Chapter 2

#P-completeness

Classical complexity theory is mainly concerned with complexity of decision problems, e.g., "Is a given graph G Hamiltonian?"[1] Formally, a *decision problem* is a predicate $\varphi : \Sigma^* \to \{0,1\}$, where Σ is some finite alphabet in which problem instances are encoded.[2] Thus, $x \in \Sigma^*$ might encode a graph G_x (as an adjacency matrix, perhaps) and $\varphi(x)$ is true iff G_x is Hamiltonian.

The most basic distinction in the theory of computational complexity is between predicates that can be decided in time polynomial in the size $|x|$ of the instance, and those that require greater (often exponential) time. This idea is formalised in the complexity class P of polynomial-time predicates. A predicate φ belongs to the complexity class P (and we say that φ is *polynomial time*) if it can be decided by a deterministic Turing machine in time polynomial in the size of the input; more precisely, there is a deterministic Turing machine T and a polynomial p such that, for every input $x \in \Sigma^*$, T terminates after at most $p(|x|)$ steps, accepting if $\varphi(x)$ is true and rejecting otherwise.[3]

Before proceeding, a few vaguely philosophical remarks addressed to readers who have only a passing acquaintance with computational complexity, with the aim of making the chapter more accessible. One motivation for using a robust class of time bounds (namely, all polynomial functions) in the above definition is to render the complexity class P independent of the model of computation. We ought to be able to substitute any "reasonable"

[1] A *closed* path in G is one that returns to its starting point; a *simple* path is one in which no vertex is repeated; a *Hamilton cycle* in G is a simple closed path that visits every vertex in G. A graph G is *Hamiltonian* if it contains a Hamilton cycle.

[2] Σ^* denotes the set of all finite sequences of symbols in Σ.

[3] Here, $|x|$ denotes the length of the word x.

sequential model of computation for the Turing machine T in the definition and end up with the same class P. By *sequential* here, we mean that the model should be able to perform just one atomic computational step in each time unit. The "Extended Church-Turing Thesis" is the assertion that the class P is independent of the model of computation used to define it. It is a thesis rather than a theorem, because we cannot expect to formalise the condition that the model be "reasonable". The upshot of all this is that the reader unfamiliar with the Turing machine model should mentally replace it by some more congenial model, e.g., that of C programs. (For a more expansive treatment of the fundamentals of machine-based computational complexity, refer to standard texts by Papadimitriou [67] or Garey and Johnson [36].)

The important complexity class NP is usually defined in terms of nondeterministic Turing machines. Indeed, NP stands for "N[ondeterministic] P[olynomial time]". In the interests of accessibility, however, we take an alternative but equivalent approach. We say that a predicate $\varphi : \Sigma^* \to \{0,1\}$ belongs to the class NP iff there exists a polynomial-time "witness-checking" predicate $\chi : \Sigma^* \times \Sigma^* \to \{0,1\}$ and a polynomial p such that, for all $x \in \Sigma^*$,

$$\varphi(x) \iff \exists w \in \Sigma^*.\, \chi(x,w) \wedge |w| \leq p(|x|)\,. \tag{2.1}$$

(Since the term "polynomial time" has been defined only for monadic predicates, it cannot strictly be applied to χ. Formally, what we mean here is that there is a polynomial-time Turing machine T that takes an input of the form $x\$y$ — where $x, y \in \Sigma^*$ and $\$ \notin \Sigma$ is a special separating symbol — and accepts iff $\chi(x,y)$ is true. The machine T is required to halt in a number of steps polynomial in $|x\$y|$.)

Example 2.1. Suppose x encodes an undirected graph G, y encodes a subgraph H of G, and $\chi(x,y)$ is true iff y is a Hamilton cycle in G. The predicate χ is easily seen to be polynomial time: one only needs to check that H is connected, that H spans G, and that every vertex of H has degree two. Since χ is clearly a witness-checker for Hamiltonicity, we see immediately that the problem of deciding whether a graph is Hamiltonian is in the class NP. Many "natural" decision problems will be seen, on reflection, to belong to the class NP.

As is quite widely known, it is possible to identify within NP a subset of "NP-complete" predicates which are computationally the "hardest" in NP. Since we shall shortly be revisiting the phenomenon of completeness in the context of the counting complexity class #P, just a rough sketch of how this is done will suffice. The idea is to define a notion of reducibility between predicates — polynomial-time many-one (or Karp) reducibility — that allows us to compare their relative computational difficulty. A predicate φ is *NP-hard* if every predicate in NP is reducible to φ; it is *NP-complete* if, in addition, $\varphi \in \mathrm{NP}$.

Logically, there are two possible scenarios: either $\mathrm{P} = \mathrm{NP}$, in which case all predicates in NP are efficiently decidable, or $\mathrm{P} \subset \mathrm{NP}$, in which case no NP-complete predicate is decidable in polynomial time. Informally, this dichotomy arises because the complete problems are the hardest in NP; formally, it is because the complexity class P is closed under polynomial-time many-one reducibility. Since the former scenario is thought to be unlikely, NP-completeness provides strong circumstantial evidence for intractability. The celebrated theorem of Cook provides a natural example of an NP-complete predicate, namely deciding whether a propositional formula Φ in CNF has a model, i.e., whether Φ is satisfiable. For convenience, this decision problem is referred to as "SAT".

2.1 The class #P

Now we are interested extending the above framework to *counting problems* — e.g., "How many Hamiltonian cycles does a given graph have?" — which can be viewed as functions $f : \Sigma^* \to \mathbb{N}$ mapping (encodings of) problem instances to natural numbers. The class P must be slightly amended to account for the fact we are dealing with functions with codomain $\mathbb{N}$ rather than predicates. A counting problem $f : \Sigma^* \to \mathbb{N}$ is said to belong to the complexity class[4] FP if it is computable by a deterministic Turing machine transducer[5] in time polynomial in the size of the input. As we saw in Chapter 1 (see Theorems 1.1 and 1.11), the following problems are in FP:

Name. #SPANNINGTREES
Instance. A graph G.
Output. The number of spanning trees in G.

Name. #PLANARPM
Instance. A planar graph G.
Output. The number of perfect matchings in G.

[4] Standing for "F[unction] P[olynomial time]" or something similar.
[5] That is, by a TM with a write-only output tape.

The analogue of NP for counting problems was introduced by Valiant [76]. A counting problem $f : \Sigma^* \to \mathbb{N}$ is said to belong to the complexity class #P if there exist a polynomial-time predicate $\chi : \Sigma^* \times \Sigma^* \to \{0, 1\}$ and a polynomial p such that, for all $x \in \Sigma^*$,

$$f(x) = \left|\left\{ w \in \Sigma^* : \chi(x, w) \wedge |w| \leq p(|x|) \right\}\right| . \tag{2.2}$$

The problem of counting Hamilton cycles in a graph is in #P by identical reasoning to that used in Example 2.1. The complexity class #P is very rich in natural counting problems. Note that elementary considerations entail $\mathrm{FP} \subseteq \#\mathrm{P}$.

Now, how could we convince ourselves that a problem f is *not* efficiently solvable? Of course, one possibility would be to *prove* that $f \notin \mathrm{FP}$. Unfortunately, such absolute results are beyond the capabilities of the current mathematical theory. Still, as in the case of decision problems, it is possible to provide persuasive evidence for the intractability of a counting problem, based on the assumption that there is *some* problem in #P that is not computable in polynomial time, i.e., that $\mathrm{FP} \neq \#\mathrm{P}$.[6] With this in mind, we are going to define a class of "most difficult" problems in #P, the so-called #P-complete problems, which have the property that if they are in FP, then #P collapses to FP. In other words, if $\mathrm{FP} \subset \#\mathrm{P}$ then no #P-complete counting problem is polynomial-time solvable. For this purpose, we seem to need a notion of reducibility that is more general than the usual many-one reducibility.

Given functions $f, g : \Sigma^* \to \mathbb{N}$, we say that g is *polynomial-time Turing* (or Cook) *reducible* to f, denoted $g \leq_{\mathrm{T}} f$, if there is a Turing machine with an oracle[7] for f that computes g in time polynomial in the input size. The relation $\leq_{\mathrm{T}}$ is transitive; moreover,

$$f \in \mathrm{FP} \wedge g \leq_{\mathrm{T}} f \Rightarrow g \in \mathrm{FP} . \tag{2.3}$$

A function f is #P-*hard* if every function in #P is Turing reducible to f; it is #P-*complete* if, in addition, $f \in \#\mathrm{P}$. Just as with the class NP, we

[6] This is clearly the counting analogue of the notorious $\mathrm{P} \neq \mathrm{NP}$ conjecture. Note, however, that $\mathrm{FP} \neq \#\mathrm{P}$ might hold even in the unlikely event that $\mathrm{P} = \mathrm{NP}$!

[7] An *oracle for f* is an addition to the Turing machine model, featuring a write-only query tape and a read-only response tape. A query $q \in \Sigma^*$ is first written onto the query tape; when the machine goes into a special "query state" the query and response tapes are both cleared and the response $f(x)$ written to the response tape. The oracle is deemed to produce the response in just one time step. In conventional programming language terms, an oracle is a subroutine or procedure, where we discount the time spent executing the body of the procedure.

have a dichotomy: either FP = #P or no #P-complete counting problem is polynomial-time solvable. Formally, this follows from (2.3), which expresses the fact that FP is closed under polynomial-time Turing reducibility.

What are examples of #P-complete problems? For one thing, the usual generic reduction of a problem in NP to SAT used to prove Cook's theorem is "parsimonious", i.e., it preserves the number of witnesses (satisfying assignments in the case of SAT). It follows that #SAT is #P-complete:

> *Name.* #SAT
> *Instance.* A propositional formula Φ in conjunctive normal form (CNF).
> *Output.* The number of models of (or satisfying assignments to) Φ.

More generally, it appears that NP-complete decision problems tend to give rise to #P-complete counting problems. To be a little more precise: any polynomial-time witness checking function χ gives rise to an NP decision problem Π via (2.1) and a corresponding counting problem $\#\Pi$ via (2.2). Empirically, whenever the decision problem Π is NP-complete, the corresponding counting problem $\#\Pi$ is #P-complete. Simon [70] lists many examples of this phenomenon, and no counterexamples are known. What he observes is that the existing reductions used to establish NP-completeness of decision problems Π are often parsimonious and hence establish also #P-completeness of the corresponding counting problem $\#\Pi$. When the existing reduction is not parsimonious it can be modified so that it becomes so.

Open Problem. Is it the case that for every polynomial-time witness-checking predicate χ, the counting problem $\#\Pi$ is #P-complete whenever the decision problem Π is NP-complete? I conjecture the answer is "no", but resolving the question may be difficult. Note that a negative answer could only reasonably be established relative to some complexity theoretic assumption, since it would entail FP $\subset$ #P. Indeed, if FP were to equal #P then every function in #P would be trivially #P-complete.

2.2 A primal #P-complete problem

What makes the theory of #P-completeness interesting is that the converse to the above conjecture is definitely false; that is, there are #P-complete counting problems $\#\Pi$ corresponding to easy decision problems $\Pi \in$ P. A celebrated example [76] is #BIPARTITEPM, that has an alternative formulation as 0,1-PERM:

Name. #BIPARTITEPM

Instance. A bipartite graph G.

Output. The number of perfect matchings in G.

Name. 0,1-PERM

Instance. A square 0,1-matrix $A = (a_{ij} : 0 \leq i, j < n)$.

Output. The *permanent*

$$\operatorname{per} A = \sum_{\sigma \in S_n} \prod_{i=0}^{n-1} a_{i,\sigma(i)}$$

of A. Here, S_n denotes the symmetric group, i.e., the sum is over all $n!$ permutations of $[n]$.

To see the correspondence, suppose, for convenience, that G has vertex set $[n] + [n]$, and interpret A as the adjacency matrix of G; thus $a_{ij} = 1$ if (i, j) is an edge of G and $a_{ij} = 0$ otherwise. Then $\operatorname{per} A$ is just the number of perfect matchings in G. In particular, the following theorem implies that planarity (or some slightly weaker assumption) is crucial for the Kasteleyn result (Theorem 1.11).

Theorem 2.2 (Valiant). 0,1-PERM *(equivalently,* #BIPARTITEPM*) is* #P*-complete.*

It is clear that 0,1-PERM is in #P: the obvious "witnesses" are permutations σ satisfying $\prod_i a_{i,\sigma(i)} = 1$. To prove #P-hardness, we use a sequence of reductions starting at #EXACT3COVER and going via a couple of auxiliary problems #wBIPARTITEMATCH and #wBIPARTITEPM.

Name. #EXACT3COVER

Instance. A set X together with a collection $T \subseteq \binom{X}{3}$ of unordered triples[8] of X.

Output. The number of subcollections $S \subseteq T$ that cover X without overlaps; that is every element of X should be contained in precisely one triple in S.

[8] I'm not sure if $\binom{X}{3}$ is a standard notation for "the set of all unordered triples from X", but it seems natural enough, given the notation 2^X.

Name. #wBipartiteMatch

Instance. A bipartite graph G with edge weights $w : E(G) \to \{1, -1, -\frac{5}{3}, \frac{1}{6}\}$. (Why exactly these weights are used will become clearer in the course of the proof.)

Output. The "total weight" of matchings $p_{\mathrm{match}}(G) = \sum_M w(M)$, where M ranges over *all* matchings in G and the weight of a matching is $w(M) = \prod_{e \in M} w(e)$.

Name. #wBipartitePM

Instance. As for #wBipartiteMatch.

Output. As for #wBipartiteMatch, but with "perfect matchings" replacing "matchings".

Remark 2.3. More generally, we might consider a graph G with edge weighting $w : E(G) \to Z \cup \mathbb{C}$, where Z is a set of indeterminates. In this case the expression $p_{\mathrm{match}}(G) = \sum_M w(M)$ appearing in the definition of #wBipartiteMatch is a polynomial in Z. If every edge is assigned a distinct indeterminate, then $p_{\mathrm{match}}(G)$ is the *matching polynomial* of G, i.e., the generating function for matchings in G.

Since #Exact3Cover is the counting version of an NP-complete problem, we expect it to be #P-complete via parsimonious reduction.

Fact 2.4. #Exact3Cover *is* #P*-complete.*

Exercise 2.5. (This exercise is mainly directed to readers with some exposure to computational complexity.) Garey and Johnson [36, §7.3] note Fact 2.4 without proof. Since I am not aware of any published proof, we should maybe pause to provide one. Garey and Johnson's reduction [36, §3.1.2] from 3Sat (the restriction of Sat to formulas with three literals per clause) to Exact3Cover (actually a special case of Exact3Cover called "3-dimensional matching") is almost parsimonious. The "truth setting component" is fine (each truth assignment corresponds to exactly one pattern of triples). The "garbage collection component" is also fine (it is not strictly parsimonious, but the number of patterns of triples is independent of the truth assignment, which is just as good). The "satisfaction testing component" needs some attention, as the number of patterns of triples depends on the truth assignment. However, with a slight modification, this defect may be corrected. Finally, to do a thorough job, we really ought to modify Garey and Johnson's reduction [36, §3.1.1] from Sat to 3Sat to make it parsimonious too.

In the light of Fact 2.4, Theorem 2.2 will follow from the following series of lemmas:

Lemma A. #EXACT3COVER $\leq_{\mathrm{T}}$ #wBIPARTITEMATCH.

Lemma B. #wBIPARTITEMATCH $\leq_{\mathrm{T}}$ #wBIPARTITEPM.

Lemma C. #wBIPARTITEPM $\leq_{\mathrm{T}}$ #BIPARTITEPM ($\equiv$ 0,1-PERM).

Proof of Lemma A. Our construction is based on the weighted bipartite graph H (depicted in Figure 2.1), where the weights of the edges on the left are as indicated, and the edges labelled a_1, a_2 and a_3 will presently all be assigned weight 1. Initially, however, to facilitate discussion, we assign to these edges distinct indeterminates z_1, z_2 and z_3, respectively.

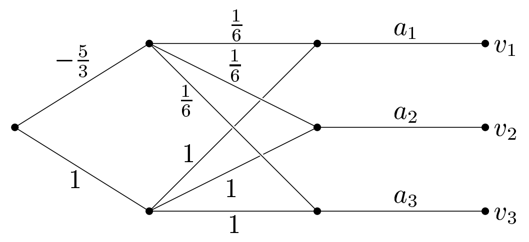

Figure 2.1: The graph H.

By direct computation, the matching polynomial of H, with weights as specified, is

$$p_{\mathrm{match}}(H) = (1 + z_1 z_2 z_3)/3. \tag{2.4}$$

Let us see how to verify (2.4) by calculating the coefficient of $z_1 z_2 z_3$; the other coefficients can be calculated similarly. (Note that there there are only four calculations since, by symmetry, only the *degree* of the monomial is significant.) So suppose we include all three edges a_1, a_2 and a_3, as we must do in order to get a matching that contributes to the coefficient of $z_1 z_2 z_3$. Then we can either add no further edge at all, or add the lower left edge with weight 1, or the upper left edge with weight $-\frac{5}{3}$. Thus, the total weight of such matchings is $(1 + 1 - \frac{5}{3}) z_1 z_2 z_3 = \frac{1}{3} z_1 z_2 z_3$.

Equation (2.4) succinctly expresses the key properties of H that we use. Suppose that H is an (induced) subgraph of a larger graph G, and that H is connected to the rest of G only via the vertices v_1, v_2 and v_3; more precisely, there are no edges of G incident to vertices $V(H) \setminus \{v_1, v_2, v_3\}$ other than the ones depicted. Consider some matching $M' \subseteq E(G) \setminus (E(H) \setminus \{a_1, a_2, a_3\})$ in G, i.e., one that does not use edges from H except perhaps a_1, a_2 and a_3. We call a matching $M \supseteq M'$ in G an *extension* of M' if it agrees with M' on the edge set $E(G) \setminus (E(H) \setminus \{a_1, a_2, a_3\})$. If M' includes all three edges a_i, then the total weight of extensions of M' to a matching M on the whole of G is $\frac{1}{3}w(M')$; a similar claim holds if M' excludes all three edges a_i. In contrast, if M includes some edges a_i and excludes others, then the total weight of extensions of M' is zero. Informally, H acts as a "coordinator" of the three edges a_i.

Using the facts encapsulated in (2.4), we proceed with the reduction of #EXACT3COVER to #wBIPARTITEMATCH. An instance of #EXACT-3COVER consists of an underlying set X, and a collection $T \subseteq \binom{X}{3}$ of triples; for convenience set $n := |X|$ and $m := |T|$. We construct a bipartite graph G as follows. Take a separate copy H_t of H for each triple $t = \{\alpha, \beta, \gamma\} \in T$ and label the three pendant edges of H_t with a^t_α, a^t_β, and a^t_γ, respectively. Furthermore, for each $\alpha \in X$, introduce vertices v_α and u_α, and connect them by an edge $\{v_\alpha, u_\alpha\}$ of weight -1. Finally, identify the right endpoint of the edge a^t_α with the vertex v_α whenever $\alpha \in t$ (see Figure 2.2).

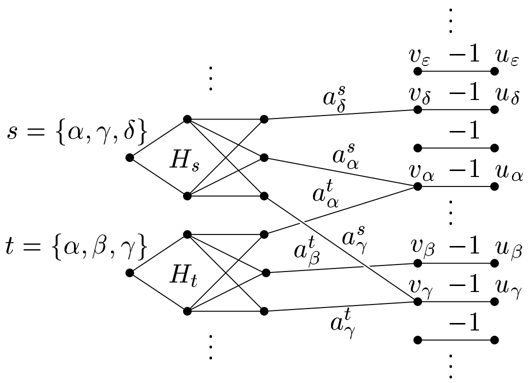

Figure 2.2: A sketch of the graph G.

Recall that the matching polynomial of G is a sum over matchings M in G of the weight $w(M)$ of M. We partition this sum according to the restriction $A = M \cap I$ of M to I, where $I := \{a^t_\alpha : t \in T \wedge \alpha \in t\}$. Computing the total weight of extensions of A to a matching in G is straightforward. For each of the subgraphs H_t, equation (2.4) gives the total weight of extensions of A to that subgraph. For each of the edges $\{v_\alpha, u_\alpha\}$, the total weight of extensions of A to that edge is simply 1 if v_α is covered by A and $(1-1)=0$ otherwise. Expressing these considerations symbolically yields the following expression for the matching polynomial of G:

$$p_{\mathrm{match}}(G) = \sum_M w(M) = \sum_{A\subseteq I} \prod_{t\in T} \varphi_t(A) \prod_{\alpha\in X} \psi_\alpha(A)\,, \tag{2.5}$$

where

$$\varphi_t(A) = \begin{cases} \frac{1}{3}, & \text{if } a^t_\alpha \in A \text{ for all } \alpha\in t; \\ \frac{1}{3}, & \text{if } a^t_\alpha \notin A \text{ for all } \alpha\in t; \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\psi_\alpha(A) = \begin{cases} 1, & \text{if } a^t_\alpha \in A \text{ for some } t\ni\alpha \\ 0, & \text{otherwise.} \end{cases}.$$

Each edge subset A contributing a non-zero term to the sum (2.5) corresponds to an exact 3-cover of X: no element α of X is covered twice (property of a matching), no element of X is uncovered (property of ψ_α), and no triple t is subdivided (property of φ_t). Since every exact 3-cover contributes $(\frac{1}{3})^m$ to (2.5), we obtain

$$p_{\mathrm{match}}(G) = \left(\frac{1}{3}\right)^m \big|\{\,\text{exact 3-covers of } X \text{ by triples in } T\,\}\big|. \tag{2.6}$$

Thus, assuming we have an oracle for the left hand side of (2.6), we can compute the number of exact three covers in time polynomial in m and n, and hence polynomial in the size of the #EXACT3COVER instance (X,T). □

Proof of Lemma B. Let G be an instance of #wBIPARTITEMATCH, that is, a bipartite graph with vertex set $V = R\cup B$ and edge set E, where $\binom{R}{2}\cap E = \binom{B}{2}\cap E = \emptyset$, and edge weights from $\{1,-1,\frac{1}{6},-\frac{5}{3}\}$. Set $r := |R|$ and $b := |B|$, so that $r+b = n := |V|$.

For $0\le k\le \min\{r,b\}$ (the maximal possible cardinality of a matching in G), we construct a bipartite graph graph G_k as follows. Take a set R' and a set B' of new vertices, $|R'| = b-k$ and $|B'| = r-k$ and connect each vertex in R with each vertex in B' and each vertex in R with each vertex in R' by new edges of weight 1 (see Figure 2.3).
In a similar vein to the proof of Lemma A, we observe that

$$\sum_{\substack{M' \text{ is a perfect} \\ \text{matching in } G_k}} w(M') = (r-k)!\,(b-k)! \sum_{\substack{M \text{ is a } k\text{-} \\ \text{matching in } G}} w(M)\,.$$

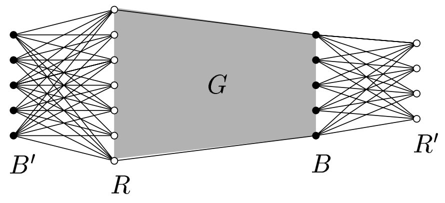

Figure 2.3: The graph G_k.

Thus, we can compute the total weight of matchings in G by invoking our oracle for #wBipartitePM on every G_k and summing over k. □

Proof of Lemma C. Let $G = (V, E)$ be an instance of #BipartitePM, with $|V| = n$. We get rid of the weights one by one using interpolation. Consider a certain weight $\zeta \in \{\frac{1}{6}, -1, -\frac{5}{3}\}$. If we replace it by an indeterminate z, then

$$p(z) := \sum_{\substack{M \text{ is a perfect} \\ \text{matching in } G}} w(M)$$

is a polynomial of degree $d \leq \frac{1}{2}n$. If we can evaluate p at $d+1$ distinct points, say at $k = 1, \ldots, d+1$, we can interpolate to find $p(\zeta)$. (Refer to Valiant [78] for a discussion of efficient interpolation.) In order to find $p(k)$ for fixed k, we construct a graph G_k from G by replacing each edge $\{u, v\}$ of weight z by k disjoint paths of length 3 between u and v such that each edge on these paths has weight 1 (see Figure 2.4).

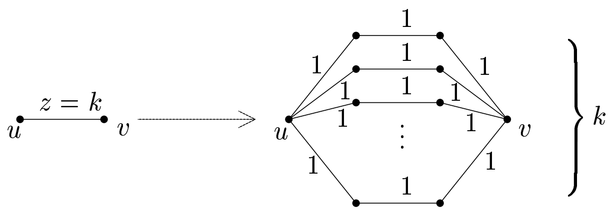

Figure 2.4: Substituting k disjoint paths for an edge.

Then $p(k) = \big|\{\text{perfect matchings in } G_k\}\big|$, and we can determine the right hand side by means of our oracle for #BipartitePM. This completes the proof of the last lemma, and hence of the theorem. □

Remarks 2.6. (a) The intermediate problems in the above proof are not in #P; however, they are "#P-easy", i.e., Turing reducible to a function in #P.

(b) #P-hard counting problems are ubiquitous. In fact, the counting problems in FP are very much the exceptions. The ones we encountered in Chapter 1 — counting trees in directed and undirected graphs (and the related Eulerian circuits in a directed graph), and perfect matchings in a graph (and the related partition function of a planar ferromagnetic Ising system) — are pretty much the only non-trivial examples.

(c) Our reduction from #Exact3Cover to #BipartitePM used polynomial interpolation in an essential way. Indeed, interpolation features prominently in a majority of #P-completeness proofs. The decision to define #P-completeness with respect to Turing reducibility rather than many-one reducibility is largely motivated by the need to perform many polynomial evaluations (which equate to oracle calls) rather than just one. It is not clear whether the phenomenon of #P-completeness would be as ubiquitous if many-one reducibility were to be used in place of Turing.

(d) Following from the previous observation: Polynomial interpolation is not numerically stable, and does not preserve closeness of approximation. Specifically, we may need to evaluate a polynomial to very great accuracy in order to know some coefficient even approximately. Thus we cannot deduce from the reductions in Lemmas A–C above that *approximating* the permanent is computationally hard, even though approximating #Exact3Cover is. We exploit this loophole in Chapter 5.

(e) Every problem in NP is trivially #P-easy. It is natural to ask how much bigger #P is than NP. The answer seems to be that it is much bigger. The complexity class PH (Polynomial Hierarchy) is defined similarly to NP, except that arbitrary quantification is allowed in equivalence (2.1), in place of simple existential quantification. PH seems intuitively to be "much bigger" than NP. Yet it is a consequence of Toda's theorem (see [73]) that every problem in PH is #P-easy!

2.3 Computing the permanent is hard on average

While many NP-complete problems are easy to decide on random instances, this does not seem to be the case for counting problems. For example, consider an (imperfect) algorithm $\mathcal{A}$ for computing the permanent of $n \times n$ matrices A over the field $\mathrm{GF}(p)$, for all $n \in \mathbb{N}$ and all primes p, with the following specification:

1. $\mathcal{A}$ has runtime polynomial in n and p;

2. For each n and each p, $\mathcal{A}$ must give the correct result except on some fraction $\frac{1}{3(n+1)}$ of all $n \times n$ matrices over $\mathrm{GF}(p)$.

Theorem 2.7. *No algorithm $\mathcal{A}$ with the above specification exists unless every problem in #P admits a polynomial-time randomised algorithm with low*[9] *error probability.*[10]

Proof. It suffices to show that some particular #P-complete problem, namely 0,1-PERM, admits a polynomial-time randomised algorithm with low error probability. Given an $n \times n$ matrix A with entries from $\{0, 1\}$, if we know $\operatorname{per} A \pmod{p_i}$ for a sequence $p_1, p_2, \ldots, p_n$ of n distinct primes larger than $n + 1$, then we can use "Chinese remaindering" to evaluate $\operatorname{per} A$. The method is as follows. If a and b are relatively coprime natural numbers, we can write $1 = ca + db$ with integer coefficients c, d, which can be found by means of the Euclidian algorithm. Now suppose we know the residues $r = x \bmod a$ and $s = x \bmod b$ of an integer x. If we set $y := rdb + sca$, we have $x \equiv y \pmod{a}$ and $x \equiv y \pmod{b}$, and hence $x \equiv y \pmod{ab}$, by relative primality. Thus, inductively, we can compute $(\operatorname{per} A) \bmod p_1 p_2 \ldots p_n$ from the n values $(\operatorname{per} A) \bmod p_i$. But since $\operatorname{per} A$ is a natural number not larger than $n! < p_1 p_2 \ldots p_n$, it is uniquely determined by its residue modulo $p_1 p_2 \ldots p_n$. Moreover, the Prime Number Theorem ensures that we may take the p_i's to be no larger than $O(n \ln n)$; in particular, we can find them by brute force in time polynomial in n.

Thus, it remains to show how, for a fixed prime $n + 2 \leq p \leq O(n \ln n)$, we can employ $\mathcal{A}$ to compute $(\operatorname{per} A) \bmod p$ with low error probability. For

[9] We can take "low" to mean $\frac{1}{3}$, since this may be reduced to an arbitrarily small value by repeatedly running $\mathcal{A}$ and taking a majority vote. Note that the error probability decreases to zero exponentially as a function of the number of trials.

[10] The error probability is with respect to random choices made by the algorithm. The input is assumed non-random.

this purpose, we select a matrix R u.a.r. from all $n \times n$ matrices over $\mathrm{GF}(p)$. Let z be an indeterminate and consider

$$p(z) := \mathrm{per}(A + zR),$$

regarded as a polynomial of degree at most n with coefficients in $\mathrm{GF}(p)$. Using $\mathcal{A}$, we evaluate (in time polynomial in n and p, hence in n) $p(z)$ at $n+1$ points $z = 1, 2, \ldots, n+1$. Observe that, since the numbers $1, \ldots, n+1$ are invertible modulo p, $A + R$, $A + 2R, \ldots, A + (n+1)R$ are again random matrices (over $\mathrm{GF}(p)$). Thus, with probability at least $1 - (n+1)\frac{1}{3(n+1)} = \frac{2}{3}$, $\mathcal{A}$ will give the correct answer in all instances. Now we interpolate to find $p(0) = (\mathrm{per}\, A) \bmod p$. $\square$

Remarks 2.8. (a) Feige and Lund [33] have considerably sharpened Theorem 2.7 using techniques from the theory of error-correcting codes.

(b) The property (of a problem) of being as hard on average as in the worst case holds quite generally in high enough complexity classes. Refer to Feigenbaum and Fortnow [34] for a discussion of this phenomenon.

Open Problem. What is the complexity of computing the permanent of a random $0,1$-matrix? It is reasonable to conjecture that computing the permanent of a $0,1$-matrix *exactly* is as hard on average as it is in the worst case. However, this purely combinatorial version of the problem leaves no space for the interpolation that was at the heart of the proof of Theorem 2.7.