Computational Complexity:

A Conceptual Perspective

Oded Goldreich

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

July 1, 2008

Chapter 10

Relaxing the Requirements

The philosophers have only interpreted the world, in various ways; the point is to change it.

Karl Marx, Theses on Feuerbach

In light of the apparent infeasibility of solving numerous useful computational problems, it is natural to ask whether these problems can be relaxed such that the relaxation is both useful and allows for feasible solving procedures. We stress two aspects about the foregoing question: on one hand, the relaxation should be sufficiently good for the intended applications; but, on the other hand, it should be significantly different from the original formulation of the problem so to escape the infeasibility of the latter. We note that whether a relaxation is adequate for an intended application depends on the application, and thus much of the material in this chapter is less robust (or generic) than the treatment of the non-relaxed computational problems.

Summary: We consider two types of relaxations. The first type of relaxation refers to the computational problems themselves; that is, for each problem instance we *extend the set of admissible solutions*. In the context of search problems this means settling for solutions that have a value that is "sufficiently close" to the value of the optimal solution (with respect to some value function). Needless to say, the specific meaning of 'sufficiently close' is part of the definition of the relaxed problem. In the context of decision problems this means that for some instances both answers are considered valid; specifically, we shall consider promise problems in which the no-instances are "far" from the yes-instances in some adequate sense (which is part of the definition of the relaxed problem).

The second type of relaxation deviates from the requirement that the solver provides an adequate answer on each valid instance. Instead, the behavior of the solver is analyzed with respect to a predetermined

input distribution (or a class of such distributions), and bad behavior may occur with negligible probability where the probability is taken over this input distribution. That is, we replace worst-case analysis by *average-case* (or rather *typical-case*) *analysis*. Needless to say, a major component in this approach is limiting the class of distributions in a way that, on one hand, allows for various types of natural distributions and, on the other hand, prevents the collapse of the corresponding notion of average-case hardness to the standard notion of worst-case hardness.

Organization. The first type of relaxation is treated in Section 10.1, where we consider approximations of search (or rather optimization) problems as well as approximate-decision problems (a.k.a property testing); see Section 10.1.1 and Section 10.1.2, respectively. The second type of relaxation, known as average/typical-case complexity, is treated in Section 10.2. The treatment of these two types is quite different. Section 10.1 provides a short and high-level introduction to various research areas, focusing on the main notions and illustrating them by reviewing some results (while providing no proofs). In contrast, Section 10.2 provides a basic treatment of a theory (of average/typical-case complexity), focusing on some basic results and providing a rather detailed exposition of the corresponding proofs.

10.1 Approximation

The notion of approximation is a very natural one, and has arisen also in other disciplines. Approximation is most commonly used 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 already encountered it as a modifier of counting problems; see Section 6.2.2.)

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 hand 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 *natural* approximation problems (e.g., general 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 will not provide a *general systematic classification* (because such a classification is not known).[1]

We shall distinguish between approximation problems that are of a "search type" and problems that 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 the distance between instances (of a decision problem).[2] We note that, sometimes 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. The common theme underlying both frameworks is that in each of them we extend the set of admissible solutions. In the case of search problems, we augment the set of optimal solutions by allowing also almost-optimal solutions. In the case of decision problems, we extend the set of solutions by allowing an arbitrary answer (solution) to some instances, which may be viewed as a promise problem that disallows these instances. In this case we focus on promise problems in which the yes- and no-instances are far apart (and the instances that violate the promise are closed to yes-instances).

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 in the context of complexity theory, we recommend doing the same in class.

10.1.1 Search or Optimization

As noted in Section 2.2.2, 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 expressibility (or, actually, for greater flexibility), we allow the value of the solution to depend also on the instance itself.[3] 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

[1] In contrast, systematic classifications of restricted classes of approximation problems are known. For example, see [56] for a classification of (approximate versions of) Constraint Satisfaction Problems.

[2] In some sense, this distinction is analogous to the distinction between the two aforementioned uses of the word *approximation*.

[3] This convention is only a matter of convenience: without loss of generality, we can express the same optimization problem using a value function that only depends on the solution by augmenting each solution with the corresponding instance (i.e., a solution y to an instance x can be encoded as a pair (x, y), and the resulting set of valid solutions for x will consist of pairs of the form $(x, \cdot)$). Hence, the foregoing convention merely allows avoiding this cumbersome encoding of solutions.

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. 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 (see Exercise 2.8).[4] Thus, the optimization problem is recast as the following search problem: *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 (maximization) task is finding y such that $f(x,y) \geq v_x/g(x)$. Indeed, in some cases the approximation factor is stated as a function of the length of the input (i.e., $g(x) = g'(|x|)$ for some $g' : \mathbb{N} \to \{r \in \mathbb{R} : r \geq 1\}$), but often the approximation factor is stated in terms of some more refined parameter of the input (e.g., as a function of the number of vertices in a graph). Typically, g is polynomial-time computable.

Definition 10.1 (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 §10.1.1.1. 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 §10.1.1.2, 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.

10.1.1.1 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.

Proposition 10.2 (factor two approximation to `minimum Vertex Cover`): *There exists a polynomial-time approximation algorithm that given a graph $G = (V,E)$*

[4] However, in this case (and in contrast to Footnote 3), the value function f must depend both on the instance and on the solution (i.e., $f(x,y)$ may no be oblivious of x).

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 search problem (of `maximum Independent Set`). This phenomenon stands in contrast to the case of optimization, where an optimal solution for one search problem (e.g., `minimum Vertex Cover`) yields an optimal solution for the complementary search 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 $G = (V, E)$; that is, $M \subseteq E$ is a matching but augmenting it by any single edge yields a set that is not a matching. 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. □

Another example. An instance of the traveling salesman problem (`TSP`) consists of a symmetric 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 10.1), but here we consider two special cases in which the distances satisfy some natural constraints (and pretty good approximations are feasible).

Theorem 10.3 (approximations to special cases of `TSP`): *Polynomial-time algorithms exist for the following two computational problems.*

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., for some constant k (e.g., $k = 2$), the points reside in a k-dimensional Euclidean space, and the distances refer to the standard Euclidean norm).

A weaker version of Part 1 is given in Exercise 10.2. A detailed survey of Part 2 is provided in [13]. We note the difference exemplified by the two items of Theorem 10.3: Whereas Part 1 provides a polynomial-time approximation for a specific constant factor, Part 2 provides such an algorithm for any constant factor. Such a result is called a *polynomial-time approximation scheme* (abbreviated PTAS).

10.1.1.2 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 (Theorem 9.16), one may prove that finding a $|V|^{1-o(1)}$-factor approximation of the maximum clique in a general graph $G = (V, E)$ is as hard as finding a maximum clique in a general graph. This follows from the fact that the approximation problem is NP-hard (cf. Theorem 10.5).

The statement of such 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 function g of Definition 10.1). Typically, g_1 and g_2 are polynomial-time computable.

Definition 10.4 (gap problem for approximation of f): *Let f be as in Definition 10.1 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 inapproximability result of the maximum clique problem.

Theorem 10.5 *For some $L(N) = N^{1-o(1)}$ and $s(N) = N^{o(1)}$, it holds that* $\texttt{gapClique}_{L,s}$ *is NP-hard.*

The proof of Theorem 10.5 is based on a major refinement of Theorem 9.16 that refers to a PCP system of amortized free-bit complexity that tends to zero (cf. §9.3.4.1). A weaker result, which follows from Theorem 9.16 itself, is presented in Exercise 10.3.

As we shall show next, results of the type of Theorem 10.5 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 10.6 *Let f, g_1, g_2 be as in Definition 10.4 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 at least $g_1(x)$ 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. □

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 10.2 implies (via Proposition 10.6) 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 10.7 *For some constants $s > 0$ and $L < 1$ such that $L > \frac{4}{3} \cdot s$* (e.g., $s = 0.62$ and $L = 0.84$), *the problem* $\texttt{gapVC}_{s,L}$ *is NP-hard.*

The proof of Theorem 10.7 is based on a complicated refinement of Theorem 9.16. Again, a weaker result follows from Theorem 9.16 itself (see Exercise 10.4).

As noted, refinements of the PCP Theorem (Theorem 9.16) play a key role in establishing inapproximability results such as Theorems 10.5 and 10.7. In that respect, it is adequate to recall that Theorem 9.21 establishes 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 Definition 9.20) 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 Theorem 9.21 does not specify the quantitative relation that underlies its qualitative assertion, when (refined and) combined with the best known PCP construction, it does yield the best possible bound.

Theorem 10.8 *For every $v < 1/8$, the problem* $\texttt{gapSAT}^3_v$ *is NP-hard.*

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

Sharp thresholds. The aforementioned opposite results (regarding $\mathtt{gapSAT}_v^3$) exemplify a sharp threshold on the (factor of) approximation that can be obtained by an efficient algorithm. 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 (for every $L \geq 0.5$) and $\mathtt{gapLin}_{1,s}$ is feasible (for every $s < 1$). In contrast, moving both thresholds (slightly) away from the corresponding extremes, yields an NP-hard gap problem:

Theorem 10.9 *For every constant $\varepsilon > 0$, the problem $\mathtt{gapLin}_{1-\varepsilon,0.5+\varepsilon}$ is NP-hard.*

The proof of Theorem 10.9 is based on a major refinement of Theorem 9.16. In fact, the corresponding PCP system (for NP) is merely a reformulation of Theorem 10.9: the verifier makes three queries and tests a linear condition regarding the answers, while using a logarithmic number of coin tosses. This verifier accepts any yes-instance with probability at least $1 - \varepsilon$ (when given oracle access to a suitable proof), and rejects any no-instance with probability at least $0.5 - \varepsilon$ (regardless of the oracle being accessed). A weaker result, which follows from Theorem 9.16 itself, is presented in Exercise 10.5.

Gap location. Theorems 10.8 and 10.9 illustrate two opposite situations with respect to the "location" of the "gap" for which the corresponding promise problem is hard. Recall that both gapSAT and gapLin are formulated with respect to two thresholds, where each threshold bounds the fraction of "local" conditions (i.e., clauses or equations) that are satisfiable in the case of yes- and no-instances, respectively. In the case of gapSAT, the high threshold (referring to yes-instances) was set to 1, and thus only the low threshold (referring to no-instances) remained a free parameter. Nevertheless, a hardness result was established for gapSAT, and furthermore this was achieved for an optimal value of the low threshold (cf. the foregoing discussion of sharp thresholds). In contrast, in the case of gapLin, setting the high threshold to 1 makes the gap problem efficiently solvable. Thus, the hardness of gapLin was established at a different location of the high threshold. Specifically, hardness (for an optimal value of the ratio of thresholds) was established when setting the high threshold to $1 - \varepsilon$, for any $\varepsilon > 0$.

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 computationally equivalent to a decision problem in $\mathcal{NP}$; see Section 2.2.2). 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, because the relaxed version remains just as hard.

10.1.2 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., the 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 any predetermined set S (and the Hamming distance between strings)):

1. *Relaxed decision w.r.t a fixed relative distance*: Fixing a distance parameter δ, we consider the problem of distinguishing inputs in S from inputs in $\Gamma_\delta(S)$, where
$$\Gamma_\delta(S) \stackrel{\text{def}}{=} \{x : \forall z \in S \cap \{0,1\}^{|x|}\ \ \Delta(x,z) > \delta \cdot |x|\} \tag{10.1}$$
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 the instances are pairs (x, δ), where x is as in Type 1 and $\delta \in [0, 1]$ is a (relative) 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, in polylogarithmic time), when given direct access to the input. A few examples will be presented in §10.1.2.2 (but, as indicated in §10.1.2.2, this is not a generic phenomenon). Before turning to these examples, we discuss several important definitional issues.

10.1.2.1 Definitional issues

Property testing is concerned not only with solving relaxed versions of NP-hard problems, but rather with 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 10.10).

Definition 10.10 (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* (see §1.2.3.6). *In addition, the algorithm is given the length of the input and possibly other parameters on a* secondary input device. *The complexity of such an algorithm is stated in terms of the length of its main input.*

Indeed, the description in §5.2.4.2 refers to such a model, but there the main input is viewed as an oracle and the secondary input is viewed as the input. In the current model, polylogarithmic time means time that is polylogarithmic in the length of the main input, which means time that is polynomial in the length of the binary representation of the length of the main input. Thus, polylogarithmic time yields a robust notion of extremely efficient computations. As we shall see, such computations suffice for solving various (property testing) problems.

Definition 10.11 (property testing for S): *For any fixed $\delta > 0$, 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. This is due to (1) the *effect of the representation on the distance measure* and to (2) the *dependence of direct access machines on the specific representation of the input.* Let us elaborate on both aspects.

1. Recall that we defined the distance between objects in terms of the Hamming distance between their representations. Clearly, in such a case, the choice of representation is crucial and different representations may yield different distance measures. Furthermore, in this case, the distance between objects is not preserved under various (natural) representations that are considered "equivalent" in standard studies of computational complexity. For example, in previous parts of this book, when referring to computational problems concerning graphs, we did not care whether the graph was represented by its adjacency matrix or by its incidence-list. In contrast, these two representations induce very different distance measures and correspondingly different property testing problems (see §10.1.2.2). Likewise, the use of padding (and other trivial syntactic conventions) becomes problematic (e.g., when using a significant amount of padding, all objects are deemed close to one another (and property testing for any set becomes trivial)).

2. Since our focus is on sub-linear time algorithms, we may not afford transforming the input from one natural format to another. Thus, representations that are considered equivalent with respect to polynomial-time algorithms, may not be equivalent with respect to sub-linear time algorithms that have a direct access to the representation of the object. For example, adjacency queries and incidence queries cannot emulate one another in small time (i.e., in time that is sub-linear in the number of vertices).

Both aspects are further clarified by the examples provided in §10.1.2.2.

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. This makes the property testing task potentially easier than the corresponding standard decision task (cf. §10.1.2.2). To demonstrate the point, consider the set S consisting of strings that have a majority of 1's. Then, deciding membership in S requires linear time, because random n-bit long strings with $\lfloor n/2 \rfloor$ ones cannot be distinguished from random n-bit long strings with $\lfloor n/2 \rfloor + 1$ ones by probing a sub-linear number of locations (even if randomization and error probability are allowed – see Exercise 10.8). On the other hand, the fraction of 1's in the input can be approximated by a randomized polylogarithmic time algorithm (which yields a property tester for S; see Exercise 10.9). Thus, for some sets, deciding membership requires linear time, while property testing can be done in polylogarithmic time.

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, $\Gamma_{0.1}(S)$. Specifically, a sub-linear time deterministic algorithm cannot distinguish 1^n from any input that has 1's in each position probed by that 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. This success is due to the fact that these lower-bounds are actually information theoretic in nature; that is, these lower-bounds actually refer to the number of queries performed by these algorithms.

10.1.2.2 Two models for testing graph properties

In this subsection 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 importance of representation in the context of property testing, we explicitly consider two standard representations of graphs (cf. Appendix G.1), which indeed yield two different models of testing graph properties.

1. The adjacency matrix representation. Here a graph $G = ([N], E)$ is represented (in a somewhat 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 somewhat 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 in the complexity of corresponding property testing problems.

Theorem 10.12 (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* (with respect to δ).

- *For every fixed $k \geq 2$, the set of k-colorable graphs.*
- *For every fixed $\rho > 0$, 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*[5] *with at least $\rho \cdot N^2$ edges.*
- *For every fixed $\rho > 0$, the set of N-vertex graphs having a bisection*[5] *with at most $\rho \cdot N^2$ edges.*

In contrast, for some $\delta > 0$, there exists a graph property in $\mathcal{NP}$ for which property testing (with respect to δ) *requires linear time.*

The testing algorithms (asserted in Theorem 10.12) use a constant number of queries, where this constant is polynomial in the constant $1/\delta$. In contrast, exact decision procedures for the corresponding sets require a linear number of queries. The running time of the aforementioned algorithms 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). Turning to the lower-bound (asserted in Theorem 10.12), we mention that the graph property for which this bound is proved is not a natural one. As in §10.1.2.1, the lower-bound on the time complexity follows from a lower-bound on the query complexity.

Theorem 10.12 exhibits a dichotomy between graph properties for which property testing is possible by a constant number of queries and graph properties for

[5] A cut in a graph $G = ([N], E)$ is a partition (S_1, S_2) of the set of vertices (i.e., $S_1 \cup S_2 = [N]$ and $S_1 \cap S_2 = \emptyset$), and the edges of the cut are the edges with exactly one endpoint in S_1. A bisection is a cut of the graph to two parts of equal cardinality.

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) when using a constant number of queries is known.[6] We note that the constant in this characterization may depend arbitrarily on δ (and indeed, in some cases, it is a function growing faster than a tower of $1/\delta$ exponents). For example, property testing for the set of *triangle-free* graphs is possible by using a number of queries that depends only on δ, but it is known that this number must grow faster than any polynomial in $1/\delta$.

Turning back to Theorem 10.12, 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 10.1). That is, for every constant $c > 1$, we obtain a *c-factor approximation* of the problem of maximizing the size of a cut (resp., minimizing the size of a 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 subgraph of a given size.

Indeed, Theorem 10.12 is meaningful only for dense graphs. This holds, in general, for any graph property in the adjacency matrix representation.[7] Also note that property testing is trivial, under the adjacency matrix representation, for any graph property S satisfying $\Gamma_{o(1)}(S) = \emptyset$ (e.g., the set of connected graphs, the set of Hamiltonian graphs, etc).

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 10.12) 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.

Theorem 10.13 (property testing in the bounded incidence-lists representation): *The following assertions refer to the representation of graphs by incidence-lists of length 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 time randomized algorithm that solves the property testing problem for the set of bipartite graphs of degree*

[6] Describing this fascinating result of Alon *et. al.* [9], which refers to the notion of regular partitions (introduced by Szemerédi), is beyond the scope of the current text.

[7] In this model, as shown next, property testing of non-dense graphs is trivial. Specifically, fixing the distance parameter δ, we call a N-vertex graph non-dense if it has less than $(\delta/2) \cdot \binom{N}{2}$ edges. The point is that, for non-dense graphs, the property testing problem for any set S is trivial, because we may just accept any non-dense (N-vertex) graph if and only if S contains some non-dense (N-vertex) graph. Clearly, the decision is correct in the case that S does not contain non-dense graphs. However, the decision is admissible also in the case that S does contain some non-dense graph, because in this case every non-dense graph is "δ-close" to S (i.e., it is not in $\Gamma_\delta(S)$).

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 of degree at most d requires $\Omega(N)$ queries.*

The running time of the algorithms (asserted in Theorem 10.13) 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 minimum of 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 $\max(|E_1|, |E_2|)$).

10.1.2.3 Beyond graph properties

Property testing has been applied to a variety of computational problems beyond the domain of graph theory. In fact, this type of computational problems 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 parameters that may depend on the length of the input (or satisfy some relationships; e.g., $q = d^3 = m^6$). Note that, in this case, the input is the ("full" or "explicit") description of an m-variate function over $\mathrm{GF}(q)$, which means that it 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).

Note that we have referred to these computational problems, under a different terminology, in §9.3.2.2 and in §9.3.2.1. In particular, in §9.3.2.1 we refereed to the special case of linear Boolean functions (i.e., individual degree 1 and $q = 2$), whereas in §9.3.2.2 we used the setting $q = \mathrm{poly}(d)$ and $m = d/\log d$ (where d is a bound on the total degree).

Other domains of computational problems in which property testing was studied include geometry (e.g., clustering problems), formal languages (e.g., testing membership in regular sets), coding theory (cf. Appendix E.1.3), probability theory (e.g., testing equality of distributions), and combinatorics (e.g., monotone and junta functions). As discuss at the end of §10.1.2.2, it is often natural to decouple the distance measure from the representation of the objects (i.e., the 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.

10.2 Average Case Complexity

Teaching note: We view average-case complexity as referring to the performance on "average" (or rather typical) instances, and not as the average performance on random instances. This choice is justified in §10.2.1.1. Thus, it may be more justified to refer to the following theory by the name typical-case complexity. Still, the name average-case was retained for historical reasons.

Our approach so far (including in Section 10.1) is termed worst-case complexity, because it refers to the performance of potential algorithms on each legitimate instance (and hence to the performance on the worst possible instance). That is, computational problems were defined as referring to a set of instances and performance guarantees were required to hold for each instance in this set. In contrast, average-case complexity allows ignoring a negligible measure of the possible instances, where *the identity of the ignored instances is determined by the analysis of potential solvers and not by the problem's statement.*

A few comments are in place. Firstly, as just hinted, the standard statement of the worst-case complexity of a computational problem (especially one having a promise) may also ignores some instances (i.e., those considered inadmissible or violating the promise), but these instances are determined by the problem's statement. In contrast, the inputs ignored in average-case complexity are not inadmissible in any inherent sense (and are certainly not identified as such by the problem's statement). It is just that they are viewed as exceptional when claiming that a specific algorithm solve the problem; that is, these exceptional instances are determined by the analysis of that algorithm. Needless to say, these exceptional instances ought to be rare (i.e., occur with negligible probability).

The last sentence raises a couple of issues. Most importantly, a distribution on the set of admissible instances has to be specified. In fact, we shall consider a new type of computational problems, each consisting of a standard computational problem coupled with a probability distribution on instances. Consequently, the question of which distributions should be considered in a theory of average-case complexity arises. This question and numerous other definitional issues will be addressed in §10.2.1.1.

Before proceeding, let us spell out the rather straightforward motivation to the study of the average-case complexity of computational problems: It is that, in real-life applications, one may be perfectly happy with an algorithm that solves the problem fast on almost all instances that arise in the relevant application. That is,