega(\sqrt{N})$ queries.*
- *For some fixed d and $\delta > 0$, property testing for the set of N-vertex 3-colorable graphs requires $\Omega(N)$ queries.*

The running time of the algorithms hides a constant that is polynomial in $1/\delta$. Providing a characterization of graph properties according to the complexity of the corresponding tester (in the bounded incidence-lists representation) is an interesting open problem.

Decoupling the distance from the representation. So far we have confined our attention to the Hamming distance between the representations of graphs. This made the choice of representation even more important than usual (i.e., more crucial than is common in complexity theory). In contrast, it is natural to consider a notion of distance between graphs that is independent of their representation. For example, the distance between $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ can be defined as the size of symmetric difference between E_1 and the set of edges in a graph that is isomorphic to G_2. The corresponding relative distance may be defined as the distance divided by $|E_1|+|E_2|$ (or by $\max(|E_1|,|E_2|)$).

2.3 Beyond graph properties

Property testing has been applied to a variety of computational problems beyond the domain of graph theory. In fact, this area first emerged in the algebraic domain, where the instances (to be viewed as inputs to the testing algorithm) are functions and the relevant properties are sets of algebraic functions. The archetypical example is the set of low-degree polynomials; that is, m-variate polynomials of total (or individual) degree d over some finite field $\mathrm{GF}(q)$, where m,d and q are functions of the length of the input (which being the full description of a m-variate function over $\mathrm{GF}(q)$, has length $q^m \cdot \log_2 q$).

Viewing the problem instance as a function suggests a natural measure of distance (i.e., the fraction of arguments on which the functions disagree) as well as a natural way of accessing the instance (i.e., querying the function for the value of selected arguments). However, as in Section 2.2, we may decouple the distance measure from the representation (i.e., a way of accessing the problem instance). This is done by introducing a representation-independent notion of distance between instances, which should be natural in the context of the problem at hand.

Notes

The following bibliographic comments are quite laconic and neglect mentioning various important works (including credits for some of the results mentioned in our text). As usual, the interested reader is referred to corresponding surveys.

Search or Optimization. The interest in approximation algorithms increased considerably following the demonstration of the NP-completeness of many natural optimization problems. But, with some exceptions (most notably [20]), the systematic study of the complexity of such problems stalled till the discovery of the "PCP connection" by Feige, Goldwasser, Lovász, and Safra [9]. Indeed the relatively "tight" inapproximation results for max-Clique, max-SAT, and the maximization of linear equations, due to Håstad [16, 17], build on previous work regarding PCP and their connection to approximation (cf., e.g., [10, 3, 2, 5, 21]). Specifically, Theorem 5 is due to [16], while Theorems 8 and 9 are due to [17]. The best known inapproximation result for minimum Vertex Cover (see Theorem 7) is due to [8], but we doubt it is tight (see, e.g., [19]). Reductions among approximation problems were defined and presented in [20]; see Exercise 16, which presents a major technique introduced in [20]. For general texts on approximation algorithms and problems (as discussed in Section 1), the interested reader is referred to the surveys collected in [18]. A compendium of NP optimization problems is available at [7].

Property testing. The study of property testing was initiated by Rubinfeld and Sudan [23] and re-initiated by Goldreich, Goldwasser, and Ron [13]. While the focus of [23] was on algebraic properties such as low-degree polynomials, the focus of [13] was on graph properties (and Theorem 12 is taken from [13]). The model of bounded-degree graphs was introduced in [14] and Theorem 13 combines results from [14, 15, 6]. For surveys of the area, the interested reader is referred to [11, 22].

Exercises

Exercise 14 (general TSP) For any function g, prove that the following approximation problem is NP-Hard. Given a general TSP instance I, the task is finding a tour of length that is at most a factor $g(I)$ of the minimum. In the case that all distances are required to be positive, show that the result holds with $g(I) = \exp(\text{poly}(|I|))$.

Guideline: By reduction from Hamiltonian path. Specifically, reduce the instance $G = ([n], E)$ to an n-by-n distance matrix $D = (d_{i,j})_{i,j\in[n]}$ such that $d_{i,j} = 1$ if $\{i, j\} \in E$ and $d_{i,j} = 0$ otherwise. In case that all distances are required to be positive, we may set $d_{i,j} = \exp(\text{poly}(n))$ if $\{i, j\} \in E$ and $d_{i,j} = 1$.

Exercise 15 (TSP with triangle inequalities) Provide a polynomial-time 2-factor approximation for the special case of TSP in which the distances satisfy the triangle inequality.

Guideline: First note that the length of any tour is lower-bounded by the weight of a minimum spanning tree in the corresponding weighted graph. Next note that such a tree yields a tour (of length twice the weight of this tree) that may visit some points several times. The triangle inequality guarantees that the tour does not become longer by "shortcuts" that eliminate multiple visits at the same point.

Exercise 16 (enforcing multi-way equalities via expanders) The aim of this exercise is presenting a major technique of Papadimitriou and Yannakakis [20], which is useful for designing reductions among approximation problems. Recalling that $\mathtt{gapSAT}^3_{0.1}$ is NP-hard, our goal is proving NP-hard of the following gap problem, denoted $\mathtt{gapSAT}^{3,c}_{\varepsilon}$, which is a special case of $\mathtt{gapSAT}^3_{\varepsilon}$. Specifically, the instances are restricted to 3CNF formulae with each variable appearing in at most c clauses, where c (as ε) is a fixed constant. Note that the standard reduction of 3SAT to the corresponding special case (see [12, Text 14]) does not preserve an approximation gap.[5] The idea is

[5]Recall that in this reduction each occurrence of each Boolean variable is replaced by a new copy of this variable, and clauses are added for enforcing the assignment of the same value to all these copies. Specifically, the m occurrence

enforcing equality of the values assigned to the auxiliary variables (i.e., the copies of each original variable) by introducing equality constraints only for pairs of variables that correspond to edges of an expander graph. For example, we enforce equality among the values of $z^{(1)},...,z^{(m)}$ by adding the clauses $z^{(i)} \vee \neg z^{(j)}$ for every $\{i,j\} \in E$, where E is the set of edges of a m-vertex expander graph. Prove that, for some constants c and $\varepsilon > 0$, the corresponding mapping reduces $\texttt{gapSAT}^3_{0.1}$ to $\texttt{gapSAT}^{3,c}_{\varepsilon}$.

Guideline: Using a d-regular expander, we map 3CNF to instances in which each variable appears in at most $2d+1$ clauses. Note that if the original formula is satisfiable then so is the reduced one. On the other hand, consider an arbitrary assignment τ' to the reduced formula ϕ' (i.e., the formula obtained by mapping ϕ). For each original variable z, if τ' assigns the same value to almost all copies of z then we consider the corresponding assignment in ϕ. Otherwise, τ' does not satisfy a constant fraction of the clauses containing a copy of z.

References

[1] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Math. Programming*, Vol. 97, pages 43–69, July 2003.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[3] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[4] N. Alon, E. Fischer, I. Newman, and A. Shapira. A Combinatorial Characterization of the Testable Graph Properties: It's All About Regularity. In *38th ACM Symposium on the Theory of Computing*, to appear, 2006.

[5] M. Bellare, O. Goldreich and M. Sudan. Free Bits, PCPs and Non-Approximability – Towards Tight Results. *SIAM Journal on Computing*, Vol. 27, No. 3, pages 804–915, 1998. Extended abstract in *36th FOCS*, 1995.

[6] A. Bogdanov, K. Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *43rd IEEE Symposium on Foundations of Computer Science*, pages 93–102, 2002.

[7] P. Crescenzi and V. Kann. A compendium of NP Optimization problems. Available at `http://www.nada.kth.se/~viggo/wwwcompendium/`

[8] I. Dinur and S. Safra. The importance of being biased. In *34th ACM Symposium on the Theory of Computing*, pages 33–42, 2002.

of variable z are replaced by the variables $z^{(1)},...,z^{(m)}$, while adding the clauses $z^{(i)} \vee \neg z^{(i+1)}$ and $z^{(i+1)} \vee \neg z^{(i)}$ (for $i = 1,...,m-1$). The problem is that almost all clauses of the reduced formula may be satisfied by an assignment in which half of the copies of each variable are assigned one value and the rest are assigned an opposite value. That is, an assignment in which $z^{(1)} = \cdots = z^{(i)} \neq z^{(i+1)} = \cdots = z^{(m)}$ violates only one of the auxiliary clauses introduced for enforcing equality among the copies of z. Using an alternative reduction that adds the clauses $z^{(i)} \vee \neg z^{(j)}$ for every $i,j \in [m]$ will not do either, because the number of added clauses may be quadratic in the number of original clauses.

[9] U. Feige, S. Goldwasser, L. Lovász and S. Safra. On the Complexity of Approximating the Maximum Size of a Clique. Unpublished manuscript, 1990.

[10] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *Journal of the ACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[11] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, Vol. 75, pages 97–126, 2001.

[12] O. Goldreich. Expositions in Complexity Theory (various texts). Unpublished notes, December 2005. Availabe from the webpage `http://www.wisdom.weizmann.ac.il/~oded/cc-texts.html`

[13] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998.

[14] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002.

[15] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, Vol. 19 (3), pages 335–373, 1999.

[16] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, Vol. 182, pages 105–142, 1999. Preliminary versions in *28th STOC* (1996) and *37th FOCS* (1996).

[17] J. Håstad. Getting optimal in-approximability results. In *29th ACM Symposium on the Theory of Computing*, pages 1–10, 1997.

[18] D. Hochbaum (ed.). *Approximation Algorithms for NP-Hard Problems.* PWS, 1996.

[19] S. Khot and O. Regev. Vertex Cover Might be Hard to Approximate to within $2-\varepsilon$. In *18th IEEE Conference on Computational Complexity*, pages 379–386, 2003.

[20] C.H. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. In *20th ACM Symposium on the Theory of Computing*, pages 229–234, 1988.

[21] R. Raz. A Parallel Repetition Theorem. *SIAM Journal on Computing*, Vol. 27 (3), pages 763–803, 1998. Extended abstract in *27th STOC*, 1995.

[22] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001. (Editors: S. Rajasekaran, P.M. Pardalos, J.H. Reif and J.D.P. Rolim.)

[23] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, Vol. 25 (2), pages 252–271, 1996.