# 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 natural computational problems, it is natural to ask whether these problems can be relaxed in a way that is both useful for applications 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 version 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; put differently, we 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 specific input distribution, and bad behavior may occur with negligible probability

where the probability is taken over the input distribution. 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 average-case complexity to worst-case complexity.

## 10.1 Approximation

*On the difficulty of being perfect and the ease of being almost perfect*

Dana Ron, title of talk at Radcliffe (2003)

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 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 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.

### 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 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 $\ell$ (see Exercise 2.8). 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 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 $\ell$) 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 $\ell$) 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 $c_x=\min_{y'\in\{0,1\}^{\ell(|x|)}}\{f(x,y')\}$.*

[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 $R$ (as in Exercise 2.8).

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 $G = (V, E)$ 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. □

**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 10.1), but here we consider two special cases in which the distances satisfies some natural constraints.

**Theorem 10.3** (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 10.2. A detailed survey of Part 2 is provided in [12].

#### 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 $G$ is as hard as finding a maximum clique in $G$. This follows from the fact that the approximation problem is NP-hard (cf. Theorem 10.5).

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 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* $\text{gap}_{g_1,g_2}$ 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* $\text{gap}_{g_1,g_2}$ 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 $\text{gap}_{g_1,g_2}$ 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 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.*

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

As stated in Section 9.3.3, the PCP Theorem and versions of it play a key role in establishing inapproximability results such as Theorems 10.5 and 10.7. In particular, 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 $\varepsilon$ fraction of the clauses. Although Theorem 9.21 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 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 threshold.** The aforementioned results regarding $\texttt{gapSAT}^3_v$ 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 $\texttt{gapLin}_{L,s}$ 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, $\texttt{gapLin}_{L,0.5}$ is trivial and $\texttt{gapLin}_{1,s}$ is feasible (for every $s < 1$). In contrast, moving both thresholds towards one another yields an NP-hard gap problem:

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

**Gap location.** Theorems 10.8 and 10.9 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., $\texttt{gapLin}_{1,s}$ is in P), whereas in the first case the corresponding gap problem may be hard. Indeed, $\texttt{gapSAT}_\varepsilon$ 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 $\texttt{gapSAT}_\varepsilon$ 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 $\mathcal{NP}$; 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, since 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., 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 $\delta$, we consider the problem of distinguishing inputs in $S$ from inputs in $\Gamma_\delta(S)$, where
$$\Gamma_\delta(S) \stackrel{\text{def}}{=} \{x : \forall z \in S \cap \{0,1\}^{|x|} \;\; \Delta(x,z) > \delta \cdot |x|\} \qquad (10.1)$$
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 instances are pairs $(x, \delta)$, where $x$ is as in Type 1 and $\delta$ is a distance parameter. The yes-instances are pairs $(x, \delta)$ such that $x \in S$, whereas $(x, \delta)$ 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 §10.1.2.2. As indicated in §10.1.2.2, this is not a generic phenomenon. We start by discussing several key definitional issues (see §10.1.2.1).

#### 10.1.2.1 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 10.10).

**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 §1.2.3.5). *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.*

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.

**Definition 10.11** (property testing for $S$): *The promise problem of distinguishing $S$ from $\Gamma_\delta(S)$ is called* property testing for $S$ (with respect to $\delta$).

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 §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. 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 10.10). 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.

#### 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 foregoing comment, referring to the importance of representation, we consider two standard representations of graphs (see Appendix G.1).

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 $i^{\text{th}}$ 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 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.*

- *The set of $k$-colorable graphs, for every fixed $k \geq 2$.*
- *The set of graphs having a clique* (resp. independent set) *of density $\rho$, 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 10.12 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

[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.* [8], which refers to the notion of regular partitions (introduced by Szemerédi), is beyond the scope of the current text.

constant in this characterization may depend arbitrarily on $\delta$ (and indeed, in some cases, it a function growing faster than a tower of $1/\delta$ exponents).

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 $\delta N^2$). 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 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 10.12 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 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 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.

[4] In this model, all $N$-vertex graphs having less than $\delta N^2$ 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 $\delta N^2$ then $\Gamma_\delta(S) = \emptyset$.

**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 $\max(|E_1|, |E_2|)$).

#### 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 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 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 GF$(q)$, has length $q^m \cdot \log_2 q$).

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 $q = \text{poly}(d)$ and $m = d/\log d$ (where $d$ is a bound on the total degree).

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 §10.1.2.2, 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.

## 10.2 Average Case Complexity

**Teaching note:** We view average-case complexity as referring to the performance on average (or typical) instances, and not as the average performance on random instances. This choice is justified in §10.2.1.1. Thus, the current theory may be termed typical-case complexity.

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; furthermore, 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. Firstly, 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 arises. These 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 application. That is, one may be willing to tolerate error provided that it occurs with negligible probability, where the probability is taken over the distribution of instances encountered in the application. We stress that a key aspect in this approach is a good modeling of the type of distributions of instances that are encountered in natural algorithmic applications.

At this point a natural question arises: *can natural computational problems be solve efficiently when considering typical instances?* The bottom-line of this section is that, for a well-motivated choice of definitions, our conjecture is that the "distributional version" of NP is not contained in the average-case (or typical-case) version of P. This means that some NP problems are not merely hard in the worst-case, but rather "typically hard" (i.e., hard on typical instances drawn from some simple distribution). Specifically, hard instances may occur in natural algorithmic applications (and not only in cryptographic (and other "adversarial") applications that are design on purpose to produce hard instances). This conjecture motivates the development of an average-case analogue of NP-completeness, which will be presented in this section. Indeed, the entire section may be viewed as an average-case analogue of Chapter 2.

**Organization.** A major part of our exposition is devoted to the definitional issues that arise when developing a general theory of average-case complexity. These issues are discussed in §10.2.1.1. In §10.2.1.2 we prove the existence of a distributional problem that is "NP-complete" in the average-case complexity sense. In

§10.2.1.3 we extend the treatment to randomized algorithms. Additional ramifications are presented in Section 10.2.2.

## 10.2.1 The basic theory

In this section we provide a basic treatment of the theory of average-case complexity, while postponing important ramifications to Section 10.2.2. The basic treatment contains the preferred definitional choices for the main notions as well as the identification of a complete problem for a natural class of average-case computational problems.

### 10.2.1.1 Definitional issues

The theory of average-case complexity is more subtle than may appear in first thought. In addition to the generic difficulty involved in defining relaxations, difficulties arise from the "interface" between standard probabilistic analysis and the conventions of complexity theory. This is most striking in the definition of the class of feasible average-case computations. Referring to the theory of worst-case complexity as a guideline, we shall address the following aspects of the analogous theory of average-case complexity.

1. *Setting the general framework.* We shall consider distributional problems, which are standard computational problems (see Section 1.2.2) coupled with distributions on the relevant instances.

2. *Identifying the class of feasible* (distributional) *problems.* Seeking an average-case analogue of classes such as $\mathcal{P}$, we shall reject the first definition (of "average polynomial time") that comes to mind, briefly discuss several related alternatives, and adopt one of them for the main treatment.

3. *Identifying the class of interesting* (distributional) *problems.* Seeking an average-case analogue of the class $\mathcal{NP}$, we shall avoid both the extreme of allowing arbitrary distributions (which collapses average-case complexity to worst-case complexity) and the opposite extreme of confining the treatment to the uniform distribution (which is misguided by the naive assumption that this distribution is the only one relevant to applications).

4. *Developing an adequate notion of reduction among* (distributional) *problems.* As in the theory of worst-case complexity, this notion should preserve feasible solveability (in the current distributional context).

We now turn to the actual treatment of each of the aforementioned aspects.

**Distributional problems.** Focusing on decision problems, we define distributional problems as pairs consisting of a decision problem and a probability ensemble.[5]

[5]We mention that even this choice is not evident. Specifically, Levin [144] (see discussion in [85]) advocates the use of a single probability distribution defined over the set of all strings.

For simplicity, here a probability ensemble $\{X_n\}_{n\in\mathbb{N}}$ is a sequence of random variables such that $X_n$ ranges over $\{0,1\}^n$. Thus, $(S,\{X_n\}_{n\in\mathbb{N}})$ is the distributional problem consisting of the problem of deciding membership in the set $S$ with respect to the probability ensemble $\{X_n\}_{n\in\mathbb{N}}$. (The treatment of search problem is similar; see §10.2.2.1.) We denote the uniform probability ensemble by $U = \{U_n\}_{n\in\mathbb{N}}$; that is, $U_n$ is uniform over $\{0,1\}^n$.

**Identifying the class of feasible problems.** The first idea that comes to mind is defining the problem $(S,\{X_n\}_{n\in\mathbb{N}})$ as feasible (on the average) if there exists an algorithm $A$ that solves $S$ such that the *average running time* of $A$ on $X_n$ is bounded by a polynomial in $n$ (i.e., there exists a polynomial $p$ such that $\mathsf{E}[t_A(X_n)] \leq p(n)$, where $t_A(x)$ denotes the running-time of $A$ on input $x$). The problem with this definition is that it very sensitive to the model of computation and is not closed under algorithmic composition. Both deficiencies are a consequence of the fact that $t_A$ may be polynomial on the average with respect to $\{X_n\}_{n\in\mathbb{N}}$ but $t_A^2$ may fail to be so (e.g., consider $t_A(x'x'') = 2^{|x'|}$ if $x' = x''$ and $t_A(x'x'') = |x'x''|^2$ otherwise, coupled with the uniform distribution over $\{0,1\}^n$). We conclude that the average running-time of algorithms is not a robust notion. We also doubt the naive appeal of this notion, and view the typical running time of algorithms (as defined next) as a more natural notion. Thus, we shall consider an algorithm as feasible if its running-time is typically polynomial.[6]

We say that $A$ is typically polynomial-time on $X = \{X_n\}_{n\in\mathbb{N}}$ if there exists a polynomial $p$ such that the probability that $A$ runs more that $p(n)$ steps on $X_n$ is *negligible* (i.e., for every polynomial $q$ and all sufficiently large $n$ it holds that $\mathsf{Pr}[t_A(X_n) > p(n)] < 1/q(n)$). The question is what is required in the "untypical" cases, and two possible definitions follow.

1. The simpler option is saying that $(S,\{X_n\}_{n\in\mathbb{N}})$ is (typically) feasible if there exists an algorithm $A$ that solves $S$ such that $A$ is typically polynomial-time on $X = \{X_n\}_{n\in\mathbb{N}}$. This effectively requires $A$ to correctly solve $S$ on each instance, which is more than was required in the motivational discussion. (Indeed, if the underlying reasoning is ignoring rare cases, then we should ignore them altogether rather than partially (i.e., only ignore their affect on the running-time).)

2. The alternative, which fits the motivational discussion, is saying that $(S,X)$ is (typically) feasible if there exists an algorithm $A$ such that typically $A$

His argument is that this makes the theory less representation-dependent. At the time we were convinced of his argument (see [85]), but currently we feel that the representation-dependent effects discussed in [85] are legitimate. Furthermore, the alternative formulation of [85] comes across as unnatural and tends to be confusing.

[6]An alternative choice, taken by Levin [144] (see discussion in [85]), is considering as feasible (w.r.t $X = \{X_n\}_{n\in\mathbb{N}}$) any algorithm that runs in time that is polynomial in a function that is linear on the average (w.r.t $X$); that is, requiring that there exists a polynomial $p$ and a function $\ell : \{0,1\}^* \to \mathbb{N}$ such that $t(x) \leq p(\ell(x))$ and $\mathsf{E}[\ell(X_n)] = O(n)$. This definition is robust (i.e., it does not suffer from the aforementioned deficiencies) and is arguably as justified as the naive definition (i.e., $\mathsf{E}[t_A(X_n)] \leq \mathrm{poly}(n)$).

solves $S$ on $X$ in polynomial-time; that is, there exists a polynomial $p$ such that the probability that on input $X_n$ algorithm $A$ either errs or runs more that $p(n)$ steps is negligible. This formulation totally ignores the untypical instances. Indeed, in this case we may assume, without loss of generality, that $A$ always runs in polynomial-time (see Exercise 10.4), but we shall not do so here (in order to facilitate viewing the first option as a special case of the current option).

We note that both alternatives actually define typical feasibility and not average-case feasibility. To illustrate the difference between the two options, consider the distributional problem of deciding whether a uniformly selected ($n$-vertex) graph contains a Hamiltonian path. Intuitively, this problem is "typically trivial" because the algorithm may always say `yes` and be wrong with exponentially vanishing probability.[7] Indeed, this trivial algorithm is admissible by the second approach, but not by the first approach. In light of the foregoing, we adopt the second approach.

**Definition 10.14** (the class tpc$\mathcal{P}$): *We say that $A$* typically solves $(S, \{X_n\}_{n\in\mathbb{N}})$ in polynomial-time *if there exists a polynomial $p$ such that the probability that on input $X_n$ algorithm $A$ either errs or runs more that $p(n)$ steps is negligible.*[8] *We denote by* tpc$\mathcal{P}$ *the class of distributional problems that are typically solvable in polynomial-time.*

Clearly, for every $S \in \mathcal{P}$ and every probability ensemble $X$, it holds that $(S, X) \in$ tpc$\mathcal{P}$. However, tpc$\mathcal{P}$ contains also distributional problems $(S, X)$ with $S \notin \mathcal{P}$ (see Exercises 10.5 and 10.6). The big question motivating the study of average-case complexity is whether natural distributional versions of $\mathcal{NP}$ are in tpc$\mathcal{P}$. Thus, we turn to identify such versions.

**Identifying the class of interesting problems.** Seeking to identify reasonable distributional versions of $\mathcal{NP}$, we note that two extreme choices should be avoided. On one hand, we must limit the class of admissible distributions so to prevent the collapse of average-case complexity to worst-case complexity (by a selection of a pathological distribution that resides on the "worst case" instances). On the other hand, we should allow for various types of natural distributions rather than confining attention merely to the uniform distribution. Recall that our aim is to address all possible input distributions that may occur in applications, and there is no justification to confining attention to the uniform distribution. Still, arguably, the distributions occuring in applications are "relatively simple" and so we seek to identify a class of simple distributions. One such notion (of simple distributions)

[7] In contrast, testing whether a given graph contains a Hamiltonian path seems "typically hard" for other distributions (see Exercise 10.15). Needless to say, in the latter distributions both yes-instances and no-instances appear with noticeable probability.

[8] Recall that a function $\mu : \mathbb{N} \to \mathbb{N}$ is negligible if for every positive polynomial $q$ and all sufficiently large $n$ it holds that $\mu(n) < 1/q(n)$. We say that $A$ errs on $x$ if $A(x)$ differs from the indicator value of the predicate $x \in S$.

underlies the following definition, while a more liberal notion will be presented in §10.2.2.2.

**Definition 10.15** (the class dist$\mathcal{NP}$): *We say that a probability ensemble $X = \{X_n\}_{n\in\mathbb{N}}$ is* simple *if there exists a polynomial time algorithm that, on any input $x \in \{0,1\}^*$, outputs $\Pr[X_{|x|} \le x]$, where the inequality refers to the standard lexicographic order of strings. We denote by* dist$\mathcal{NP}$ *the class of distributional problems consisting of decision problems in $\mathcal{NP}$ coupled with simple probability ensembles.*

Note that the uniform probability ensemble is simple, but so are many other "simple" probability ensembles. Actually, it makes sense to relax the definition such that the algorithm is only required to output an approximation of $\Pr[X_{|x|} \le x]$, say, to within a factor of $1 \pm 2^{-|x|}$. We note that although Definition 10.15 does not refer explicitly to any structural property of the probability ensemble, it imposes a computational restriction on the admissible ensembles (which, in turn, limits their "complexity"). In §10.2.2.2 we shall consider the more intuitive and robust class of all polynomial-time sampleable ensembles (and show that it contains all simple ensembles). We believe that the results presented in §10.2.1.2 and §10.2.2.2 retrospectively justify the choice underlying Definition 10.15. We articulate this point next.

We note that a wider class of distributions weakens the conjecture that some distributional version of NP are not feasible. On the other hand, the conclusion that some distributional problem is not feasible becomes stronger when restricting the admissible class of distributions. The results presented in §10.2.1.2 and §10.2.2.2 assert that a conjecture that refers to polynomial-time sampleable ensembles implies a conclusion that refers to a (very) simple probability ensemble. The current setting in which both the conjecture and the conclusion refer to simple probability ensembles is thus but an intermediate step.

Indeed, the big question in the current context is whether dist$\mathcal{NP}$ is contained in tpc$\mathcal{P}$. A positive answer (especially if extended to sampleable ensembles) would deem the P-vs-NP Question of little practical significant. However, our daily experience as well as much research effort indicate that some NP problems are not merely hard in the worst-case, but rather "typically hard". This supports the *conjecture that* dist$\mathcal{NP}$ *is not contained in* tpc$\mathcal{P}$.

Needless to say, the latter conjecture implies $\mathcal{P} \neq \mathcal{NP}$, and thus we should not expect to see a proof of it. What we may hope to see is "dist$\mathcal{NP}$-complete" problems; that is, problems in dist$\mathcal{NP}$ that are not in tpc$\mathcal{P}$ unless the entire class dist$\mathcal{NP}$ is contained in tpc$\mathcal{P}$. An adequate notion of a reduction is used towards formulating this notion.

**Reductions among (distributional) problems.** Intuitively, such reductions must preserve average-case feasibility. Thus, in addition to the standard conditions (i.e., that the reduction be efficiently computable and yield a correct result), we require that the reduction "respects" the probability distribution of the corresponding distributional problems. Specifically, the reduction should not map very likely instances of the first ("starting") problem to rare instances of the second ("target") problem. Otherwise, having a typically polynomial-time algorithm for the second distributional problem does not necessarily yield such an algorithm for the first distributional problem. Following is the adequate analogue of a Cook reduction (i.e., general polynomial-time reduction), where the analogue of a Karp-reduction (many-to-one reduction) can be easily derived as a special case.

> **Teaching note:** One may prefer presenting in class only the special case of many-to-one reductions, which suffices for Theorem 10.17. See Footnote 10.

**Definition 10.16** (reductions among distributional problems): *We say that the oracle machine $M$* reduces *the distributional problem $(S,X)$ to the distributional problem $(T,Y)$ if the following three conditions hold.*

1. Efficiency: *The machine $M$ runs in polynomial-time.*[9]
2. Validity: *For every $x \in \{0,1\}^*$, it holds that $M^T(x) = 1$ if an only if $x \in S$, where $M^T(x)$ denotes the output of the oracle machine $M$ on input $x$ and access to an oracle for $T$.*
3. Domination:[10] *There exists a polynomial $p$ such that, for every $y \in \{0,1\}^*$, it holds that*

$$\max_{n\in\mathbb{N}}\{\mathsf{Pr}[Q(X_n) \ni y]\} \le p(|y|)\cdot \mathsf{Pr}[Y_{|y|} = y], \tag{10.2}$$

   *where $Q(x)$ denotes the set of queries made by $M$ on input $x$ and oracle access to $T$. Furthermore, if $y \in Q(x)$ then $|x| \le p(|y|)$.*

The l.h.s. of Eq. (10.2) refers to the probability that, on input distributed as $X_n$, the reduction makes the query $y$. This probability is required not to exceed the probability that $y$ occurs in the distribution $Y_{|y|}$ by more than a polynomial factor in $|y|$. In this case we say that the l.h.s. of Eq. (10.2) is dominated by $\mathsf{Pr}[Y_{|y|} = y]$.

Indeed, the domination condition is the only aspect of Definition 10.16 that extends beyond the worst-case treatment of reductions and refers to the distributional setting. The domination condition does not insist that the distribution induced by $Q(X)$ equals $Y$, but rather allows some slackness that, in turn, is bounded so to guarantee preservation of typical feasibility (see Exercise 10.7).

We note that the reducibility arguments extensively used in Chapters 7 and 8 (see discussion in Section 7.1.2) are actually reductions in the spirit of Definition 10.16 (except that they refer to a different type of computational tasks).

[9] In fact, one may relax the requirement and only require that $M$ is typically polynomial-time with respect to $X$. The validity condition may also be relaxed similarly.

[10] For simplicity, we use the same polynomial for both bounds. Let us spell out the meaning of Eq. (10.2) in the special case of many-to-one reductions (i.e., $M^T(x) = 1$ if and only if $f(x) \in T$, where $f$ is a polynomial-time computable function): in this case $\mathsf{Pr}[Q(X_n) \ni y]$ is replaced by $\mathsf{Pr}[f(X_n) = y]$. Assuming that $f$ is one-to-one, Eq. (10.2) simplifies to $\mathsf{Pr}[X_{|f^{-1}(y)|} = f^{-1}(y)] \le p(|y|)\cdot\mathsf{Pr}[Y_{|y|} = y]$ for any $y$ in the image of $f$. Indeed, nothing is required for $y$ not in the image of $f$.

#### 10.2.1.2 Complete problems

Recall that our conjecture is that dist$\mathcal{NP}$ is not contained in tpc$\mathcal{P}$, which in turn strengthens the conjecture $\mathcal{P} \neq \mathcal{NP}$ (making infeasibility a typical phenomenon rather than a worst-case one). Having no hope of proving that dist$\mathcal{NP}$ is not contained in tpc$\mathcal{P}$, we turn to the study of complete problems with respect to that conjecture. Specifically, we say that a distributional problem $(S,X)$ is dist$\mathcal{NP}$-complete if $(S,X) \in$ dist$\mathcal{NP}$ and every $(S',X') \in$ dist$\mathcal{NP}$ is reducible to $(S,X)$ (under Definition 10.16).

Recall that it is quite easy to prove the mere existence of NP-complete problems and many natural problems are NP-complete. In contrast, in the current context, establishing completeness results is quite hard. This should not be surprising in light of the restricted type of reductions allowed in the current context. The restriction (captured by the domination condition) requires that "typical" instances of one problem should not be mapped to "untypical" instances of the other problem. On the other hand, it is fair to say that standard Karp-reductions (used in establishing NP-completeness results) map "typical" instances of one problem to quite "bizarre" instances of the second problem. Thus, the current subsection may be viewed as a study of reductions that do not commit this sin.

**Theorem 10.17** (dist$\mathcal{NP}$-completeness): *dist$\mathcal{NP}$ contains a distributional problem $(S,X)$ such that each distributional problem in* dist$\mathcal{NP}$ *is reducible* (per Definition 10.16) *to* $(S,X)$. *Furthermore, the reduction is deterministic and many-to-one.*

**Proof:** We start by introducing such a problem, which is a natural distributional version of the decision problem $S_{\mathtt{u}}$ (used in the proof of Theorem 2.18). Recall that $S_{\mathtt{u}}$ contains the instance $\langle M,x,1^t\rangle$ if there exists $y \in \cup_{i\le t}\{0,1\}^i$ such that $M$ accepts the input pair $(x,y)$ within $t$ steps. We couple $S_{\mathtt{u}}$ with the "quasi-uniform" probability ensemble $U'$ that assigns to the instance $\langle M,x,1^t\rangle$ a probability mass proportional to $2^{-(|M|+|x|)}$. Specifically, for $|\langle M,x,1^t\rangle| = n$ it holds that $\mathsf{Pr}[U'_n = \langle M,x,1^t\rangle] = 2^{-(|M|+|x|)}/\binom{n}{2}$. Note that, under a suitable encoding, the ensemble $U'$ is indeed simple.[11]

The reader can easily verify that the generic reduction used when reducing any set in $\mathcal{NP}$ to $S_{\mathtt{u}}$ (see the proof of Theorem 2.18), fails to reduce dist$\mathcal{NP}$ to $(S_{\mathtt{u}},U')$. Specifically, in some cases (see next paragraph), these reductions do not satisfy the domination condition. Indeed, the difficulty is that we have to reduce all dist$\mathcal{NP}$ problems (i.e., pairs consisting of decision problems and simple distributions) to one single distributional problem (i.e., $(S_{\mathtt{u}},U')$). Applying the aforementioned reductions, we end up with many distributional versions of $S_{\mathtt{u}}$, and furthermore the corresponding distributions are very different (and are not necessarily dominated by a single distribution).

Let us take a closer look at the aforementioned generic reduction, when applied to an arbitrary $(S,X) \in$ dist$\mathcal{NP}$. This reduction maps an instance $x$ to a triple $(M_S,x,1^{p_S(|x|)})$, where $M_S$ is a machine verifying membership in $S$ (while using

[11] For example, we may encode $\langle M,x,1^t\rangle$, where $M = \sigma_1\cdots\sigma_k \in \{0,1\}^k$ and $x = \tau_1\cdots\tau_\ell \in \{0,1\}^\ell$, by the string $\sigma_1\sigma_1\cdots\sigma_k\sigma_k01\tau_1\tau_1\cdots\tau_\ell\tau_\ell01^t$.

adequate NP-witnesses) and $p_S$ is an adequate polynomial. The problem is that $x$ may have relatively large probability mass (i.e., it may be that $\Pr[X_{|x|}=x] \gg 2^{-|x|}$) while $(M_S, x, 1^{p_S(|x|)})$ has "uniform" probability mass (i.e., $\langle M_S, x, 1^{p_S(|x|)}\rangle$ has probability mass smaller than $2^{-|x|}$). This violates the domination condition (see Exercise 10.10), and thus an alternative reduction is required.

The key to the alternative reduction is an (efficiently computable) encoding of strings taken from an arbitrary *simple* distribution by strings that have a similar probability mass under the uniform distribution. This means that the encoding should shrink strings that have relatively large probability mass under the original distribution. Specifically, this encoding will map $x$ (taken from the ensemble $\{X_n\}_{n\in\mathbb{N}}$) to a codeword $x'$ of length that is upper-bounded by the logarithm of $1/\Pr[X_{|x|}=x]$, ensuring that $2^{-|x'|} \geq \Pr[X_{|x|}=x]$. Accordingly, the reduction will map $x$ to a triple $(M_{S,X}, x', 1^{p'(|x|)})$, where $|x'| < O(1) + \log_2(1/\Pr[X_{|x|}=x])$ and $M_{S,X}$ is an algorithm that first verifies that $x'$ is a proper encoding of $x$ and next applies the standard verification (i.e., $M_S$) of the problem $S$. Such a reduction will be shown to satisfy all three conditions (i.e., efficiency, validity, and domination). Thus, instead of forcing the structure of the original distribution $X$ on the target distribution $U'$, the reduction will incorporate the structure of $X$ in the reduced instance. A key ingredient in making this possible is the fact that $X$ is simple (as per Definition 10.15).

With the foregoing motivation in mind, we now turn to the actual proof; that is, proving that any $(S, X)$ is reducible to $(S_{\mathtt{u}}, U')$. The following technical lemma is the basis of the reduction. In this lemma as well as in the sequel, it will be convenient to consider the (accumulative) distribution function of the probability ensemble $X$. That is, we consider $\mu(x) \stackrel{\text{def}}{=} \Pr[X_{|x|} \leq x]$, and note that $\mu : \{0,1\}^* \to [0,1]$ is polynomial-time computable (because $X$ satisfies Definition 10.15).

Coding Lemma:[12] Let $\mu$ be a polynomial-time computable distribution function. Then there exist an encoding function $C_\mu$ satisfying the following three conditions.

1. Compression: For every $x$ it holds that $|C_\mu(x)| \leq 1 + \min\{|x|, \log_2(1/\mu'(x))\}$, where $\mu'(x) \stackrel{\text{def}}{=} \Pr[X_{|x|}=x]$.
2. Efficient Encoding: The function $C_\mu$ is computable in polynomial-time.
3. Unique Decoding: For every $n \in \mathbb{N}$, when restricted to $\{0,1\}^n$, the function $C_\mu$ is one-to-one (i.e., if $C_\mu(x) = C_\mu(x')$ and $|x| = |x'|$ then $x = x'$).

Proof: The function $C_\mu$ is defined as follows. If $\mu'(x) \leq 2^{-|x|}$ then $C_\mu(x) = 0x$ (i.e., in this case $x$ serves as its own encoding). Otherwise (i.e., $\mu'(x) > 2^{-|x|}$) then $C_\mu(x) = 1z$, where $z$ is chosen such that $|z| \leq \log_2(1/\mu'(x))$ and the mapping of $n$-bit strings to their encoding is one-to-one. Loosely speaking, $z$ is selected to

[12] The lemma actually refers to $\{0,1\}^n$, for a fixed value of $n$, but the efficiency condition is stated more easily when allowing $n$ to vary (and using the standard asymptotic analysis of algorithms). Furthermore, the lemma holds for any monotonically non-decreasing function that is efficiently computable, and its proof is less cumbersome when stated for functions defined over $\{0,1\}^*$. See further discussion in Exercise 10.11.

equal the shortest binary expansion of a number in the interval $(\mu(x)-\mu'(x),\mu(x)]$. Bearing in mind that this interval has length $\mu'(x)$ and that the different intervals are disjoint, we obtain the desired encoding. Details follows.

We focus on the case that $\mu'(x) > 2^{-|x|}$, and detail the way that $z$ is selected (for the encoding $C_\mu(x) = 1z$). If $x > 0^{|x|}$ and $\mu(x) < 1$, then we let $z$ be the longest common prefix of the binary expansions of $\mu(x-1)$ and $\mu(x)$, where $x-1$ is the string preceding $x$ in lexicographic order (e.g., if $\mu(1010) = 0.10010$ and $\mu(1011) = 0.10101111$ then $C_\mu(1011) = 1z$ with $z = 10$). Thus, in this case $0.z1$ is in the interval $(\mu(x-1),\mu(x)]$ (i.e., $\mu(x-1) < 0.z1 \leq \mu(x)$). For $x = 0^{|x|}$, we let $z$ be the longest common prefix of the binary expansions of 0 and $\mu(x)$ and again $0.z1$ is in the relevant interval (i.e., $(0,\mu(x)]$). Finally, for $x$ such that $\mu(x) = 1$ and $\mu(x-1) < 1$, we let $z$ be the longest common prefix of the binary expansions of $\mu(x-1)$ and $1-2^{-|x|-1} < \mu(x)$ (and again $0.z1$ is in $(\mu(x-1),\mu(x)]$). Note that if $\mu(x) = \mu(x-1) = 1$ then $\mu'(x) = 0 < 2^{-|x|}$.

We now verify that the foregoing $C_\mu$ satisfies the conditions of the lemma. We start with the compression condition. Clearly, if $\mu'(x) \leq 2^{-|x|}$ then $|C_\mu(x)| = 1+|x| \leq 1+\log_2(1/\mu'(x))$. On the other hand, suppose that $\mu'(x) > 2^{-|x|}$ and let us focus on the sub-case that $x > 0^{|x|}$ and $\mu(x) < 1$. Let $z = z_1\cdots z_\ell$ be the longest common prefix of the binary expansions of $\mu(x-1)$ and $\mu(x)$. Then, $\mu(x-1) = 0.z0u$ and $\mu(x) = 0.z1v$, where $u,v \in \{0,1\}^*$, and it follows that

$$\mu'(x) \;=\; \mu(x)-\mu(x-1) \;\leq\; \left(\sum_{i=1}^{\ell} 2^{-i}z_i + \sum_{i=\ell+1}^{\mathrm{poly}(|x|)} 2^{-i}\right) - \sum_{i=1}^{\ell} 2^{-i}z_i \;<\; 2^{-|z|}.$$

Thus, $|z| < \log_2(1/\mu'(x)) \leq |x|$ and it follows that $|C_\mu(x)| \leq 1+\min(|x|,\log_2(1/\mu'(x)))$ holds in both cases. Clearly, $C_\mu$ can be computed in polynomial-time by computing $\mu(x-1)$ and $\mu(x)$. Finally, note that $C_\mu$ satisfies the unique decoding condition, by separately considering the two aforementioned cases (i.e., $C_\mu(x) = 0x$ and $C_\mu(x) = 1z$). Specifically, in the second case (i.e., $C_\mu(x) = 1z$), use the fact that $\mu(x-1) < 0.z1 \leq \mu(x)$. □

To obtain an encoding that is one-to-one when applied to strings of different lengths we augment $C_\mu$ in the obvious manner; that is, we consider $C'_\mu(x) \stackrel{\mathrm{def}}{=} (|x|, C_\mu(x))$, which may be implemented as $C'_\mu(x) = \sigma_1\sigma_1\cdots\sigma_\ell\sigma_\ell 01 C_\mu(x)$ where $\sigma_1\cdots\sigma_\ell$ is the binary expansion of $|x|$. Note that $|C'_\mu(x)| = O(\log|x|) + |C_\mu(x)|$ and that $C'_\mu$ is one-to-one.

The machine associated with $(S,X)$. Let $\mu$ be the accumulative probability function associated with the probability ensemble $X$ and $M_S$ be the polynomial-time machine that verifies membership in $S$ while using adequate NP-witnesses (i.e., $x \in S$ if and only if there exists $y \in \{0,1\}^{\mathrm{poly}(|x|)}$ such that $M(x,y) = 1$). Using the encoding function $C'_\mu$, we introduce an algorithm $M_{S,\mu}$ with the intension of reducing the distributional problem $(S,X)$ to $(S_{\mathtt{u}},U')$ such that all instances (of $S$) are mapped to triples in which the first element equals $M_{S,\mu}$. Machine $M_{S,\mu}$ is given an alleged encoding (under $C'_\mu$) of an instance to $S$ along with an alleged proof that the corresponding instance is in $S$, and verifies these claims in the obvious manner. That is, on input $x'$ and $\langle x,y\rangle$, machine $M_{S,\mu}$ first verifies that $x'=C'_\mu(x)$, and next verifiers that $x\in S$ by running $M_S(x,y)$. Thus, $M_{S,\mu}$ verifies membership in the set $S'=\{C'_\mu(x):x\in S\}$, while using proofs of the form $\langle x,y\rangle$ such that $M_S(x,y)=1$ (for the instance $C'_\mu(x)$).[13]

The reduction. We maps an instance $x$ (of $S$) to the triple $(M_{S,\mu},C'_\mu(x),1^{p(|x|)})$, where $p(n)\stackrel{\rm def}{=}p_S(n)+p_C(n)$ such that $p_S$ is a polynomial representing the running-time of $M_S$ and $p_C$ is a polynomial representing the running-time of the encoding algorithm. That is, on input $(x,y)$, algorithm $M_S$ makes at most $p_S(|x|)$ steps (and rejects $(x,y)$ if $|y|>p(|x|)-|x|$).

Analyzing the reduction. Our goal is proving that *the foregoing mapping constitutes a reduction of* $(S,X)$ *to* $(S_{\tt u},U')$. We verify the corresponding three requirements (of Definition 10.16).

1. Using the fact that $C_\mu$ is polynomial-time computable (and noting that $p$ is a polynomial), it follows that the foregoing mapping can be computed in polynomial-time.

2. Recall that, on input $(x',\langle x,y\rangle)$, machine $M_{D,\mu}$ accepts if and only if $x'=C'_\mu(x)$ and $M_D$ accepts $(x,y)$ within $p_S(|x|)$ steps. It follows that $x\in S$ if and only if there exists a string $y$ of length at most $p(|x|)$ such that $M_{S,\mu}$ accepts $(C'_\mu(x),\langle x,y\rangle)$ in at most $p(|x|)$ steps. Thus, $x\in S$ if and only if $(M_{S,\mu},C'_\mu(x),1^{p(|x|)})\in S_{\tt u}$, and the validity condition follows.

3. In order to verify the domination condition, we first note that the foregoing mapping is one-to-one (because the transformation $x\to C'_\mu(x)$ is one-to-one). Next, we note that it suffices to consider instances of $S_{\tt u}$ that have a preimage under the foregoing mapping (since instances with no preimage trivially satisfy the domination condition). Each of these instances (i.e., each image of this mapping) is a triple with the first element equal to $M_{S,\mu}$ and the second element being an encoding under $C'_\mu$. By the definition of $U'$, for every such image $\langle M_{S,\mu},C'_\mu(x),1^{p(|x|)}\rangle\in\{0,1\}^n$, it holds that

$$\begin{aligned}\Pr[U'_n=\langle M_{S,\mu},C'_\mu(x),1^{p(|x|)}\rangle] &= \binom{n}{2}^{-1}\cdot 2^{-(|M_{S,\mu}|+|C'_\mu(x)|)}\\ &> c\cdot n^{-2}\cdot 2^{-(|C_\mu(x)|+O(\log|x|))},\end{aligned}$$

where $c=2^{-|M_{S,\mu}|-1}$ is a constant depending only on $S$ and $\mu$ (i.e., on the distributional problem $(S,X)$). Thus, for some positive polynomial $p'$, we have

$$\Pr[U'_n=\langle M_{S,\mu},C'_\mu(x),1^{p(|x|)}\rangle]>p'(n)^{-1}\cdot 2^{-|C_\mu(x)|}. \tag{10.3}$$

[13] Note that $|y|=\mathrm{poly}(|x|)$, but $|x|=\mathrm{poly}(|C'_\mu(x)|)$ does not necessarily hold (and so $S'$ is not necessarily in $\mathcal{NP}$). As we shall see, the latter point is immaterial.

By virtue of the compression condition (of the Coding Lemma), we have $2^{-|C_\mu(x)|} \geq 2^{-1-\min(|x|,\log_2(1/\mu'(x)))}$. It follows that

$$2^{-|C_\mu(x)|} \geq \Pr[X_{|x|} = x]/2. \qquad (10.4)$$

Recalling that $x$ is the only preimage that is mapped to $\langle M_{S,\mu}, C'_\mu(x), 1^{p(|x|)}\rangle$ and combining Eq. (10.3) & (10.4), we establish the domination condition.

The theorem follows. ■

**Reflections.** The proof of Theorem 10.17 demonstrates the fact that, unlike more advanced worst-case reductions, the reduction used in the proof of Theorem 2.18 does not introduce much structure in the reduced instances (i.e., does not reduce the original problem to a "highly structured special case" of the target problem). Put in other words, the latter reduction does not map "random" (i.e., uniformly distributed) instances to highly structured instances (which occur with negligible probability under the uniform distribution). Thus, the reduction used in the proof of Theorem 2.18 suffices for reducing any distributional problem in dist$\mathcal{NP}$ to a distributional problem consisting of $S_{\mathbf{u}}$ coupled with *some* simple probability ensemble.[14]

However, Theorem 10.17 states more than the latter assertion. That is, it states that any distributional problem in dist$\mathcal{NP}$ is reducible to the same distributional version of $S_{\mathbf{u}}$. Indeed, the effort involved in proving Theorem 10.17 was due to the need for mapping instances taken from any simple probability ensemble (which may not be the uniform ensemble) to instances distributed in a manner that is dominated by a single probability ensemble (i.e., the quasi-uniform ensemble $U'$).

Once we have established the existence of one dist$\mathcal{NP}$-complete problem, we may establish the dist$\mathcal{NP}$-completeness of other problems in dist$\mathcal{NP}$ by reducing any dist$\mathcal{NP}$-complete problem to them (and relying on the transitivity of reductions (see Exercise 10.9)). Thus, the difficulties encountered in the proof of Theorem 10.17 are no longer relevant. Unfortunately, a seemingly more severe difficulty arises: almost all know reductions in the theory of NP-completeness work by introducing much structure in the reduced instances (i.e., they actually reduce to highly structured special cases). Furthermore, this structure is too complex in the sense that the distribution of reduced instances does not seem simple (in the sense of Definition 10.15). Designing reductions that avoid the introduction of such structure has turned out to be quite difficult; still several such reductions are cited in [85].

#### 10.2.1.3 Probabilistic versions

The definitions in §10.2.1.1 can be easily extended to refer to randomized algorithms. For example, extending Definition 10.14, we have:

[14] Note that this cannot be said of most known Karp-reductions.

**Definition 10.18** (the class tpc$\mathcal{BPP}$): *For a probabilistic algorithm $A$, a Boolean function $B$, and $t : \mathbb{N} \to \mathbb{N}$, we say that the string $x$ is* $t$-bad for $A$ with respect to $B$ *if with probability exceeding* $1/3$*, on input $x$, either $A(x) \neq B(x)$ or $A$ runs more that $t(|x|)$ steps. We say that $A$* typically solves $(S, \{X_n\}_{n\in\mathbb{N}})$ in probabilistic polynomial-time *if there exists a polynomial $p$ such that the probability that $X_n$ is $p$-bad for $A$ with respect to the characteristic function of $S$ is negligible. We denote by* tpc$\mathcal{BPP}$ *the class of distributional problems that are typically solvable in probabilistic polynomial-time.*

The definition of reductions can be similarly extended. This means that in Definition 10.16, both $M^T(x)$ and $Q(x)$ (mentioned in Items 2 and 3, respectively) are random variables rather than fixed objects. Furthermore, validity is required to hold (for every input) only with probability $2/3$, where the probability space refers only to the internal coin tosses of the reduction. Randomized reductions are closed under composition and preserve typical feasibility (see Exercise 10.12).

Randomized reductions allow the presentation of a dist$\mathcal{NP}$-complete problem that refers to the (perfectly) uniform ensemble. Recall that Theorem 10.17 establishes the dist$\mathcal{NP}$-completeness of $(S_{\mathtt{u}}, U')$, where $U'$ is a quasi-uniform ensemble (i.e., $\Pr[U'_n = \langle M, x, 1^t\rangle] = 2^{-(|M|+|x|)}/\binom{n}{2}$, where $n = |\langle M, x, 1^t\rangle|$). We first note that $(S_{\mathtt{u}}, U')$ can be randomly reduced to $(S'_{\mathtt{u}}, U'')$, where $S'_{\mathtt{u}} = \{\langle M, x, z\rangle : \langle M, x, 1^{|z|}\rangle \in S_{\mathtt{u}}\}$ and $\Pr[U''_n = \langle M, x, z\rangle] = 2^{-(|M|+|x|+|z|)}/\binom{n}{2}$ for every $\langle M, x, z\rangle \in \{0,1\}^n$. The randomized reduction consists of mapping $\langle M, x, 1^t\rangle$ to $\langle M, x, z\rangle$, where $z$ is uniformly selected in $\{0,1\}^t$. Recalling that $U = \{U_n\}_{n\in\mathbb{N}}$ denotes the uniform probability ensemble (i.e., $U_n$ is uniformly distributed on strings of length $n$) and using a suitable encoding we get.

**Proposition 10.19** *There exists $S \in \mathcal{NP}$ such that every $(S', X') \in$ dist$\mathcal{NP}$ is randomly reducible to $(S, U)$.*

**Proof Sketch:** By the forgoing discussion, every $(S', X') \in$ dist$\mathcal{NP}$ is randomly reducible to $(S'_{\mathtt{u}}, U'')$. Thus, we focus on reducing $(S'_{\mathtt{u}}, U'')$ to $(S''_{\mathtt{u}}, U)$, where $S''_{\mathtt{u}} \in \mathcal{NP}$ is defined as follows. The string $\alpha\beta uvw$ is in $S''_{\mathtt{u}}$ if $\langle u, v, w\rangle \in S'_{\mathtt{u}}$ and $\alpha$ (resp., $\beta$) represents the binary encoding of the integer $|u|$ (resp., $|v|$), where the encoding is padded with zeros to a total length of $\log_2|uvw|$. The reduction maps $\langle M, x, z\rangle$ to the string $\alpha\cdot\beta\cdot M\cdot x\cdot z$, where $\alpha$ (resp., $\beta$) represents the binary encoding of $|M|$ (resp., $|x|$) padded with zeros to a total length of $\log_2(|M|+|x|+|z|)$. Noting that this reduction satisfies all conditions of Definition 10.16, the proposition follows. □

### 10.2.2 Ramifications

In our opinion, the most problematic aspect of the theory described in Section 10.2.1 is the definition of simple probability ensembles, which in turn restricts the definition of dist$\mathcal{NP}$ (Definition 10.15). This restriction strengthens the conjecture that dist$\mathcal{NP}$ is not contained in tpc$\mathcal{BPP}$, which means that it weakens conditional results that are based on this conjecture. An appealing extension of the class dist$\mathcal{NP}$ is presented in §10.2.2.2, where it is shown that if the extended class is not contained in tpc$\mathcal{BPP}$ then dist$\mathcal{NP}$ itself is not contained in tpc$\mathcal{BPP}$. Thus, dist$\mathcal{NP}$-complete problems enjoy the benefit of both being in the more restricted class (i.e., dist$\mathcal{NP}$) and being hard as long as some problems in the extended class is hard.

In §10.2.2.1, we extend the treatment from decision problems to search problems. This extension is motivated by the realization that search problem are actually of greater importance to real-life applications (cf. Section 2.1.1), and hence a theory motivated by real-life applications must address such problems, as we do next.

#### 10.2.2.1 Search versus Decision

Indeed, as in the case of worst-case complexity, search problems are at least as important as decision problems. Thus, an average-case treatment of search problems is indeed called for. We first present distributional versions of $\mathcal{PF}$ and $\mathcal{PC}$ (cf. Section 2.1.1), following the underlying principles of the definitions of tpc$\mathcal{P}$ and dist$\mathcal{NP}$.

**Definition 10.20** (the classes tpc$\mathcal{PF}$ and dist$\mathcal{PC}$): *As in Section 2.1.1, we consider only polynomially bounded search problems; that is, binary relations $R \subseteq \{0,1\}^* \times \{0,1\}^*$ such that for some polynomial $q$ it holds that $(x,y) \in R$ implies $|y| \le q(|x|)$. Recall that $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$.*

- *A* distributional search problem *consists of a polynomially bounded search problem coupled with a probability ensemble.*

- *The class* tpc$\mathcal{PF}$ *consists of all distributional search problems that are typically solvable in polynomial-time. That is, $(R, \{X_n\}_{n\in\mathbb{N}}) \in$ tpc$\mathcal{PF}$ if there exists an algorithm $A$ and a polynomial $p$ such that the probability that on input $X_n$ algorithm $A$ either errs or runs more that $p(n)$ steps is negligible, where $A$ errs on $x$ if $A(x) \notin R(x)$ in case $R(x) \neq \emptyset$ and $A(x) \neq \perp$ otherwise.*

- *A distributional search problem $(R, X)$ is in* dist$\mathcal{PC}$ *if $R \in \mathcal{PC}$ and $X$ is simple* (as in Definition 10.15).

Likewise, the class tpc$\mathcal{BPPF}$ consists of all distributional search problems that are typically solvable in *probabilistic* polynomial-time (cf., Definition 10.18). The definitions of *reductions among distributional problems*, presented in the context of decision problem, extend to search problems.

Fortunately, as in the context of worst-case complexity, the study of distributional search problems "reduces" to the study of distributional decision problems.

**Theorem 10.21** (reducing search to decision): dist$\mathcal{PC} \subseteq$ tpc$\mathcal{BPPF}$ *if and only if* dist$\mathcal{NP} \subseteq$ tpc$\mathcal{BPP}$. *Furthermore, every problem in* dist$\mathcal{NP}$ *is reducible to some problem in* dist$\mathcal{PC}$*, and every problem in* dist$\mathcal{PC}$ *is* randomly *reducible to some problem in* dist$\mathcal{NP}$.

**Proof Sketch:** The furthermore part is analogous to the actual contents of the proof of Theorem 2.6 (see also Step 1 in the proof of Theorem 2.15). Indeed the reduction of $\mathcal{NP}$ to $\mathcal{PC}$ presented in the proof of Theorem 2.6 extends to the current context. Specifically, for any $S \in \mathcal{NP}$, we consider a relation $R \in \mathcal{PC}$ such that $S = \{x : R(x) \neq \emptyset\}$, and note that, for any probability ensemble $X$, the identity transformation reduces $(S, X)$ to $(R, X)$.

A difficulty arises in the opposite direction. Recall that in the proof of Theorem 2.6 we reduced the search problem of $R \in \mathcal{PC}$ to deciding membership in $S'_R \stackrel{\text{def}}{=} \{\langle x, y'\rangle : \exists y'' \text{ s.t. } (x, y'y'') \in R\} \in \mathcal{NP}$. The difficulty encountered here is that, on input $x$, this reduction makes queries of the form $\langle x, y'\rangle$, where $y'$ is a prefix of some string in $R(x)$. These queries may induce a distribution that is not dominated by any simple distribution. Thus, we seek an alternative reduction.

As a warm-up, let us assume for a moment that $R$ has unique solutions (in the sense of Definition 6.26); that is, for every $x$ it holds that $|R(x)| \leq 1$. In this case we may easily reduce the search problem of $R \in \mathcal{PC}$ to deciding membership in $S''_R \in \mathcal{NP}$, where $\langle x, i, \sigma\rangle \in S''_R$ if and only if $R(x)$ contains a string in which the $i^{\text{th}}$ bit equals $\sigma$. Specifically, on input $x$, the reduction issues the queries $\langle x, i, \sigma\rangle$, where $i \in [\ell]$ (with $\ell = \text{poly}(|x|)$) and $\sigma \in \{0, 1\}$, which allows for determining the single string in $R(x)$ (whenever such a string exists). The point is that this reduction can be used to reduce any $(R, X) \in \text{dist}\mathcal{PC}$ (having unique solutions) to $(S''_R, X'') \in \text{dist}\mathcal{NP}$, where $X''$ equally distributes the probability mass of $x$ (under $X$) to all the tuples $\langle x, i, \sigma\rangle$; that is, for every $i \in [\ell]$ and $\sigma \in \{0, 1\}$, it holds that $\mathsf{Pr}[X''_{|\langle x,i,\sigma\rangle|} = \langle x, i, \sigma\rangle]$ equals $\mathsf{Pr}[X_{|x|} = x]/2\ell$.

Unfortunately, in the general case, $R$ may not have unique solutions. Nevertheless, applying the main idea that underlies the proof of Theorem 6.27, this difficulty can be overcome. We first note that the foregoing mapping of instances of the distributional problem $(R, X) \in \text{dist}\mathcal{PC}$ to instances of $(S''_R, X'') \in \text{dist}\mathcal{NP}$ satisfies the efficiency and domination condition even in the case that $R$ does not have unique solutions. What may possibly fail (in the general case) is the validity condition (i.e., if $|R(x)| > 1$ then we may fail to recover any element of $R(x)$).

Recall that the main part of the proof of Theorem 6.27 is a randomized reduction that maps instances of $R$ to triples of the form $(x, m, h)$, where $m$ is an integer and $h$ is a hashing function that are uniformly distributed in some adequate sets $[\ell]$ and $H^m_\ell$, where $\ell = \text{poly}(|x|)$ and $H^m_\ell$ is as in Appendix D.2. Furthermore, if $R(x) \neq \emptyset$ then, with probability $\Omega(1/\ell)$ over yje choices of $m \in [\ell]$ and $h \in H^m_\ell$, there exists a unique $y \in R(x)$ such that $h(y) = 0^m$. Defining $R'(x, m, h) \stackrel{\text{def}}{=} \{y \in R : h(y) = 0^m\}$, this yields a randomized reduction of the search problem of $R$ to the search problem of $R'$ such that with noticeable probability[15] the reduction maps instances that have solutions to instances having a unique solution. Furthermore, this reduction can be used to reduce any $(R, X) \in \text{dist}\mathcal{PC}$ to $(R', X') \in \text{dist}\mathcal{PC}$, where $X'$ distributes the probability mass of $x$ (under $X$) to all the triples $(x, m, h)$ such that for every $m \in [\ell]$ and $h \in H^m_\ell$ it holds that $\mathsf{Pr}[X''_{|(x,m,h)|} = (x, m, h)]$

[15] Recall that the probability of an event is said to be noticeable (in a relevant parameter) if it is greater than the reciprocal of some positive polynomial. In the context of randomized reductions, the relevant parameter is the length of the input to the reduction.

equals $\Pr[X_{|x|} = x]/(\ell \cdot |H_\ell^m|)$. (Note that with a suitable encoding, $X'$ is indeed simple.)

The theorem follows by combining the two aforementioned reductions. That is, we first apply the randomized reduction of $(R, X)$ to $(R', X')$, and next reduce the resulting instance to an instance of the corresponding decision problem $(S''_{R'}, X'')$. The combined randomized mapping satisfies the efficiency and domination conditions, and is valid with noticeable probability. The error probability can be made negligible by straightforward amplification (see Exercise 10.12). □

#### 10.2.2.2 Simple versus sampleable distributions

Recall that the definition of simple probability ensembles (underlying Definition 10.15) requires that the accumulating distribution function (as defined in the proof of Theorem 10.17) is polynomial-time computable. Recall that $\mu : \{0,1\}^* \to [0,1]$ is called the accumulating distribution function of $X = \{X_n\}_{n\in\mathbb{N}}$ if for every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$ it holds that $\mu(x) \stackrel{\rm def}{=} \Pr[X_n \leq x]$, where the inequality refers to the standard lexicographic order of $n$-bit strings.

As argued in §10.2.1.1, the requirement that the accumulating distribution function is polynomial-time computable imposes severe restrictions on the set of admissible ensembles. Furthermore, it seems that these simple ensembles are indeed "simple" in some intuitive sense and hence represent distributions that may occur in practice. However, a more robust definition of the latter is offered by the notion of polynomial-time sampleable ensembles (underlying Definition 10.22). We believe that the class of such ensembles contains all distributions that may occur in practice, because we believe that the real world should be modeled as a *feasible* (rather than an arbitrary) randomized process

**Definition 10.22** (sampleable ensembles and the class samp$\mathcal{NP}$): *We say that a probability ensemble $X = \{X_n\}_{n\in\mathbb{N}}$ is* (polynomial-time) sampleable *if there exists a probabilistic polynomial-time algorithm $A$ such that for every $x \in \{0,1\}^*$ it holds that $\Pr[A(1^{|x|}) = x] = \Pr[X_{|x|} = x]$. We denote by* samp$\mathcal{NP}$ *the class of distributional problems consisting of decision problems in $\mathcal{NP}$ coupled with sampleable probability ensembles.*

We first note that all simple probability ensembles are indeed sampleable (see Exercise 10.13), and thus dist$\mathcal{NP} \subseteq$ samp$\mathcal{NP}$. On the other hand, it seems that there are sampleable probability ensembles that are not simple (see Exercise 10.14). In fact, extending the scope of distributional problems (from dist$\mathcal{NP}$ to samp$\mathcal{NP}$) allows proving that every NP-complete problem has a distributional version in samp$\mathcal{NP}$ that is dist$\mathcal{NP}$-hard (see Exercise 10.15). Furthermore, it is possible to prove that all natural NP-complete problem have distributional versions that are samp$\mathcal{NP}$-complete.

**Theorem 10.23** (samp$\mathcal{NP}$-completeness): *Suppose that $S \in \mathcal{NP}$ and that every set in $\mathcal{NP}$ is reducible to $S$ by a Karp-reduction that does not shrink the input. Then there exists a polynomial-time sampleable ensemble $X$ such that any problem in* samp$\mathcal{NP}$ *is reducible to $(S, X)$*

The proof of Theorem 10.23 is based on the observation that there exists a polynomial-time sampleable ensemble that dominates all polynomial-time sampleable ensembles. The existence of this ensemble is based on the notion of a universal (sampling) machine. For further details see Exercise 10.16. (Recall that when proving Theorem 10.17, we did not establish an analogous result for simple ensembles (but rather capitalized on the universal nature of $S_{\mathtt{u}}$).)

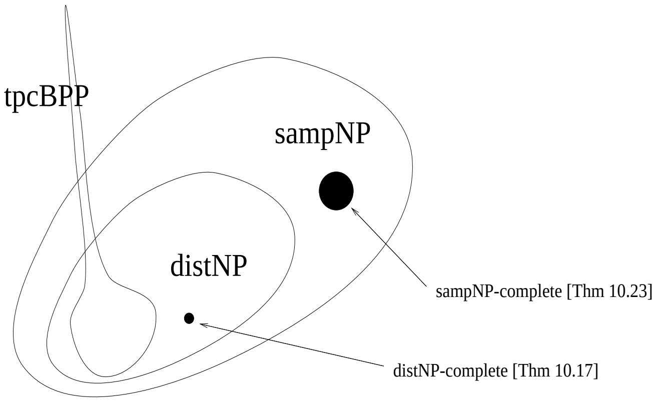


Figure 10.1: Two types of average-case completeness

Theorem 10.23 establishes a rich theory of samp$\mathcal{NP}$-completeness, but does not relate this theory to the previously presented theory of dist$\mathcal{NP}$-completeness (see Figure 10.1). This is done in the next theorem, which asserts that the existence of typically hard problems in samp$\mathcal{NP}$ implies their existence in dist$\mathcal{NP}$.

**Theorem 10.24** (samp$\mathcal{NP}$-completeness versus dist$\mathcal{NP}$-completeness): *If* samp$\mathcal{NP}$ *is not contained in* tpc$\mathcal{BPP}$ *then* dist$\mathcal{NP}$ *is not contained in* tpc$\mathcal{BPP}$.

Thus, the two "typical-case complexity" versions of the P-vs-NP Question are equivalent. That is, if some "sampleable distribution" versions of NP are not typically feasible then some "simple distribution" versions of NP are not typically feasible. In particular, if samp$\mathcal{NP}$-complete problems are not in tpc$\mathcal{BPP}$ then dist$\mathcal{NP}$-complete problems are not in tpc$\mathcal{BPP}$.

The foregoing assertions would all follow if samp$\mathcal{NP}$ were (randomly) reducible to dist$\mathcal{NP}$ (i.e., if every problem in samp$\mathcal{NP}$ were reducible (under a randomized version of Definition 10.16) to some problem in dist$\mathcal{NP}$); but, unfortunately, we do not know whether such reductions exist. Yet, underlying the proof of Theorem 10.24 is a more liberal notion of a reduction among distributional problem.

**Proof Sketch:** We shall prove that if dist$\mathcal{NP}$ is contained in tpc$\mathcal{BPP}$ then the same holds for samp$\mathcal{NP}$ (i.e., samp$\mathcal{NP}$ is contained in tpc$\mathcal{BPP}$). Actually, we shall show that if dist$\mathcal{PC}$ is contained in tpc$\mathcal{BPPF}$ then the sampleable version of dist$\mathcal{PC}$, denoted samp$\mathcal{PC}$, is contained in tpc$\mathcal{BPPF}$ (and refer to Exercise 10.17).

Specifically, we shall show that under a relaxed notion of a randomized reduction, every problem in samp$\mathcal{PC}$ is reduced to some problem in dist$\mathcal{PC}$. Loosely speaking, this relaxed notion of a randomized reduction requires only a noticeable fraction of the probability space of the reduction to satisfies the validity and domination conditions (of Definition 10.16, when adapted to randomized reductions). We start by formulating this notion, when referring to distributional *search* problems.

A relaxed reduction of the distributional problem $(R, X)$ to the distributional problem $(T, Y)$ is a probabilistic polynomial-time oracle machine $M$ that satisfies the following conditions:

Notation: For every $x \in \{0,1\}^*$, we denote by $m(|x|) = \text{poly}(|x|)$ the number of internal coin tosses of $M$ on input $x$, and denote by $M^T(x, r)$ the execution of $M$ on input $x$, internal coins $r \in \{0,1\}^m$, and oracle access to $T$.

Validity: For every $x \in \{0,1\}^*$, there exists a set $\Omega_x \subseteq \{0,1\}^{m(|x|)}$ of size at least $\rho(|x|) \cdot 2m(|x|)$, where $\rho(|x|) > 1/\text{poly}(|x|)$ such that for every $r \in \Omega_x$ the reduction yields a correct answer (i.e., $M^T(x, r) \in R(x)$ if $R(x) \neq \emptyset$ and $M^T(x, r) = \perp$ otherwise).

Domination: There exists a positive polynomial $p$ such that, for every $y \in \{0,1\}^*$, it holds that

$$\max_{n \in \mathbb{N}} \{\Pr[Q'(X_n) \ni y]\} \leq p(|y|) \cdot \Pr[Y_{|y|} = y], \tag{10.5}$$

where $Q'(x)$ is a random variable, defined over the set $\Omega_x$ (as in the validity condition), representing the set of queries made by $M$ on input $x$ and oracle access to $T$. That is, $Q'(x)$ is defined by uniformly selecting $r \in \Omega_x$ and considering the set of queries made by $M$ on input $x$, internal coins $r$, and oracle access to $T$.

The reader may verify that this relaxed notion of a reduction preserves typical feasibility; that is, for $R \in \mathcal{PC}$, if there exists a relaxed reduction of $(R, X)$ to $(T, Y)$ and $(T, Y)$ is in tpc$\mathcal{BPPF}$ then $(R, X)$ is in tpc$\mathcal{BPPF}$. The key observation is that the analysis may discard the case that, on input $x$, the reduction selects coins not in $\Omega_x$. Indeed, the queries made in that case may be untypical and the answers received may be wrong, but this is immaterial (because correct solutions can be recognized using $R \in \mathcal{PC}$). That is, if $x$ has a solution then with noticeable probability the reduction will find one and output it, whereas the reduction will never output a wrong solution.

Our goal is presenting, for every $(R, X) \in$ samp$\mathcal{PC}$, a relaxed reduction of $(R, X)$ to a related problem $(R', X') \in$ dist$\mathcal{PC}$. (As usual, let $X = \{X_n\}_{n \in \mathbb{N}}$ and similarly for $X'$.) For starters, *suppose that $X_n$ is uniformly distributed on some set $S_n \subseteq \{0,1\}^n$ and that there is a polynomial-time computable and invertible mapping $\mu$ of $S_n$ to $\{0,1\}^{\ell(n)}$, where $\ell(n) = \log_2 |S_n|$*. Then, mapping $x$ to $1^{|x| - \ell(|x|)}0\mu(x)$, we obtain a reduction of $(R, X)$ to $(R', X')$, where $X'_{n+1}$ is uniform over $\{1^{n-\ell(n)}0v : v \in \{0,1\}^{\ell(n)}\}$ and $R'(1^{n-\ell(n)}0v) = R(\mu^{-1}(v))$ (or, equivalently, $R(x) = R'(1^{|x|-\ell(|x|)}0\mu(x))$). Note that $X'$ is a simple ensemble and $R' \in \mathcal{PC}$;

hence, $(R', X') \in \text{dist}\mathcal{PC}$. Also note that the foregoing mapping is indeed a valid reduction (i.e., it satisfies the efficiency, validity, and domination conditions). Thus, $(R, X)$ is reduced to a problem in $\text{dist}\mathcal{PC}$ (and indeed the relaxation was not used here).

Next, we drop the assumption that there is a polynomial-time computable and invertible mapping $\mu$ of $S_n$ to $\{0,1\}^{\ell(n)}$, but maintain the assumption that $X_n$ is uniform on some set $S_n \subseteq \{0,1\}^n$ and assume that $|S_n| = 2^{\ell(n)}$ is easily computable (from $n$). In this case, we may map $x \in \{0,1\}^n$ to its image under a suitable randomly chosen hashing function $h$, which in particular maps $n$-bit strings to $\ell(n)$-bit strings. That is, we randomly map $x$ to $(h, 1^{n-\ell(n)}0h(x))$, where $h$ is uniformly selected in a set $H_n^{\ell(n)}$ of suitable hash functions (see Appendix D.2). This calls for redefining $R'$ such that $R'(h, 1^{n-\ell(n)}0v)$ corresponds to the preimages of $v$ under $h$ that are in $S_n$. Assuming that $h$ is a 1-1 mapping of $S_n$ to $\{0,1\}^{\ell(n)}$, we may define $R'(h, 1^{n-\ell(n)}0v) = R(x)$ where $x$ is the unique string satisfying $x \in S_n$ and $h(x) = v$, where the condition $x \in S_n$ may be *verified by providing the internal coins of the sampling procedure that generate $x$*. Denoting the sampling procedure of $X$ by $S$, and letting $S(1^n, r)$ denote the output of $S$ on input $1^n$ and internal coins $r$, we actually redefine $R'$ as

$$R'(h, 1^{n-\ell(n)}0v) = \{\langle r, y\rangle : h(S(1^n, r)) = v \wedge y \in R(S(1^n, r))\}. \tag{10.6}$$

We note that $\langle r, y\rangle \in R'(h, 1^{|x|-\ell(|x|)}0h(x))$ yields a solution $y \in R(x)$ if $S(1^{|x|}, r) = x$, but otherwise "all bets are off" (as $y$ will be a solution for $S(1^{|x|}, r) \neq x$). Now, although typically $h$ will not be a 1-1 mapping of $S_n$ to $\{0,1\}^{\ell(n)}$, for each $x \in S_n$, with constant probability over the choice of $h$, it holds that $h(x)$ has a unique preimage in $S_n$ under $h$. (See the proof of Theorem 6.27.) In this case $\langle r, y\rangle \in R'(h, 1^{|x|-\ell(|x|)}0h(x))$ implies $S(1^{|x|}, r) = x$ (which, in turn, implies $y \in R(x)$). We claim that *the randomized mapping of $x$ to $(h, 1^{n-\ell(n)}0h(x))$, where $h$ is uniformly selected in $H_{|x|}^{\ell(|x|)}$, yields a relaxed reduction of $(R, X)$ to $(R', X')$, where $X'_{n'}$ is uniform over $H_n^{\ell(n)} \times \{1^{n-\ell(n)}0v : v \in \{0,1\}^{\ell(n)}\}$*. (Needless to say, the claim refers to the reduction that makes the query $(h, 1^{n-\ell(n)}0h(x))$ and returns $y$ if the oracle answer equals $\langle r, y\rangle$ and $y \in R(x)$.)

The claim is proved by considering the set $\Omega_x$ of choices of $h \in H_{|x|}^{\ell(|x|)}$ for which $x \in S_n$ is the only preimage of $h(x)$ under $h$ that resides in $S_n$ (i.e., $|\{x' \in S_n : h(x') = h(x)\}| = 1$). In this case (i.e., $h \in \Omega_x$) it holds that $\langle r, y\rangle \in R'(h, 1^{|x|-\ell(|x|)}0h(x))$ implies that $S(1^{|x|}, r) = x$ and $y \in R(x)$, and the (relaxed) validity condition follows. The (relaxed) domination condition follows by noting that $\Pr[X_n = x] \approx 2^{-\ell(|x|)}$, that $x$ is mapped to $(h, 1^{|x|-\ell(|x|)}0h(x))$ with probability $1/|H_{|x|}^{\ell(|x|)}|$, and that $x$ is the only preimage of $(h, 1^{|x|-\ell(|x|)}0h(x))$ under the mapping (among $x' \in S_n$ such that $\Omega_{x'} \ni h$).

Before going any further, let us highlight the importance of hashing $X_n$ to $\ell(n)$-bit strings. On one hand, this mapping is "typically" one-to-one, and thus (with constant probability) the solution provided for the hashed instance (i.e., $h(x)$) yield a solution for the original instance (i.e., $x$). This guarantees the validity of the reduction. On the other hand, for a typical $h$, the mapping of $X_n$ to $h(X_n)$ covers the relevant range almost uniformly. This guarantees that the reduction satisfies the domination condition. Note that these two phenomena impose conflicting requirements that are both met at the correct value of $\ell$; that is, the one-to-one condition requires $\ell(n) \geq \log_2 |S_n|$, whereas an almost uniform cover requires $\ell(n) \leq \log_2 |S_n|$. Also note that $\ell(n) = \log_2(1/\mathsf{Pr}[X_n = x])$ for every $x$ in the support of $X_n$; the latter quantity will be in our focus in the general case.

Finally, we need to get rid of the assumption that $X_n$ is *uniformly distributed* over some subset of $\{0,1\}^n$. All that we know is that there exists a probabilistic polynomial-time ("sampling") algorithm $S$ such that $S(1^n)$ is distributed identically to $X_n$. In this (general) case, we map instances of $(R, X)$ according to their probability mass such that $x$ is mapped to an instance (of $R'$) that consists of $(h, h(x))$ and additional information, where $h$ is a random hash function mapping $n$-bit long string to strings of length $\ell_x \stackrel{\text{def}}{=} \lceil \log_2(1/\mathsf{Pr}[X_{|x|} = x]) \rceil$. Since (in the general case) there may be more than $2^{\ell_x}$ strings in the support of $X_n$, we need to augment the reduced instance in order to ensure that it is uniquely associated with $x$. The basic idea is augmenting the mapping of $x$ to $(h, h(x))$ with additional information that restricts $X_n$ to strings that occur with probability at least $2^{-\ell_x}$.

Let $q(n)$ denote the randomness complexity of $S$ and $S(1^n, r)$ denote the output of $S$ on input $1^n$ and internal coin tosses $r \in \{0,1\}^{q(n)}$. Then, we randomly map $x$ to $(h, h(x), h', v')$, where $h : \{0,1\}^{|x|} \to \{0,1\}^{\ell_x}$ and $h' : \{0,1\}^{q(|x|)} \to \{0,1\}^{q(|x|)-\ell_x}$ are random hash functions and $v' \in \{0,1\}^{q(|x|)-\ell_x}$ is uniformly distributed. The instance $(h, v, h', v')$ of $R'$ has solutions that consists of pairs $\langle r, y \rangle$ such that $h(S(1^n, r)) = v \wedge h'(r) = v'$ and $y \in R(S(1^n, r))$. As we shall see, this augmentation guarantees that, with constant probability (over the choice of $h, h', v'$), the solutions to the reduced instance $(h, h(x), h', v')$ correspond to the solutions to the original instance $x$.

The foregoing description assumes that, on input $x$, we can determine $\ell_x$, which is an assumption that we cannot justify. Instead, we may just select $\ell$ uniformly in $\{0, 1, ..., q(|x|)\}$ and be correct with noticeable probability (i.e., $\mathsf{Pr}[\ell = \ell_x] = 1/(q(|x|) + 1) = 1/\mathrm{poly}(|x|)$). Furthermore, for clarity, we make $n$ and $\ell$ explicit in the reduced instance. Specifically, we randomly map $x \in \{0,1\}^n$, to $(1^n, 1^\ell, h, h(x), h', v') \in \{0,1\}^{n'}$, where $\ell \in \{0, 1, ..., q(n)\}$, $h \in H_n^\ell$, $h' \in H_{q(n)}^{q(n)-\ell}$, and $v' \in \{0,1\}^{q(n)-\ell}$ are uniformly distributed.[16] This mapping will be used to reduce $(R, X)$ to $(R', X')$, where

$$R'(1^n, 1^\ell, h, v, h', v') = \{\langle r, y \rangle : h(S(1^n, r)) = v \wedge h'(r) = v' \wedge y \in R(S(1^n, r))\} \quad (10.7)$$

and $X'_{n'}$ assigns equal probability to each $X_{n',\ell}$ (for $\ell \in \{0, 1, ..., n\}$), and each $X_{n',\ell}$ is isomorphic to the uniform distribution over $H_n^\ell \times \{0,1\}^\ell \times H_{q(n)}^{q(n)-\ell} \times \{0,1\}^{q(n)-\ell}$. Note that indeed $(R', X') \in \mathrm{dist}\mathcal{PC}$.

The aforementioned randomized mapping is analyzed by considering the correct choice for $\ell$; that is, on input $x$, we focus on the choice $\ell = \ell_x$. Under this condi-

[16] As in other places, a suitable encoding will be used such that the reduction maps strings of the same length to strings of the same length (i.e., $n$-bit string are mapped to $n'$-bit strings, for $n' = \mathrm{poly}(n)$). For example, we may encode $\langle 1^n, 1^\ell, h, h(x), h', v' \rangle$ as $1^n 0 1^\ell 0 1^{q(n)-\ell} 0 \langle h \rangle \langle h(x) \rangle \langle h' \rangle \langle v' \rangle$, where each $\langle w \rangle$ denotes an encoding of $w$ by a string of length $(n' - (n + q(n) + 3))/4$.

tioning (as we shall show), *with constant probability over the choice of* $h, h'$ *and* $v'$, *the instance* $x$ *is the only value* $x'$ (of $X_n$) *that is mapped to* $(1^n, 1^\ell, h, h(x), h', v')$ *such that there exists* $r$ *that satisfies* $S(1^n, r) = x'$ *and* $h'(r) = v'$. It follows that (for such $h, h'$ and $v'$) any solution $\langle r, y\rangle \in R'(1^n, 1^\ell, h, h(x), h', v')$ satisfies $S(1^n, r) = x$ and thus $y \in R(x)$, which means that the validity condition is satisfied. The domination condition is satisfied too, because (for such $h, h'$ and $v'$) the probability that $X_n$ is mapped to $(1^n, 1^\ell, h, h(x), h', v')$ approximately equals $\mathsf{Pr}[X'_{n',\ell} = (1^n, 1^\ell, h, h(x), h', v')]$.

We now turn to analyze the probability, over the choice of $h, h'$ and $v'$, that the instance $x$ is the only value $x'$ (of $X_n$) that is mapped to $(1^n, 1^{\ell_x}, h, h(x), h', v')$ such that there exists $r$ that satisfies $S(1^n, r) = x'$ and $h'(r) = v'$. Firstly, we note that $|\{r : S(1^n, r) = x\}| \geq 2^{q(n)-\ell_x}$, and thus, with constant probability over the choice of $h' \in H_{q(n)}^{q(n)-\ell_x}$ and $v' \in \{0,1\}^{q(n)-\ell_x}$, there exists $r$ that satisfies $S(1^n, r) = x$ and $h'(r) = v'$. Next, we note that, with constant probability over the choice of $h \in H_n^{\ell_x}$, it holds that $x$ is the only string having probability mass at least $2^{-\ell_x}$ (under $X_n$) that is mapped to $h(x)$ under $h$. Finally, we prove that, with constant probability over the choice of $h \in H_n^{\ell_x}$ and $h' \in H_{q(n)}^{q(n)-\ell_x}$ (and even when conditioning on the previous items), the mapping $r \mapsto (h(S(1^n, r)), h'(r))$ maps the set $\{r : \mathsf{Pr}[X_n = S(1^n, r)] \leq 2^{-\ell_x}\}$ almost uniformly to $\{0,1\}^{q(n)}$. Specifically, with constant probability, no other $r$ is mapped to the aforementioned pair $(h(x), v')$. Thus, the claim follows and so does the theorem. □

**Reflection.** Theorem 10.24 implies that if samp$\mathcal{NP}$ is not contained in tpc$\mathcal{BPP}$ then every dist$\mathcal{NP}$-complete problem is not in tpc$\mathcal{BPP}$. This means that the hardness of some distributional problems that refer to sampleable distributions implies the hardness of some distributional problems that refer to simple distributions. Furthermore, by Proposition 10.19, this implies the hardness of distributional problems that refer to the uniform distribution. Thus, hardness with respect to some distribution in an utmost wide class (which arguably captures all distributions that may occur in practice) implies hardness with respect to a single simple distribution (which arguably is the simplest one).

**Relation to one-way functions.** We note that the existence of one-way functions (see Section 7.1) implies the existence of problems in samp$\mathcal{PC}$ that are not in tpc$\mathcal{BPPF}$ (which in turn implies the existence of such problems in dist$\mathcal{PC}$). Specifically, for a length-preserving one-way function $f$, consider the distributional search problem $(R_f, \{f(U_n)\}_{n\in\mathbb{N}})$, where $R_f = \{(f(r), r) : r \in \{0,1\}^*\}$.[17] On the other hand, it is not known whether the existence of a problem in samp$\mathcal{PC} \setminus$ tpc$\mathcal{BPPF}$ implies the existence of one-way functions. In particular, the existence of a problem $(R, X)$ in samp$\mathcal{PC} \setminus$ tpc$\mathcal{BPPF}$ represents the feasibility of generating hard instances for the search problem $R$, whereas the existence of one-way function represents the feasibility of generating instance-solution pairs such that the instances

[17] Note that the distribution $f(U_n)$ is uniform in the special case that $f$ is a permutation over $\{0,1\}^n$.

are hard to solve (see Section 7.1.1). Indeed, the gap refers to whether or not *hard instances can be efficiently generated together with corresponding solutions.* Our world view is thus depicted in Figure 10.2, where lower levels indicate seemingly weaker assumptions.

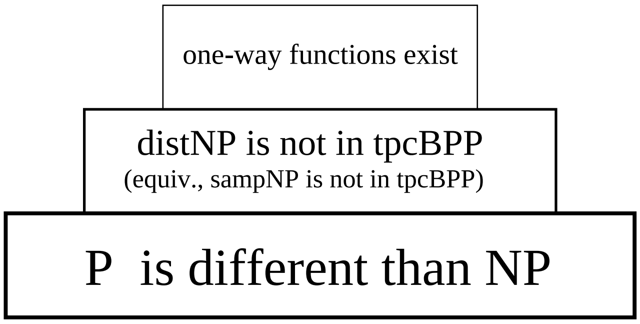


Figure 10.2: Worst-case vs average-case assumptions

## Chapter Notes

In this chapter, we presented two different approaches to the relaxation of computational problems. The first approach refers to the concept of approximation, while the second approach refers to average-case analysis. We demonstrated that various natural notions of approximation can be cast within the framework of promise problems (see Section 2.4.1). In contrast, the study of average-case complexity requires the introduction of a new conceptual framework and addressing various definitional issues.

A natural question at this point is what have we gained by relaxing the requirements. In the context of approximation, the answer is mixed: in some natural cases we gain alot (i.e., we obtained feasible relaxations of hard problems), while in other natural cases we gain nothing (i.e., even extreme relaxations remain as intractable as the original version). In the context of average-case complexity, the answer appears to be negative. In particular, assuming the existence of one-way functions, every natural NP-complete problem has a distributional version that is hard (and refers to a sampleable ensemble). Furthermore, in this case, some problems in NP have hard distributional versions that refer to the uniform distribution.

Another difference between the two approaches is that the theory of approximation seems to lack a comprehensive structure, whereas the theory of average-case complexity seems to have a too rigid structure (which foils attempts to present more appealing dist$\mathcal{NP}$-complete problems).

### Approximation

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 [166]), the systematic study of the complexity of such problems stalled till the discovery of the "PCP connection" (see Section 9.3.3) by Feige, Goldwasser, Lovász, and Safra [69]. Indeed the relatively "tight" inapproximation results for max-Clique, max-SAT, and the maximization of linear equations, due to Håstad [111, 112], build on previous work regarding PCP and their connection to approximation (cf., e.g., [70, 14, 13, 27, 172]). Specifically, Theorem 10.5 is due to [111], while Theorems 10.8 and 10.9 are due to [112]. The best known inapproximation result for minimum Vertex Cover (see Theorem 10.7) is due to [65], but we doubt it is tight (see, e.g., [133]). Reductions among approximation problems were defined and presented in [166]; see Exercise 10.3, which presents a major technique introduced in [166]. For general texts on approximation algorithms and problems (as discussed in Section 10.1.1), the interested reader is referred to the surveys collected in [117]. A compendium of NP optimization problems is available at [61].

**Property testing.** The study of property testing was initiated by Rubinfeld and Sudan [182] and re-initiated by Goldreich, Goldwasser, and Ron [93]. While the focus of [182] was on algebraic properties such as low-degree polynomials, the focus of [93] was on graph properties (and Theorem 10.12 is taken from [93]). The model of bounded-degree graphs was introduced in [99] and Theorem 10.13 combines results from [99, 100, 39]. For surveys of the area, the interested reader is referred to [73, 181].

**Average-case complexity**

The theory of average-case complexity was initiated by Levin [144], who in particular proved Theorem 10.17. In light of the laconic nature of the original text [144], we refer the interested reader to a survey [85], which provides a more detailed exposition of the definitions suggested by Levin as well as a discussion of the considerations underlying these suggestions. (This survey [85] provides also a brief account of further developments.)

As noted in §10.2.1.1, the current text uses a variant of the original definitions. In particular, our definition of "typical feasibility" differs from the original definition of "average feasibility" in totally discarding exceptional instances and in even allowing the algorithm to fail on them (and not merely run for an excessive amount of time). The alternative definition was suggested by several researchers and appears as a special case of the general treatment provided in [41].

Section 10.2.2 is based on [28, 120]. Specifically, Theorem 10.21 (or rather the reduction of search to decision) is due to [28] and so is the introduction of the class samp$\mathcal{NP}$. A version of Theorem 10.24 was proven in [120], and our proof follows their ideas, which in turn are closely related to the ideas underlying the proof of Theorem 8.11 (proved in [113]).

Recall that we know of the existence of problems in dist$\mathcal{NP}$ that are hard provided samp$\mathcal{NP}$ contains hard problems. However, these problems refer to somewhat generic decision problems such as $S_{\mathtt{u}}$. The presetation of dist$\mathcal{NP}$-complete problems that combine a more natural decision problem (like `SAT` or `Clique`) with a simple probability ensemble is an open problem.

# Exercises

**Exercise 10.1 (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 10.2 (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 10.3 (enforcing multi-way equalities via expanders)** The aim of this exercise is presenting a major technique of Papadimitriou and Yannakakis [166], 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 $\varepsilon$) is a fixed constant. Note that the standard reduction of `3SAT` to the corresponding special case (see proof of Proposition 2.22) does not preserve an approximation gap.[18] The idea is enforcing equality of the values assigned

[18] 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 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.

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 (see Appendix E.2). 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 $\mathtt{gapSAT}^3_{0.1}$ to $\mathtt{gapSAT}^{3,c}_{\varepsilon}$.

**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 $\tau'$ to the reduced formula $\phi'$ (i.e., the formula obtained by mapping $\phi$). For each original variable $z$, if $\tau'$ assigns the same value to almost all copies of $z$ then we consider the corresponding assignment in $\phi$. Otherwise, $\tau'$ does not satisfy a constant fraction of the clauses containing a copy of $z$.

**Exercise 10.4 (an equivalent definition of** tpc$\mathcal{P}$**)** Prove that $(S,X) \in \text{tpc}\mathcal{P}$ if and only if there exists a polynomial-time algorithm $A$ such that the probability that $A(X_n)$ errs (in determining membership in $S$) is a negligible function in $n$.

**Exercise 10.5 (**tpc$\mathcal{P}$ **versus** $\mathcal{P}$ **– Part 1)** Prove that tpc$\mathcal{P}$ contains a problem $(S,X)$ such that $S$ is not even recursive. Furthermore, use $X = U$.

**Guideline:** Let $S = \{0^{|x|}x : x \in S'\}$, where $S'$ is an arbitrary (non-recursive) set.

**Exercise 10.6 (**tpc$\mathcal{P}$ **versus** $\mathcal{P}$ **– Part 2)** Prove that there exists a distributional problem $(S,X)$ such that $S \notin \mathcal{P}$ and yet there exists an algorithm solving $S$ (correctly on all inputs) in time that is typically polynomial with respect to $X$. Furthermore, use $X = U$.

**Guideline:** For any time-constructible function $t : \mathbb{N} \to \mathbb{N}$ that is super-polynomial and sub-exponential, use $S = \{0^{|x|}x : x \in S'\}$ for any $S' \in \text{DTIME}(t) \setminus \mathcal{P}$.

**Exercise 10.7 (reductions preserve typical polynomial-time solveability)** Prove that if the distributional problem $(S,X)$ is reducible to the distributional problem $(S',X')$ and $(S',X') \in \text{tpc}\mathcal{P}$, then $(S,X)$ is in tpc$\mathcal{P}$.

**Guideline:** Let $B'$ denote the set of exceptional instances for the distributional problem $(S',X')$ (i.e., the set of instances on which the solver in the hypothesis errs or exceeds the typical running-time). Prove that $\Pr[Q(X_n) \cap B' \neq \emptyset]$ is a negligible function (in $n$), using both $\Pr[y \in Q(X_n)] \leq p(|y|) \cdot \Pr[X'_{|y|} = y]$ and $|x| \leq p(|y|)$ for every $y \in Q(x)$. Specifically, use the latter condition for inferring that $\sum_{y \in B'} \Pr[y \in Q(X_n)]$ equals $\sum_{y \in \{y' \in B' : p(|y'|) \geq n\}} \Pr[y \in Q(X_n)]$, which guarantees that a negligible function in $|y|$ for any $y \in Q(X_n)$ is negligible in $n$.

**Exercise 10.8 (reductions preserve error-less solveability)** In continuation to Exercise 10.7, prove that reductions preserve error-less solveability (i.e., solveability by algorithms that never err and typically run in polynomial-time).

**Exercise 10.9 (transitivity of reductions)** Prove that reductions among distributional problems (as in Definition 10.16) are transitive.

**Guideline:** The point is establishing the domination property of the composed reduction. The hypothesis that reductions do not make too short queries is instrumental here.

**Exercise 10.10** For any $S \in \mathcal{NP}$ present a simple probability ensemble $X$ such that the generic reduction used in the proof of Theorem 2.18 violates the domination condition regarding a possible reduction of $(S, X)$ to $(S_{\mathtt{u}}, U')$.

**Guideline:** Consider $X = \{X_n\}_{n\in\mathbb{N}}$ such that $X_n$ is uniform over $\{0^{n/2}x' : x' \in \{0,1\}^{n/2}\}$.

**Exercise 10.11 (variants of the Coding Lemma)** Prove the following two variants of the Coding Lemma (which is stated in the proof of Theorem 10.17).

1. A variant that refers to any monotonically non-decreasing function $\mu : \{0,1\}^* \to [0,1]$ that is efficiently computable, where here we refer to the lexicographic order over $\{0,1\}^*$.

2. As in Part 1, except that in this variant the function $\mu$ is strictly increasing and the compression condition requires that $|C_\mu(x)| \leq \log_2(1/\mu'(x))$ rather than $|C_\mu(x)| \leq 1 + \min\{|x|, \log_2(1/\mu'(x))\}$, where $\mu'(x) \stackrel{\rm def}{=} \mu(x) - \mu(x-1)$.

**Guideline:** In both cases, the proof is less cumbersome than the one presented in the main text.

**Exercise 10.12 (randomized reductions)** Following the outline in §10.2.1.3, provide a definition of randomized reductions among distributional problems.

1. Prove that randomized reductions are transitive (cf. Exercise 10.9).

2. In analogy to Exercise 10.7, prove that randomized reductions preserve feasible solveability (i.e., typical solveability in probabilistic polynomial-time). That is, if the distributional problem $(S, X)$ is randomly reducible to the distributional problem $(S', X')$ and $(S', X') \in \text{tpc}\mathcal{BPP}$, then $(S, X)$ is in $\text{tpc}\mathcal{BPP}$.

3. In analogy to Exercise 10.8, prove that randomized reductions preserve solveability by probabilistic algorithms that err with probability at most $1/3$ on each input and typically run in polynomial-time.

4. Show that the error probability of such reductions can be reduced (while preserving the domination condition).

Extend the foregoing to reductions that involve distributional *search* problems.

**Exercise 10.13 (simple vs sampleable ensembles – Part 1)** Prove that any simple probability ensemble is polynomial-time sampleable.

**Guideline:** Let $\mu$ be the accumulating distribution function of $X = \{X_n\}_{n\in\mathbb{N}}$ (i.e., $\mu(x) = \Pr[X_{|x|} \leq x]$), and let $p$ be a polynomial such that (without loss of generality) $|\mu(x)| = p(|x|)$ for every $x$. Consider the algorithm that, on input $1^n$, uniformly selects $i \in [2^{p(n)}]$ and outputs $x$ if and only if $x$ is the lexicographically first string such that $i \leq \mu(x) \cdot 2^{p(n)}$. Note that this $x$ can be found by binary search, using the fact that $\mu$ is polynomial-time computable.

**Exercise 10.14 (simple vs sampleable ensembles – Part 2)** Assuming that $\#\mathcal{P}$ contains functions that are not computable in polynomial-time, prove that there exists polynomial-time sampleable ensembles that are not simple.

**Guideline:** Consider any $R \in \mathcal{PC}$ and suppose that $p$ is a polynomial such that $(x, y) \in R$ implies $|y| = p(|x|)$. Then consider the sampling algorithm $A$ that, on input $1^n$, uniformly selects $(x, y) \in \{0,1\}^{n-1} \times \{0,1\}^{p(n-1)}$ and outputs $x1$ if $(x, y) \in R$ and $x0$ otherwise. Note that $\#R(x) = 2^{p(|x|-1)} \cdot \Pr[A(1^{|x|-1}) = x1]$.

**Exercise 10.15 (distributional versions of NPC problems – Part 1 [28])** Prove that for any NP-complete problem $S$ there exists a polynomial-time sampleable ensemble $X$ such that any problem in dist$\mathcal{NP}$ is reducible to $(S, X)$. We actually assume that the many-to-one reductions establishing the NP-completeness of $S$ do not shrink the length of the input.

**Guideline:** Prove that the guaranteed reduction (of $S_{\mathtt{u}}$ to $S$) also reduces $(S_{\mathtt{u}}, U')$ to $(S, X)$, for some sampleable probability ensemble $X$. Specifically, note that $U'$ is sampleable (by Exercise 10.13) and prove that the standard reduction of $S_{\mathtt{u}}$ to $S$, when applied to a sampleable probability ensemble, induces a sampleable distribution on the instances of $S$. Consider first the case that the standard reduction is length preserving, and next extend the treatment to the general case.

**Exercise 10.16 (distributional versions of NPC problems – Part 2 [28])** Prove Theorem 10.23 (i.e., for any NP-complete problem $S$ there exists a polynomial-time sampleable ensemble $X$ such that any problem in samp$\mathcal{NP}$ is reducible to $(S, X)$). As in Exercise 10.15, we actually assume that the many-to-one reductions establishing the NP-completeness of $S$ do not shrink the length of the input.

**Guideline:** We establish the result for $S_{\mathtt{u}}$, and the rest follows as in Exercise 10.15. Thus, we focus on showing that, for a fixed sampleable $X$, we can reduce any $(S', X') \in$ samp$\mathcal{NP}$ to $(S_{\mathtt{u}}, X)$. Loosely speaking, $X$ will be an adequate convex combination of all sampleable distributions (and thus $X$ will not equal $U'$ or $U$). Specifically, $X = \{X_n\}_{n\in\mathbb{N}}$ is defined such that $X_n$ selects $i \in [n]$ with probability $\approx 1/i^2$, emulate the execution of the $i^{\text{th}}$ algorithm (in lexicographic order) on input $1^n$ for $n^3$ steps,[19] and outputs whatever the latter has output (or $0^n$ in case the said algorithm has not halted within $n^3$ steps). Prove that, for any $(S'', X'') \in$ samp$\mathcal{NP}$ such that $X''$ is sampleable in cubic time, the standard reduction of $S''$ to $S_{\mathtt{u}}$ reduces $(S'', X'')$ to $(S_{\mathtt{u}}, X)$ (as per Definition 10.15; i.e.,

[19]Needless to say, the choice to consider $n$ algorithms in the definition of $X_n$ is quite arbitrary. Any other unbounded function of $n$ that is at most a polynomial (and is computable in polynomial-time) will do. Likewise, the choice to emulate each algorithm for a cubic number of steps (rather some other polynomial number of steps) is quite arbitrary.

in particular, it satisfies the domination condition).[20] Finally, using adequate padding, reduce any $(S', X') \in \text{samp}\mathcal{NP}$ to some $(S'', X'') \in \text{samp}\mathcal{NP}$ such that $X''$ is sampleable in cubic time.

**Exercise 10.17 (search vs decision in the context of sampleable ensembles)** Prove that every problem in samp$\mathcal{NP}$ is reducible to some problem in samp$\mathcal{PC}$, and every problem in samp$\mathcal{PC}$ is *randomly* reducible to some problem in samp$\mathcal{NP}$.

**Guideline:** See proof of Theorem 10.21.

[20]Note that applying this reduction, denoted $f$, to $X''$ yields an ensembles that is also sampleable in cubic time. This uses the fact that the standard reduction runs in time that is almost linear in its output, which in turn is longer than the input.

# Epilogue

*Farewell, Hans – whether you live or end where you are! Your chances are not good. The wicked dance in which you are caught up will last a few more sinful years, and we would not wager much that you will come out whole. To be honest, we are not really bothered about leaving the question open. Adventures in the flesh and spirit, which enhanced and heightened your ordinariness, allowed you to survive in the spirit what you probably will not survive in the flesh. There were majestic moments when you saw the intimation of a dream of love rising up out of death and the carnal body. Will love someday rise up out of this world-wide festival of death, this ugly rutting fever that inflames the rainy evening sky all round?*

Thomas Mann, The Magic Mountain, The Thunderbolt.

We hope that this work succeeds in conveying the fascinating flavour of the concepts, results and open problems that dominate the field of computational complexity. We believe that the new century will witness even more exciting developments in this field, and urge the reader to try to contribute to them. But before bidding goodbye, we wish to express a few more thoughts.

As noted in Section 1.1.1, so far complexity theory has been far more successful in relating fundamental computational phenumena than in providing definite answers regarding fundamental questions. Consider, for example, the theory of NP-completeness versus the P-versus-NP Question, or the theory of pseudorandomness versus establishing the existence of one-way function (even under $\mathcal{P} \neq \mathcal{NP}$). The failure to resolve questions of the "absolute" type is the source of common frustration and one often wonders about the reasons for this failure.

Our feeling is that many of these failures are really due to the difficulty of the questions asked, and that it is the appealing and natural flavor of these questions that makes us underestimate their hardness. Indeed, the underlying assumption is that if a question is appealing and natural then answering it should not be hard. We doubt this assumption. Our feeling is that the more intuitive a question is, the harder it may be to answer. Our view is that intuitive questions arise from an encounter with the raw and chaotic reality of life, rather than from an artificial construct which is typically endowed with a rich internal structure. Indeed, natural complexity classes and natural questions regarding computation arise from looking at the reality of computation from the outside and lack any internal structure. That is, complexity classes are defined in terms of the "external behavior" of potential algorithms rather than in terms of the "internal structure" (of the problem).

# Appendix A

# Glossary of Complexity Classes

**Summary:** This glossary includes self-contained definitions of most complexity classes mentioned in the book. Needless to say, the glossary offers a very minimal discussion of these classes and the reader is referred to the main text for further discussion. The items are organized by topics rather than by alphabetic order. Specifically, the glossary is partitioned into two parts, dealing seperately with complexity classes that are defined in terms of algorithms and their resources (i.e., time and space complexity of Turing machines) and complexity classes defined in terms of non-uniform circuit (and referring to their size and depth). The algorithmic classes include time-complexity based classes (such as $\mathcal{P}$, $\mathcal{NP}$, co$\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$, co$\mathcal{RP}$, $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$) and the space complexity classes $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{RL}$ and $\mathcal{PSPACE}$. The non-uniform classes include the circuit classes $\mathcal{P}/\text{poly}$ as well as $\mathcal{NC}^k$ and $\mathcal{AC}^k$.

The definition (and basic results) regarding many other complexity classes is available at the constantly evolving *Complexity Zoo* [1].

## A.1 Introduction

Complexity classes are sets of computational problems, where each class contains problems that can be solved with specific computational resources. To define a complexity class one specifies a model of computation, a complexity measure (like time or space), which is always measured as a function of the input length, and a bound on the complexity (of problems in the class).

We follow the tradition of focusing on decision problems, but refer to these problems using the terminology of promise problems (see Section 2.4.1). That is, we will refer to the problem of distinguishing inputs in $\Pi_{\text{yes}}$ from inputs in $\Pi_{\text{no}}$, and denote the corresponding decision problem by $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$. Standard decision problems are viewed as a special case in which $\Pi_{\text{yes}} \cap \Pi_{\text{no}} = \{0,1\}^*$, and the standard formulation of complexity classes is obtained by postulating that this is the case. We refer to this case as the case of a trivial promise.

The prevailing model of computation is that of Turing machines, a model which captures the notion of (uniform) algorithms (see Section 1.2.3). Another important model is the one of non-uniform circuits (see Section 1.2.4). The term uniformity refers to whether the algorithm is the same one for every input length or whether a different "algorithm" (or rather a "circuit") is considered for each input length.

We focus on natural complexity classes, obtained by considering natural complexity measures and bounds. Typically, these classes contain natural computational problems (which are defined in Appendix G). Furthermore, almost all of these classes can be "characterized" by natural problems, which capture every problem in the class. Such problems are called complete for the class, which means that they are in the class and every problem in the class can be "easily" reduced to them, where "easily" means that the reduction takes less resources than what each of the problems seems to require individually. Efficient algorithm for a complete problem implies an algorithm of similar efficiency for *all* problems in the class.

**Organization:** The glossary is organized by topics (rather than by alphabetic order of the various items). Specifically, we partition the glossary to classes defined in terms of algorithmic resources (i.e., time and space complexity of Turing machines) and classes defined in terms of circuit (size and depth). The former include the time-complexity classes $\mathcal{P}$, $\mathcal{NP}$, co$\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$, co$\mathcal{RP}$, $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$ as well as the space-complexity classes $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{RL}$ and $\mathcal{PSPACE}$, which are all reviewed in Section A.2. The latter include the circuit classes $\mathcal{P}$/poly as well as $\mathcal{NC}^k$ and $\mathcal{AC}^k$, which are reviewed in Section A.3.

## A.2 Algorithm-based classes

The two main complexity measures considered in the context of (uniform) algorithms are the number of steps taken by the algorithm (i.e., its time complexity) and the amount of "memory" or "work-space" consumed by the computation (i.e., its space complexity). We review the time complexity based classes $\mathcal{P}$, $\mathcal{NP}$, co$\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$, co$\mathcal{RP}$, $\mathcal{ZPP}$, $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$ as well as the space complexity classes $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{RL}$ and $\mathcal{PSPACE}$.

By prepending the name of a complexity class (of decision problems) with the prefix "co" we mean the class of complement problems; that is, the problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in co$\mathcal{C}$ if and only if $(\Pi_{\text{no}}, \Pi_{\text{yes}})$ is in $\mathcal{C}$. Specifically, deciding membership in the set $S$ is in the class co$\mathcal{C}$ if and only if deciding membership in the set $\{0,1\}^* \setminus S$ is in the class $\mathcal{C}$. Thus, the definition of co$\mathcal{NP}$ and co$\mathcal{RP}$ can be easily derived from the definitions of $\mathcal{NP}$ and $\mathcal{RP}$, respectively. Complexity classes defined in terms of symmetric acceptance criteria (e.g., deterministic and two-sided error randomized classes) are trivially closed under complementation (e.g., co$\mathcal{P} = \mathcal{P}$ and co$\mathcal{BPP} = \mathcal{BPP}$) and so we don't even present their "co"-classes. In other cases (most notablly $\mathcal{NL}$ and $\mathcal{SZK}$), the closure property is highly non-trivial and we comment about it.

### A.2.1 Time complexity classes

We start with classes that are closely related to polynomial-time computations (i.e., $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$ and $\mathcal{ZPP}$), and latter consider the classes $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$.

#### A.2.1.1 Classes closely related to polynomial time

The most prominent complexity classes are $\mathcal{P}$ and $\mathcal{NP}$, which are extensively discussed in Section 2.1. We also consider classes related to randomized polynomial-time, which are discussed in Section 6.1.

**P and NP.** The class $\mathcal{P}$ consists of all decision problem that can be solved in (deterministic) polynomial-time. A decision problem $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$ is in $\mathcal{NP}$ if there exists a polynomial $p$ and a (deterministic) polynomial-time algorithm $V$ such that the following two conditions hold

1. For every $x \in \Pi_{\rm yes}$ there exists $y \in \{0,1\}^{p(|x|)}$ such that $V(x,y) = 1$.
2. For every $x \in \Pi_{\rm no}$ and every $y \in \{0,1\}^*$ it holds that $V(x,y) = 0$.

A string $y$ satisfying Condition 1 is called an NP-witness (for $x$).

**Reductions and NP-completeness (NPC).** A problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and every problem in $\mathcal{NP}$ is polynomial-time reducible to it, where polynomial-time reducibility is defined and discussed in Section 2.2. Loosely speaking, a polynomial-time reduction of problem $\Pi$ to problem $\Pi'$ is a polynomial-time algorithm that solves $\Pi$ by making queries to a subroutine that solves problem $\Pi'$, where the running-time of the subroutine is not counted in the algorithm's time complexity. Typically, NP-completeness is defined while restricting the reduction to make a single query and output its answer. Such a reduction, called a Karp-reduction, is represented by a polynomial-time computible mapping that maps yes-instances of $\Pi$ to yes-instances of $\Pi'$ (and no-instances of $\Pi$ to no-instances of $\Pi'$).

**Probabilistic polynomial-time (BPP, RP and ZPP).** A decision problem $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$ is in $\mathcal{BPP}$ if there exists a probabilistic polynomial-time algorithm $A$ such that the following two conditions hold

1. For every $x \in \Pi_{\rm yes}$ it holds that $\mathsf{Pr}[A(x)=1] \geq 2/3$.
2. For every $x \in \Pi_{\rm no}$ it holds that $\mathsf{Pr}[A(x)=0] \geq 2/3$.

That is, the algorithm has two-sided error probability (of $1/3$), which can be further reduced by repetitions. We stress that due to the two-sided error probability of $\mathcal{BPP}$, it is not known whether or not $\mathcal{BPP}$ is contained in $\mathcal{NP}$. In contrast to $\mathcal{BPP}$, there are one-sided and zero-error classes, denoted $\mathcal{RP}$ and $\mathcal{ZPP}$, respectively. A problem $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$ is in $\mathcal{RP}$ if there exists a probabilistic polynomial-time algorithm $A$ such that the following two conditions hold

1. For every $x \in \Pi_{\rm yes}$ it holds that $\mathsf{Pr}[A(x)=1] \geq 1/2$.
2. For every $x \in \Pi_{\rm no}$ it holds that $\mathsf{Pr}[A(x)=0] = 1$.

Again, the error probability can be reduced by repetitions, and thus $\mathcal{RP} \subseteq \mathcal{BPP} \cap \mathcal{NP}$. A problem $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$ is in $\mathcal{ZPP}$ if there exists a probabilistic polynomial-time algorithm $A$, which may output a special ("don't know") symbol $\perp$, such that the following two conditions hold

1. For every $x \in \Pi_{\rm yes}$ it holds that $\mathsf{Pr}[A(x) \in \{1, \perp\}] = 1$ and $\mathsf{Pr}[A(x)=1] \geq 1/2$.
2. For every $x \in \Pi_{\rm no}$ it holds that $\mathsf{Pr}[A(x) \in \{0, \perp\}] = 1$ and $\mathsf{Pr}[A(x)=0] \geq 1/2$.

Note that $\mathcal{P} \subseteq \mathcal{ZPP} = \mathcal{RP} \cap {\rm co}\mathcal{RP}$. When defined in terms of promise problems, these classes have complete problems (w.r.t Karp-reductions), but the same is not known when considering only standard decision problems (with trivial promise).

**The counting class $\#\mathcal{P}$.** Functions in $\#\mathcal{P}$ count the number of solutions to an NP-type search problem (or, equivalently, the number of NP-witnesses for a yes-instance of a decision problem in $\mathcal{NP}$). Formally, a function $f$ is in $\#\mathcal{P}$ if there exists a polynomial $p$ and a (deterministic) polynomial-time algorithm $V$ such that $f(x) = |\{y \in \{0,1\}^{p(|x|)} : V(x,y)=1\}|$. Indeed, $p$ and $V$ are as in the definition of $\mathcal{NP}$, and it follows that deciding membership in the set $\{x : f(x) \geq 1\}$ is in $\mathcal{NP}$. Clearly, $\#\mathcal{P}$ problems are solvable in polynomial space. Surprisingly, the permanent of positive integer matrices is $\#\mathcal{P}$-complete (i.e., it is in $\#\mathcal{P}$ and any function in $\#\mathcal{P}$ is polynomial-time reducible to it).

#### A.2.1.2 Other time complexity classes

The classes $\mathcal{E}$ and $\mathcal{EXP}$ corresponding to problems that can be solved (by a deterministic algorithm) in time $2^{O(n)}$ and $2^{{\rm poly}(n)}$, respectively, for $n$-bit long inputs. Clearly, $\mathcal{NP} \subseteq \mathcal{EXP}$.

In general, one may define a complexity class for every time bound and every type of machine (i.e., deterministic, probabilistic and non-deterministic), but polynomial and exponential bounds seem most natural and very robust. Another robust type of time bounds that is sometimes used is quasi-polynomial time (e.g., $\widetilde{\mathcal{P}}$ denotes the class of problems solvable by deterministic machines of time complexity $\exp({\rm poly}(\log n))$). Another class that occurs is $\mathcal{NEXP}$ the class of problems that can be solved by a non-deterministic machine in $2^{{\rm poly}(n)}$ steps.[1]

[1] Alternatively, analogously to the definition of $\mathcal{NP}$, a problem $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$ is in $\mathcal{NEXP}$

**The Polynomial-time hierarchy, $\mathcal{PH}$.** For any natural number $k$, the $k^{\text{th}}$ level of the polynomial-time hierarchy consists of problems $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ such that there a polynomial $p$ and a polynomial-time algorithm $V$ that satisfies the following two conditions:

1. For every $x \in \Pi_{\text{yes}}$ there exists $y_1 \in \{0,1\}^{p(|x|)}$ such that for every $y_2 \in \{0,1\}^{p(|x|)}$ there exists $y_3 \in \{0,1\}^{p(|x|)}$ such that for every $y_4 \in \{0,1\}^{p(|x|)}$ ... (going for $k$ quantifier alternations) ... it holds that $V(x, y_1, y_2, y_3, y_4, ..., y_k) = 1$.

2. For every $x \in \Pi_{\text{no}}$, the above condition does not hold. That is, for every $y_1 \in \{0,1\}^{p(|x|)}$ there exists $y_2 \in \{0,1\}^{p(|x|)}$ such that for every $y_3 \in \{0,1\}^{p(|x|)}$ there exists $y_4 \in \{0,1\}^{p(|x|)}$ ... it holds that $V(x, y_1, y_2, y_3, y_4, ..., y_k) = 0$.

Such a problem $\Pi$ is said to be in $\Sigma_k$ (and $\Pi_k \stackrel{\text{def}}{=} \text{co}\Sigma_k$). Indeed, $\mathcal{NP} = \Sigma_1$ corresponds to the special case where $k = 1$. Interestingly, $\mathcal{PH}$ is polynomial-time reducible to $\#\mathcal{P}$.

## A.2.2 Space complexity

When defining space-complexity classes, one counts *only* the space consumed by the actual computation, and *not* the space occupied by the input and output. This is formalized by postulating that the input is read from a read-only device (resp., the output is written on a write-only device). Four important classes of decision problems are:

- The class $\mathcal{L}$ consists of problems solvable in logarithmic space. That is, a problem $\Pi$ is in $\mathcal{L}$ if there exists a standard (i.e., deterministic) algorithm of logarithmic space-complexity for solving $\Pi$. This class contains some simple computational problems (e.g., matrix multiplication), and arguably captures the most space-efficient computations.

- Classes of problems solvable by randomized algorithms of logarithmic space-complexity include $\mathcal{RL}$ and $\mathcal{BPL}$, which are defined analogously to $\mathcal{RP}$ and $\mathcal{BPP}$. That is, $\mathcal{RL}$ corresponds to algorithms with one-sided error probability, whereas $\mathcal{BPL}$ allows two-sided error.

- The class $\mathcal{NL}$ is the non-deterministic analogue of $\mathcal{L}$, and is traditionally defined in terms of non-deterministic machines of logarithmic space-complexity.[2] The class $\mathcal{NL}$ contains the problem of deciding whether there exists a directed

if there exists a polynomial $p$ and a polynomial-time algorithm $V$ such that the two conditions hold

1. For every $x \in \Pi_{\text{yes}}$ there exists $y \in \{0,1\}^{2^{p(|x|)}}$ such that $V(x, y) = 1$.
2. For every $x \in \Pi_{\text{no}}$ and every $y \in \{0,1\}^*$ it holds that $V(x, y) = 0$.

[2]See further discussion of this definition in Section 5.3. In particular, note that we refrained here from presenting a definition analogous to the definition of $\mathcal{NP}$.

path between two given vertexes in a given directed graph. In fact, the latter problem is complete for the class (under logarithmic-space reductions). Interestingly, co$\mathcal{NL}$ equals $\mathcal{NL}$.

- The class $\mathcal{PSPACE}$ consists of problems solvable in polynomial space. This class contains very difficult problems, including the computation of winning strategies for any "efficient 2-party games" (see Section 5.4).

Clearly, $\mathcal{L} \subseteq \mathcal{RL} \subseteq \mathcal{NL} \subseteq \mathcal{P}$ and $\mathcal{NP} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXP}$.

## A.3 Circuit-based classes

We refer the reader to Section 1.2.4 for a definition of Boolean circuits as computing devices. The two main complexity measures considered in the context of (non-uniform) circuits are the number of gates (or wires) in the circuit (i.e., the circuit's size) and the length of the longest directed path from an input to an output (i.e., the circuit's depth).

Throughout this section, when we talk of circuits, we actually refer to families of circuits conatining a circuit for each instance length, where the $n$-bit long instances of the computational problem are handled by the $n^{\text{th}}$ circuit in the family. Similarly, the size and depth of the circuit actually refers to the family of circuits and to the dependence (on $n$) of the size and depth of the $n^{\text{th}}$ circuit in the family.

**General polynomial-size circuits (P/poly).** The main motivation for the introduction of complexity classes based on (non-uniform) circuits is the development of lower-bounds. For example, the class of problems solvable by polynomial-size circuits, denoted $\mathcal{P}$/poly, is a super-set of $\mathcal{P}$ (because it clearly contains $\mathcal{P}$ as well as deciding membership in any subset of $\{1\}^*$, whereas there exists such sets that represents decision problems that are not solvable (i.e., by any uniform algorithm)). Thus, showing that $\mathcal{NP}$ is not contained in $\mathcal{P}$/poly would imply $\mathcal{P} \neq \mathcal{NP}$. For further discussion see Section B.2.

**The subclasses AC0 and TC0.** The class $\mathcal{AC}^0$, discussed in Section B.2, consists of problems solvable by constant-depth polynomial-size circuits of *unbounded fan-in*. The analogue class that allows also (unbounded fan-in) majority-gates (or, equivalently, threshold-gates) is denoted $\mathcal{TC}^0$.

**The subclasses AC and NC.** Turning back to the standard basis (of $\neg$, $\vee$ and $\wedge$), for any non-negative integer $k$, we denote by $\mathcal{NC}^k$ (resp., $\mathcal{AC}^k$) the class of problems solvable by polynomial-size circuits of *bounded fan-in* (resp., unbounded fan-in) having depth $O(\log^k n)$, where $n$ is the input length. Clearly, $\mathcal{NC}^k \subseteq \mathcal{AC}^k \subseteq \mathcal{NC}^{k+1}$. A commonly referred class is $\mathcal{NC} \stackrel{\text{def}}{=} \cup_{k\in\mathbb{N}}\mathcal{NC}^k$.

We mention that the class $\mathcal{NC}^2 \supseteq \mathcal{NL}$ is the habitat of most natural computational problems of Linear Algebra: solving a linear system of equations as well as computing the rank, inverse and determinant of a matrix. The class $\mathcal{NC}^1$

contains all symmetric functions, regular languages as well as word problems for finite groups and monoids. The class $\mathcal{AC}^0$ contains all properties of finite objects expressible by first-order logic.

**Uniformity.** The above classes make no reference to the complexity of constructing the adequate circuits, and it is plausible that there is no effective way of constructing these circuits (e.g., as in case of circuits that trivially solve undecidable problem regarding unary instances). A minimal notion of constructability of such (polynomial-size) circuits is the existance of a polynomial time algorithm that given $1^n$ produces the $n^{\text{th}}$ relevant circuit (i.e., the circuit that solves the problem on instances of length $n$). Such a notion of constructability means that the family of circuits is "uniform" in some sense (rather than consisting of circuits that have no relation between one another). Stronger notions of uniformity (e.g., log-space constructability) are more adequate for subclasses such as AC and NC. We mention that log-space uniform NC circuits correspond to parallel algorithms that use polynomially many processors and run in polylogarithmic time.

# Appendix B

# On the Quest for Lower Bounds

*Alas, Philosophy, Medicine, Law, and unfortunately also Theology, have I studied in detail, and still remained a fool, not a bit wiser than before. Magister and even Doctor am I called, and for a decade am I sick and tired of pulling my pupils by the nose and understanding that we can know nothing.*[1]

J.W. Goethe, Faust, Lines 354–364

**Summary:** In this appendix we survey some attempts at proving lower bounds on the complexity of natural computational problems. In the first part, devoted to Circuit Complexity, we describe lower bounds for the *size* of (restricted) circuits that solve natural computational problems. This can be viewed as a program whose long-term goal is proving that $\mathcal{P} \neq \mathcal{NP}$. In the second part, devoated to Proof Complexity, we describe lower bounds on the length of (restricted) propositional proofs of natural tautologies. This can be viewed as a program whose long-term goal is proving that $\mathcal{NP} \neq \text{co}\mathcal{NP}$.

The current activity in these areas is aimed towards developing proof techniques that may be applied to the resolution of the "big problems" (such as P versus NP), but the current achievements (though very impressive) seem far from reaching this goal. Current crown-jewel achievements in these areas take the form of tight (or strong) lower bounds on the complexity of computing (resp., proving) "relatively simple" functions (resp., claims) in *restricted* models of computation (resp., proof systems).

[1]This quote reflects a common sentiment, not shared by the author of the current book.

## B.1 Preliminaries

Circuit complexity refers to a non-uniform model of computation; specifically the model of Boolean circuits, focusing on the size of such circuits, while ignoring the complexity of constructing adequate circuits. Similarly, proof complexity refers to proofs of tautologies, focusing on the length of such proofs, while ignoring the complexity of generating such proofs. Both circuits and proofs are finite objects that are defined on top of the notion of a *directed acyclic graph* (dag), which we review next.

A dag $G(V, E)$ consists of a finite set of vertices $V$, and a set of ordered pairs called directed edges $E \subseteq V \times V$, in which there are no directed cycles. The vertices with no incoming edges are called the inputs of the dag $G$, and the vertices with no outgoing edges are called the outputs. We will restrict ourselves to dags in which the number of *incoming* edges to every vertex is at most 2. If the number of *outgoing* edges from every node is at most 1, the dag is called a tree. Finally, we assume that every vertex can be reached from some input via a directed path. The size of a dag will be its number of edges.

To make a dag into a computational device (or a proof), each non-input vertex will be marked by a rule, converting values in its predecessors to values at that vertex. It is easy to see that the vertices of every dag can be linearly ordered, such that predecessors of every vertex (if any) appear before it in the ordering. Thus, if the input vertices are labeled with some values, we can label the remaining vertices (in that order), one at a time, till all vertices (and in particular all outputs) are labeled.

For computation devices, the non-input vertices will be marked by functions (called gates), which make the dag a circuit. If we label the input vertices by specific values from some domain, the outputs will be determined by them, and the circuit will naturally define a function (from input values to output values). For more details see Section 1.2.4.

For proofs, the non-input vertices will be marked by sound deduction (or inference) rules, which make the dag a proof. If we label the inputs by formulae that are axioms in a given proof system, the output again will be determined by them, and will yield the tautology proved by this proof.

We note that both settings fit the paradigm of simplicity shared by computational models discussed in Section 1.2; the rules are simple by definition – they are applied to at most 2 previous values. The main difference is that this model is finite – each dag can compute only functions/proofs with a fixed input length. To allow all input lengths, one must consider families of dags, one for each, thus significantly extending the power of the computation model beyond that of the notion of *algorithm* (discussed in Section 1.2.3). However, as we are interested in lower bounds here, this is legitimate, and one can hope that the finiteness of the model will potentially allow for combinatorial techniques to analyze its power and limitations. Furthermore, these models allow for the introduction (and study) of meaningful restricted classes of computations.

## B.2 Boolean Circuit Complexity

In Boolean circuits all inputs, outputs, and values at intermediate nodes of the dag are bits. The set of allowed gates is naturally taken to be a *complete basis* – one that allows the circuit to compute *all* Boolean functions. The specific choice of a complete basis hardly effects the study of circuit complexity. A typical choice is the set $\{\wedge, \vee, \neg\}$ of (respectively) conjunction, disjunction (each on 2 bits) and negation (on 1 bit).

For a finite function $f$, we denote by $\mathcal{S}(f)$ the size of the smallest Boolean circuit computing $f$. We will be interested in sequences of functions $\{f_n\}$, where $f_n$ is a function on $n$ input bits, and will study (their size complexity) $\mathcal{S}(f_n)$ asymptotically as a function of $n$. With some abuse of notation, for $f(x) \stackrel{\text{def}}{=} f_{|x|}(x)$, we let $\mathcal{S}(f)$ denote the integer function that assigns to $n$ the value $\mathcal{S}(f_n)$. Thus, we refer to the following definition.

**Definition B.1** (circuit complexity): *Let* $f : \{0,1\}^* \to \{0,1\}^*$ *and* $\{f_n\}$ *be such that* $f(x) = f_{|x|}(x)$ *for every* $x$. *The* complexity *of a function* $f$ (resp., $\{f_n\}$), *denoted* $\mathcal{S}(f)$ (resp., $n \mapsto \mathcal{S}(f_n)$), *is a function of* $n$ *that represents the size of the smallest Boolean circuit computing* $f_n$.

We note that different circuits (in particular having a different number of inputs) are used for each $f_n$. Still there may be a simple description of this sequence of circuits, say, an algorithm that on input $n$ produces a circuit computing $f_n$. In case such an algorithm exists and works in time polynomial in the size of its output, we say that the corresponding sequence of circuits is uniform. Note that if $f$ has a uniform sequence of polynomial-size circuits then $f \in \mathcal{P}$. On the other hand, it can be shown that any $f \in \mathcal{P}$ has (a uniform sequence of) polynomial-size circuits. Consequently, a super-polynomial size lower-bound on any function in $\mathcal{NP}$ would imply that $\mathcal{P} \neq \mathcal{NP}$.

Definition B.1 makes no reference to "uniformity" and indeed the sequence of smallest circuits computing $\{f_n\}$ may be highly "nonuniform". Indeed, non-uniformity makes the circuit model stronger than Turing machines (or, equivalently, than the model of uniform circuits): there exist functions $f$ that cannot be computed by Turing machines (regardless of their running time), but do have linear-size circuits. So isn't proving circuit lower bounds a much harder task than we need to resolve the P vs. NP question?

The answer is that there is a strong sentiment that the extra power provided by non-uniformity is irrelevant to the P vs. NP question; that is, it is conjectured that NP-complete sets do not have polynomial-size circuits. This conjecture is supported by the fact that its failure will yield an unexpected collapse in the complexity world of standard computations (see Section 3.2). Furthermore, the hope is that abstracting away the (supposedly irrelevant) uniformity condition will allow for combinatorial techniques to analyze the power and limitations of polynomial-size circuits (w.r.t NP-sets). This hope has materialized in the study of restricted classes of circuits (see Sections B.2.2 and B.2.3).

We also mention that Boolean circuits are a natural computational model, corresponding to "hardware complexity" (which was indeed the original motivation for their introduction by Shannon [188]), and so their study is of independent interest. Moreover, some of the techniques for analyzing Boolean functions found applications elsewhere (e.g., in computational learning theory, combinatorics and game theory).

### B.2.1 Basic Results and Questions

We have already mentioned several basic facts about Boolean circuits, in particular the fact that they can efficiently simulate Turing Machines. Another basic fact is that *most Boolean functions require exponential size circuits*, which is due to the gap between the number of functions and the number of small circuits.

Thus, hard functions (i.e., function that require large circuits and thus have no efficient algorithms) do exist, to say the least. However, the aforementioned hardness result is proved via a counting argument, and so provides no way of pointing to one hard function. Using more conventional language, we cannot prove analogous hardness results for any *explicit* function $f$ (e.g., for an NP-complete function like SAT or even for functions in $\mathcal{EXP}$). The situation is even worse: no *nontrivial* lower-bound is known for any explicit function. Note that for any function $f$ on $n$ bits (which depends on all its inputs), we trivially must have $\mathcal{S}(f) \geq n$, just to read the inputs. The main open problem of circuit complexity is beating this trivial bound.

**Open Problem B.2** *Find an explicit Boolean function $f$ (or even a length-preserving function $f$) for which $\mathcal{S}(f)$ is not $O(n)$.*

A particularly basic special case of this problem, is the question whether addition is easier to perform than multiplication. Let $\mathtt{ADD}:\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ and $\mathtt{MULT}:\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^{2n}$, denote the addition and multiplication functions, respectively, applied to a pair of integers (presented in binary). For addition we have an optimal upper bound; that is, $\mathcal{S}(\mathtt{ADD}) = O(n)$. For multiplication, the standard (elementary school) quadratic-time algorithm can be greatly improved (via Discrete Fourier Transforms) to slightly super-linear, yielding $\mathcal{S}(\mathtt{MULT}) = O(n \cdot (\log n)^2)$. Now, the question is *whether or not there exist linear-size circuits for multiplication* (i.e., is $\mathcal{S}(\mathtt{MULT}) = O(n)$)?

Unable to report on any nontrivial lower bound (for an explicit function), we turn to restricted models. There has been some remarkable successes in developing techniques for proving strong lower bounds for natural restricted classes of circuits. We describe the most important ones, and refer the reader to [43, 218] for further detail.

General Boolean circuits, as described above, can compute every function and can do it at least as efficiently as general (uniform) algorithms. Restricted circuits may be only able to compute a subclass of all functions (e.g., monotone functions). The restriction makes sense when the related classes of functions and the computations represented by the restricted circuits are natural (from a conceptual or practical viewpoint). The models discussed below satisfy this condition.

### B.2.2 Monotone Circuits

An extremely natural restriction comes by forbidding negation from the set of gates, namely allowing only $\{\wedge, \vee\}$. The resulting circuits are called monotone circuits and it is easy to see that they can compute every function $f : \{0,1\}^n \to \{0,1\}$ that is monotone with respect to the standard partial order on $n$-bit strings ($x \preceq y$ iff for every bit position $i$ we have $x_i \leq y_i$).

It is as easy to see that most monotone functions require exponential size monotone circuits. Still, proving a super-polynomial lower bound on an explicit monotone function was open for over 40 years, till the invention of the so-called *approximation method* (by Razborov [174]).

Let CLIQUE be the function that, given a graph on $n$ vertices (by its adjacency matrix), outputs 1 iff it contains a complete subgraph of size (say) $\sqrt{n}$ (namely, all pairs of vertices in some $\sqrt{n}$ subset are connected by edges). This function is clearly monotone. Moreover, it is known to be NP-complete.

**Theorem B.3** ([174], improved in [7]): *There are no polynomial-size monotone circuits for* CLIQUE.

We note that the lower bounds are sub-exponential in the number of vertices (i.e., size $\exp(\Omega(n^{1/8}))$ for $n$ vertices), and that similar lower-bounds are known for functions in $\mathcal{P}$. Thus, there exists an *exponential separation of the monotone and non-monotone circuit complexities* (referring, of course, to the computation of monotone functions).

### B.2.3 Bounded-Depth Circuits

The next restriction is structural: *we allow all gates, but limit the depth of the circuit.* The depth of a dag is simply the length of the longest directed path in it. So in a sense, depth captures the *parallel time* to compute the function: if a circuit has depth $d$, then the function can be evaluated by enough processors in $d$ phases (where in each phase many gates are evaluated at once). Parallel time is another important computational resource.

We will restrict $d$ to be a constant, which still is interesting not only as parallel time, but also due to the relation of this model to expressibility in first order logic as well as to complexity classes above NP called the *Polynomial-time Hierarchy* (see Section 3.2). In the current setting (of constant-depth circuits), we allow *unbounded fan-in* (i.e., $\wedge$-gates and $\vee$-gates taking any number of incoming edges), as otherwise each output bit can depend only on a constant number of input bits.

Let PAR (for parity) denote the sum modulo two of the input bits, and MAJ (for majority) be 1 iff there are more 1's than 0's among the input bits. The invention of the *random restriction method* (by Furst, Saxe, and Sipser [79]) led to the following basic result.

**Theorem B.4** ([79], improved in [222, 110]): *For all constant $d$, the functions* PAR *and* MAJ *have no polynomial size circuit of depth $d$.*

The aforementioned improvement (of Håstad [110], following Yao [110]) gives a relatively tight lower bound of $\exp(\Omega(n^{1/(d-1)}))$ on the size of $n$-input `PAR` circuits of depth $d$.

Interestingly, `MAJ` remains hard (for constant-depth polynomial-size circuits) even if the circuits are also allowed (unbounded fan-in) `PAR`-gates (this result is based on yet another proof technique: *approximation by polynomials* [195, 175]). However, the "converse" does not hold (i.e., constant-depth polynomial-size circuits with `MAJ`-gates can compute `PAR`), and in general the class of constant-depth polynomial-size circuits with `MAJ`-gates (denoted $\mathcal{TC}^0$) seems quite powerful.

### B.2.4 Formula Size

The final restriction is again structural – we require the dag to be a tree. Intuitively, this forbids the computation from reusing a previously computed partial result (and if it is needed again, it has to be recomputed). The resulting circuits are simply formulae, which are natural not only for their prevalent mathematical use, but also since their size can be related to the *memory* requirements of a Turing machine. (Here we go back to the standard basis of negation ($\neg$), and $\wedge, \vee$ gates *of fan-in 2*.)

One of the oldest results on Circuit Complexity, is that `PAR` and `MAJ` are non-trivial in this model. The proof follows a simple combinatorial (or information theoretic) argument.

**Theorem B.5** [134]: *Boolean formuale for $n$-bit* `PAR` *and* `MAJ` *require size $\Omega(n^2)$ size.*

This should be contrasted with the linear-size circuits that exist for both functions. We comment that $\mathcal{S}(\texttt{PAR}) = O(n)$ is trivial, but $\mathcal{S}(\texttt{MAJ}) = O(n)$ is not. Encouraged by Theorem B.5, one may ask whether we can hope to provide super-polynomial lower bounds on the formula size of explicit functions. This is indeed a famous open problem.

**Open Problem B.6** *Find an explicit Boolean function $f$ for which $\mathcal{S}(f)$ is super-polynomial.*

One of the cleanest methods suggested is the *communication complexity method* (of Karchmer and Wigderson [131]). This method asserts that the depth of a formula for a Boolean function $f$ equals the communication complexity in the following two party game, $G_f$. The first party is given $x \in f^{-1}(1) \cap \{0,1\}^n$, the second party is given $y \in f^{-1}(0) \cap \{0,1\}^n$, and their goal is to find a bit location on which $x$ and $y$ disagree (i.e., $i$ such that $x_i \neq y_i$, which cleraly exists). To that end, the party exchange messages, according to a predetermined protocol, and the question is what is the communication complexity (in terms of total number of bits exchanged on the worst-case input pair) of the best such protocol.

Note that proving a super-logarithmic lower-bound of the communication complexity of the aforementioned game $G_f$ will establish a super-polynomial lower-bound on the size of formulae computing $f$ (becuase formula depth can be made logarithmic in their size). We stress that a lower bound of purely information theoretic nature (no computational restriction were paced on the parties in the game) implies a computational lower bound!

We mention that the communication complexity method has a *monotone version* in which the depth of monotone circuits is related to the communication complexity of protocols that are required to find an $i$ such that $1 = x_i \neq y_i = 0$ (rather than any $i$ such that $x_i \neq y_i$).[2] In fact, the montone version is better known than the general one, due to its success in establishing linear lower bounds on monotone depth of natural problems such as perfect matching (by Raz and Wigderson [173]).

## B.3 Arithmetic Circuits

We now leave the Boolean rind, and discuss circuits over general fields. Fix any field $F$. The gates of the dag will now be the standard $+$ and $\times$ operations in the field. This requires two immediate clarifications. First, to allow using constants of the field, one adds a special input vertex whose value is the constant '1' of the field. Moreover, multiplication by any field element (e.g., $-1$) is free. Second, one may wonder about division. However, we will be mainly interested in computing polynomials, and for computing polynomials (over infinite fields) division can be efficiently emulated by the other operations.

Now the inputs of the dag will hold elements of the field $F$, and hence so will all computed values at vertices. Thus an arithmetic circuit computes a polynomial map $p : F^n \to F^m$, and every such polynomial map is computed by some circuit. We denote by $\mathcal{S}_F(p)$ the size of a smallest circuit computing $p$ (when no subscript is given, $F = \mathcal{Q}$ the field of rational numbers). As usual, we'll be interested in sequences of polynomials, one for every input size, and will study size asymptotically.

It is easy to see that over any *fixed* finite field, arithmetic circuits can simulate Boolean circuits on Boolean inputs with only constant factor loss in size. Thus the study of arithmetic circuits focuses more on infinite fields, where lower bounds may be easier to obtain.

As in the Boolean case, the existence of hard functions is easy to establish (via dimension considerations, rather than counting argument), and we will be interested in *explicit* (families of) polynomials. However, the notion of explicitness is more delicate here (e.g., allowing polynomials with algebraically independent coefficients would yield strong lower bounds, which are of no interest whatsoever). Very roughly speaking, polynomials are called explicit if the mapping from monomials to (a finite description of) their coefficients has an efficient program.

An important parameter, which is absent in the Boolean model, is the *degree* of the polynomial(s) computed. It is obvious, for example, that a degree $d$ polynomial (even in one variable, i.e., $n = 1$) requires size at least $\log d$. We briefly consider the univariate case (in which $d$ is the only measure of input size), which already contains

[2] Note that since $f$ is monotone, $f(x) = 1$ and $f(y) = 0$ implies the existence of an $i$ such that $1 = x_i \neq y_i = 0$.

striking and important problems. Then we move to the general multivariate case, in which as usual $n$, the number of inputs will be the main parameter. (We refer the reader to [201] for further detail.)

### B.3.1 Univariate Polynomials

How tight is the $\log d$ lower bounds for the size of an arithmetic circuit computing a degree $d$ polynomial? A simple dimension argument shows that for most degree $d$ polynomials $p$, $\mathcal{S}(p) = \Omega(d)$. However, we know of no explicit one:

**Open Problem B.7** *Find an explicit polynomial $p$ of degree $d$, such that $\mathcal{S}(p)$ is not $O(\log d)$.*

To illustrate the question, we consider the following two concrete polynomials $p_d(x) = x^d$, and $q_d(x) = (x+1)(x+2)\cdots(x+d)$. Clearly, $\mathcal{S}(p_d) \leq 2\log d$ (via repeated squaring), so the trivial lower bound is tight. On the other hand, it is a major open problem to determine $\mathcal{S}(q_d)$, and the conjecture is that $\mathcal{S}(q_d)$ is not polynomial in $\log d$. To realize the importance of this question, we state the following proposition:

**Proposition B.8** *If given $d$ one can find in polynomial-time an arithmetic circuit computing $q_d$ then the integer factorization problem can be solved in polynomial-time.*

Recall that it is widely believed that the integer factorization problem is intractable (i.e., cannot be solved in polynomial-time). Proposition B.8 follows by observing that $q_d(0) \equiv (d!) \pmod{N}$ for any integer $N$. Thus, the proposition's hypothesis provides a polynomial time algorithm for computing $(d!) \bmod N$, when given $d$ and $N$. Next, observe that $\gcd(N, (d!) \bmod N)$ yields a non-trivial factor of $N$ if some prime factors of $N$ are smaller than $d$ and some are larger than $d$. Thus, using a binary search for a suitable $d$, we can obtain such a non-trivial factor.

### B.3.2 Multivariate Polynomials

We are now back to polynomials with $n$ variables. To make $n$ our only input size parameter, it is convenient to restrict ourselves to polynomials whose total degree is at most $n$.

Once again, almost every polynomial $p$ in $n$ variables requires size $\mathcal{S}(p) \geq \exp(n/2)$, via a dimension argument, and we seek explicit polynomial (families) that are hard. Unlike in the Boolean world, here there are slightly nontrivial lower bounds (via elementary tools from algebraic geometry).

**Theorem B.9** [23]: $\mathcal{S}(x_1^n + x_2^n + \cdots + x_n^n) = \Omega(n \log n)$.

The same techniques extend to prove a similar lower-bound for other natural polynomials such as the symmetric polynomials and the determinant. Establishing a stronger lower-bound for any explicit polynomial is a major open problem. Another open problem is obtaining a super-linear lower bound for a polynomial map of constant (even 1) total degree. Outstanding candidates for the latter open problem are the *linear* maps computing the Discrete Fourier Transform over the Complex numbers, or the Walsh transform over the Rationals (for both $O(n \log n)$ algorithms are known, but no super-linear lower bounds are known).

We now focus on specific polynomials of central importance. The most natural and well studied candidate for the last open problem is the matrix multiplication function $\mathtt{MM}$: let $A, B$ be two $m \times m$ matrices of variables over $F$, and define $\mathtt{MM}(A, B)$ to be the $n = m^2$ entries of the matrix $A \times B$. Thus, $\mathtt{MM}$ is a set of $n$ explicit bilinear forms over the $2n$ input variables. It is known that $\mathcal{S}_{\mathrm{GF}(2)}(\mathtt{MM}) \geq 3n$ (cf., [192]). On the other hand, the obvious $m^3 = n^{3/2}$ algorithm can be improved.

**Theorem B.10** [59]: *For every field $F$, $\mathcal{S}_F(\mathtt{MM}) = O(n^{1.19})$.*

So what is the complexity of $\mathtt{MM}$ (even if one counts only multiplication gates)? Is it linear or almost-linear or is it the case that $\mathcal{S}(\mathtt{MM}) > n^\alpha$ for some $\alpha > 1$? This is indeed a famous open problem.

We next consider the determinant and permanent polynomials ($\mathtt{DET}$ and $\mathtt{PER}$, resp.) over the $n = m^2$ variables representing an $m \times m$ matrix. While $\mathtt{DET}$ plays a major role in classical mathematics, $\mathtt{PER}$ is somewhat esoteric (though it appears in Statistical Mechanics and Quantum Mechanics). In the context of complexity theory both polynomials are of great importance, because they capture natural complexity classes. The function $\mathtt{DET}$ has relatively low complexity (and is closely related to the class of polynomials having polynomial-sized arithmetic formulae), whereas $\mathtt{PER}$ seems to have high complexity (and it is complete for the counting class $\#\mathcal{P}$ (see §6.2.1), which contains $\mathcal{NP}$). Thus, it is conjectured that $\mathtt{PER}$ is *not* polynomial-time reducible to $\mathtt{DET}$. A specific type of reduction that makes sense in this algebraic context is a reduction by projection.

**Definition B.11** (projections): *Let $X$ and $Y$ be two disjoint finite sets of variables, and let $p$ and $q$ be polynomial maps over $F[X]$ and $F[Y]$, respectively. We say that there is a* projection from $p$ to $q$ over $F$*, denoted $p \propto q$ if there exists a function $h : X \to Y \cup F$ such that $p(\bar{x}) \equiv q(h(\bar{x}))$.*

Clearly, if $p \propto q$ then $\mathcal{S}_F(p) \leq \mathcal{S}_F(q)$. Let $\mathtt{DET}_m$ and $\mathtt{PER}_m$ denote the functions $\mathtt{DET}$ and $\mathtt{PER}$ restricted to $m$-by-$m$ matrices. It is known that $\mathtt{PER}_m \propto \mathtt{DET}_{3^m}$, but to yield a polynomial-time reduction one would need a projection of $\mathtt{PER}_m$ to $\mathtt{DET}_{\mathrm{poly}(m)}$. It is conjectured that no such projection exists.

## B.4 Proof Complexity

The concept of *proof* is what distinguishes the study of Mathematics from all other fields of human inquiry. Mathematicians have gathered millennia of experience to attribute such adjectives to proofs as "insightful, original, deep" and most notably, "difficult". Can one quantify, mathematically, the difficulty of proving various theorems? This is exactly the task undertaken in Proof Complexity. It seeks to classify theorems according to the difficulty of proving them, much like Circuit Complexity seeks to classify functions according to the difficulty of computing them. In proofs, just like in computation, there will be a number of models, called *proof systems* capturing the power of reasoning allowed to the prover.

We will consider only propositional proof systems, and so our theorems will be *tautologies*. We will see soon why the complexity of proving tautologies is highly nontrivial and amply motivated.

The formal definition of a proof system spells out what we take for granted: the efficiency of the verification procedure. In the following definition the efficiency of the verification procedure refers to its running-time measured in terms of the *total length of the alleged theorem and proof*. In contrast, in Chapter 9, we consider the running-time as a function of the *length of the alleged theorem*. (Both approaches were mentioned in Section 2.1, where the two approaches coincide because in Section 2.1 we mandated proofs of length polynomial in the alleged theorem.)

**Definition B.12** [58]: *A* (propositional) proof system *is a polynomial-time Turing machine $M$ such that a formula $T$ is a tautology of and only if exists a string $\pi$, called a* proof*, such that $M(\pi,T)=1$.*

In agreement with standard formalisms (see below), the proof is viewed as coming before the theorem. Note that Definition B.12 guarantees the completeness and soundness of the proof system, as well as verification efficiency (relative to the total length of the alleged proof–theorem pair). Definition B.12 judiciously ignores the length of the proof $\pi$ (of the tautology $T$), viewing the length of the proof as a measure of the complexity of the tautology $T$ with respect to the proof system $M$.

For each tautology $T$, let $\mathcal{PS}_M(T)$ denote the size of the shortest proof of $T$ in $M$ (i.e., the length of the shortest string $\pi$ such that $M$ accepts $(\pi,T)$). That is, $\mathcal{PS}_M$ captures the *proof complexity* of various tautologies with respect to the proof system $M$.

Abusing notation, we let $\mathcal{PS}_M(n)$ denotes the maximum $\mathcal{PS}_M(T)$ over all tautologies $T$ of length $n$. The following simple theorem provides a basic connection between proof complexity (with respect to any propositional proof system) and computational complexity (i.e., the NP-vs-coNP Question).

**Theorem B.13** [58]: *There exists a propositional proof system $M$ such that $\mathcal{PS}_M$ is polynomial if and only if $\mathcal{NP}=\mathrm{co}\mathcal{NP}$.*

In particular, a propositional proof system $M$ such that $\mathcal{PS}_M$ is polynomial coincides with a NP-proof system (as in Definition 2.5) for the set of propositional tautologies, which is a co$\mathcal{NP}$-complete set.

The long-term goal of Proof Complexity is to establish super polynomial lower bounds on the length of proofs in any propositional proof system (and thus establish $\mathcal{NP}\neq\mathrm{co}\mathcal{NP}$). It is natural to start this formidable project by considering first simple (and thus weaker) proof systems, and then move on to more and more complex ones. Moreover, various natural proof systems, capturing basic (restricted) types and "primitives" of reasoning as well as natural tautologies, suggest themselves as objects for this study. In the rest of this section we focus on such restricted proof systems. Different branches of Mathematics such as logic, algebra and geometry provide different such systems, often implicitly. A typical system would have a set of axioms, and a set of deduction rules. A proof would proceed to derive the desired tautology in a sequence of steps, each producing a formula (often called a line of the proof), which is either an axiom, or follows from previous formulae via one of the deduction rules.

Regarding these proof systems, we make two observations. First, proofs in these systems can be easily verified by an algorithm (and thus they fit the general framework of Definition B.12). Second, these proof systems perfectly fit our dag model. The inputs will be labeled by the axioms, the internal vertices by deduction rules, which in turn "infer" a formula for that vertex from the formulae at the vertices pointing to it.[3]

**An important observation:** For various proof systems $\Pi$, we turn to study the proof length $\mathcal{PS}_\Pi(T)$ of tautologies $T$ in proof system $\Pi$. The first observation, revealing a major difference between proof complexity and circuit complexity, is that the trivial counting argument *fails*. The reason is that, while the number of functions on $n$ bits is $2^{2^n}$, there are at most $2^n$ tautologies of this length. Thus, in proof complexity, even the *existence* of a hard tautology, not necessarily an explicit one, would be of interest (and, in particular, if established for any propositional proof system then it would yield $\mathcal{NP} \neq \text{co}\mathcal{NP}$). (Note that here we refer to hard instances of of a problem and not to hard problems.) Anyhow, as we shall see, most known lower bounds (in restricted proof systems) apply to very natural (let alone explicit) tautologies.

**Conventions:** There is an equivalent and somewhat more convenient view of (simple) proof systems, namely as (simple) refutation systems. First, recalling that 3SAT is NP-complete, note that every (negation of a) tautology can be written as a conjunction of clauses, with each clause being a disjunction of only 3 literals (variables or their negation). Now, if we take these clauses as axioms, and derive (using the rules of the system) a contradiction (e.g., the negation of an axiom, or better yet the empty clause), then we have proved the tautology (since we have proved that its negation yields a contradiction). Proof complexity often takes the refutation viewpoint, and often exchanges "tautology" with its negation ("contradiction").

The rest of this section is divided to three parts, referring to logical, algebraic and geometric proof systems. We will briefly describe important representative and basic results in each of these domains, and refer the reader to [24] for further detail (and, in particular, to adequate references).

[3]General proof systems as in Definition B.12 can also be adapted to this formalism, by considering a deduction rule that corresponds to a single step of the machine $M$. However, the deduction rules considered below are even simpler, and more importantly they are natural.

### B.4.1 Logical Proof Systems

The proof systems in this section will all have lines that are Boolean formulae, and the differences will be in the structural limits imposed on these formulae.

The most basic proof system, called Frege system, puts no restriction on the formulae manipulated by the proof. It has one derivation rule, called the cut rule: $A \vee C, B \vee \neg C \vdash A \vee B$ (adding any other sound rule, like *modus ponens*, has little effect on the length of proofs in this system). Frege systems are basic in the sense that they (in several variants) are the most common in Logic, and in that polynomial length proofs in these systems naturally corresponds to "polynomial-time reasoning" about feasible objects.

The major open problem in proof complexity is to find any tautology (as usual, we mean a family of tautologies) that has no polynomial-size proof in the Frege system.

Since lower bounds for Frege are hard, we turn to subsystems of Frege which are interesting and natural. The most widely studied system is Resolution, whose importance stems from its use by most propositional (as well as first order) automated theorem provers. The formulae allowed in Resolution refutations are simply clauses (disjunctions), and so the derivation cut rule simplifies to the "resolution rule": $A \vee x, B \vee \neg x \vdash A \vee B$, for clauses $A, B$ and variable $x$.

An example of a tautology that is easy for Frege and hard for Resolution, is the pigeonhole principle, $\mathtt{PHP}_n^m$, expressing the fact that there is no one-to-one mapping of $m$ pigeons to $n < m$ holes.

**Theorem B.14** $\mathcal{PS}_{\mathrm{Frege}}(\mathtt{PHP}_n^{n+1}) = n^{O(1)}$ *but* $\mathcal{PS}_{\mathrm{Resolution}}(\mathtt{PHP}_n^{n+1}) = 2^{\Omega(n)}$

### B.4.2 Algebraic Proof Systems

Just as a natural contradiction in the Boolean setting is an unsatisfiable collection of clauses, a natural contradiction in the algebraic setting is a system of polynomials without a common root. Moreover, CNF formulae can be easily converted to a system of polynomials, one per clause, over any field. One often adds the polynomials $x_i^2 - x_i$ which ensure Boolean values.

A natural proof system (related to Hilbert's Nullstellensatz, and to computations of Grobner bases in symbolic algebra programs) is Polynomial Calculus, abbreviated PC. The lines in this system are polynomials (represented explicitly by all coefficients), and it has two deduction rules: For any two polynomials $g, h$, the rule $g, h \vdash g + h$, and for any polynomial $g$ and variable $x_i$, the rule $g, x_i \vdash x_i g$. Strong size lower bounds (obtained from degree lower bounds) are known for this system. For example, encoding the pigeonhole principle as a contradicting set of constant degree polynomials, we have

**Theorem B.15** *For every $n$ and every $m > n$, $\mathcal{PS}_{\mathrm{PC}}(\mathtt{PHP}_n^m) \geq 2^{n/2}$, over every field.*

### B.4.3 Geometric Proof Systems

Yet another natural way to represent contradictions is a by a set of regions in space that have empty intersection. Again, we care mainly about discrete (say, Boolean) domains, and a wide source of interesting contradictions are Integer Programs from Combinatorial Optimization. Here, the constraints are (affine) linear inequalities with integer coefficients (so the regions are subsets of the Boolean cube carved out by halfspaces). The most basic system is called Cutting Planes (CP). Its lines are linear inequalities with integer coefficients. Its deduction rules are (the obvious) addition of inequalities, and the (less obvious) division of the coefficients by a constant (and rounding, taking advantage of the integrality of the solution space).

While $\mathtt{PHP}_n^m$ is easy in this system, exponential lower bounds are known for other tautologies. We mention that they are obtained from the *monotone circuit* lower bounds of Section B.2.2.