§13. Tough nuts. Starting with Sat, and using repeated applications of Theorem 12.2, very many naturally occurring problems can be shown to be NP-complete. Some of these problems are listed below.

(1) The 3-Sat problem.

Instance: A CNF Boolean formula ϕ with at most three literals per clause.

Question: Is there an assignment of truth values to the variables of ϕ that makes ϕ true?

(2) The Clique problem.

Instance: An undirected graph $G = (V, E)$, and an integer k.

Question: Does G possess a k-clique? (A k-clique is a subset $U \subseteq V$ of size k, such that every pair of distinct vertices in U is joined by an edge.)

(3) The Colourability problem.

Instance: An undirected graph G, and an integer k.

Question: Is there an assignment of k colours to the vertices of G such that no two adjacent vertices receive the same colour?

(4) The *Directed Hamiltonian Cycle* problem, DHC.

Instance: A directed graph G.

Question: Does G possess a Hamiltonian cycle, i.e., a directed closed path that visits every vertex of G precisely once?

(5) The *Independent Set* problem, IndSet.

Instance: An undirected graph G, and an integer k.

Question: Does G contain an independent set of size k? (A subset U of the vertex set of G is an *independent set* if and only if no edge in G has both endpoints in U.)

(6) The *Integer Programming* problem, IntProg.

Instance: An integer $m \times n$ matrix A, and an integer m-vector $\mathbf{b}$.

Question: Does there exist an integer n-vector $\mathbf{x}$ such that $A\mathbf{x} \leq \mathbf{b}$? (The vector inequality signifies that each component on the left is to be less than or equal to the corresponding component on the right.)

(7) The SubsetSum problem.

INSTANCE: A finite set X, a positive integer 'size' $s(x)$ for each $x \in X$, and an integer 'goal' b.

QUESTION: Is there a subset $A \subseteq X$ such that $\sum_{x \in A} s(x) = b$?

(8) The *Undirected Hamiltonian Cycle* problem, UHC.

INSTANCE: An undirected graph G.

QUESTION: Does G possess a Hamiltonian cycle, i.e., a closed path which visits every vertex of G precisely once?

(9) The *Vertex Cover* problem, VC.

INSTANCE: An undirected graph G and an integer k.

QUESTION: Does G possess a vertex cover of size k? (A subset U of the vertex set of G is a *vertex cover* for G if and only if every edge of G has at least one endpoint in U.)

With the exception of IntProg, it is a straightforward task to verify that each of the above problems is in NP. The problem IntProg is also in NP, but verifying this fact is a little tricky.

We have already demonstrated (see Theorem 12.3) that Clique is NP-complete. The other eight problems are dealt with below.

THEOREM 13.1 IndSet *is NP-complete.*

PROOF. We shall exhibit a reduction from Clique to IndSet. It will follow, from Theorem 12.2, that IndSet is NP-hard. Here, as in subsequent proofs, the task of verifying that the language in question is a member of NP is left as an exercise for the reader.

Let $G = (V, E)$ be an undirected graph, and k a positive integer; taken together, G and k form an arbitrary instance of Clique. We seek to transform the pair (G, k) into an instance (G', k') of IndSet such that

$$G \text{ has a } k\text{-clique} \iff G' \text{ has an independent set of size } k'. \tag{1}$$

The sought-for graph $G' = (V, \overline{E}\,)$ has the same vertex set as G, but complementary edge set

$$\overline{E} = \{\{u, v\} \mid u, v \in V,\ u \neq v \text{ and } \{u, v\} \notin E\};$$

the integer k' is set equal to k.

Observe that $U \subseteq V$ is a k-clique in G if and only if U is an independent set of size k in G'. This verifies assertion (1), and demonstrates that the function that maps (G, k) to (G', k') is reduction from Clique to IndSet. Clearly, the reduction can be computed in polynomial time. □

THEOREM 13.2 *VC is NP-complete.*

PROOF. The proof, by reduction from INDSET, is left as an exercise for the reader. □

THEOREM 13.3 3-SAT *is NP-complete.*

PROOF. We demonstrate that 3-SAT is NP-complete by exhibiting a polynomial-time reduction from SAT to 3-SAT. Let ϕ be a CNF Boolean formula in the variables $x_1, x_2, \ldots, x_n$. We transform the formula ϕ, in polynomial time, into a 3-CNF formula ϕ', such that ϕ is satisfiable if and only if ϕ' is.

Let $C = (\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_k)$ be a clause of ϕ containing $k > 3$ literals. Introduce a new variable y_1, and consider a *particular* assignment to the variables $x_1, \ldots, x_n$. It is clear that the clause C is true under the assignment if and only if there is a setting of the variable y_1 that makes the clauses $(\alpha_1 \vee \alpha_2 \vee y_1)$ and $(\neg y_1 \vee \alpha_3 \vee \cdots \vee \alpha_k)$ *simultaneously* true. Note that the second clause contains one literal fewer than the original clause C. Iterating the procedure $k-3$ times, we see that C is true if and only if there is some assignment to the variables $y_1, \ldots, y_{k-3}$ which makes the following formula true:

$$(\alpha_1 \vee \alpha_2 \vee y_1) \wedge (\neg y_1 \vee \alpha_3 \vee y_2) \wedge (\neg y_2 \vee \alpha_4 \vee y_3) \wedge \cdots \wedge (\neg y_{k-3} \vee \alpha_{k-1} \vee \alpha_k).$$

The formula ϕ' is obtained from ϕ by replacing each clause in ϕ by a conjunction of several clauses obtained by the above construction. (The transformation of each clause of ϕ involves the introduction of a set of new variables; it should be noted that these sets are pairwise disjoint.) The formula ϕ' is in CNF and has no clause containing more than three literals; moreover ϕ' is satisfiable if and only if ϕ is. It is clear that ϕ' is computable from ϕ in polynomial time. □

THEOREM 13.4 *DHC is NP-complete.*

PROOF. The proof, which involves a somewhat intricate reduction from VC, is beyond the scope of the course.[27] □

THEOREM 13.5 *UHC is NP-complete.*

PROOF. We exhibit a reduction from DHC to UHC. Let $G = (V, E)$ be any directed graph. Define $G' = (V', E')$ to be the undirected graph with vertex set

$$V' = V \times \{0, 1, 2\},$$

and edge set

$$\begin{aligned} E' = &\{\{\langle v, 0\rangle, \langle v, 1\rangle\} \mid v \in V\} \\ &\cup \{\{\langle v, 1\rangle, \langle v, 2\rangle\} \mid v \in V\} \\ &\cup \{\{\langle u, 2\rangle, \langle v, 0\rangle\} \mid (u, v) \in E\}. \end{aligned}$$

(Each vertex in G is expanded into a chain of three vertices in G'. Incoming edges are attached to the first vertex in the appropriate chain, and outgoing edges are attached to

[27] See, for example, A. Gibbons, *Algorithmic Graph Theory*, CUP, p. 230.

the final vertex.) We claim that the function which maps each directed graph G to the undirected graph G' is a polynomial-time reduction from DHC to UHC.

Suppose G contains a (directed) Hamiltonian cycle C. Then the directed cycle C may be transformed into a Hamiltonian cycle C' in G' by expanding each directed edge (u, v) in C into a sequence $\{\langle u, 0\rangle, \langle u, 1\rangle\}$, $\{\langle u, 1\rangle, \langle u, 2\rangle\}$, $\{\langle u, 2\rangle, \langle v, 0\rangle\}$ of three undirected edges in C'. Thus G' contains a Hamiltonian cycle if G does.

Conversely, suppose G' contains an (undirected) Hamiltonian cycle C'. Since C' visits all the vertices in G', it includes the edges $\{\langle v, 0\rangle, \langle v, 1\rangle\}$, $\{\langle v, 1\rangle, \langle v, 2\rangle\}$ for all $v \in V$. Thus, during a traversal of the cycle C', the second component of the vertices encountered must follow the sequence $0, 1, 2, 0, 1, 2, \ldots$, or its reversal. Contracting each sequence $\{\langle u, 0\rangle, \langle u, 1\rangle\}$, $\{\langle u, 1\rangle, \langle u, 2\rangle\}$, $\{\langle u, 2\rangle, \langle v, 0\rangle\}$ of three undirected edges in C' to the single directed edge (u, v), we obtain a directed Hamiltonian cycle in G. Thus G has a Hamiltonian cycle if G' has.

The verification of the reduction is completed by noting that G' can be computed from G in polynomial time. □

THEOREM 13.6 COLOURABILITY *is NP-complete.*

PROOF. We employ a reduction from SAT to COLOURABILITY. Let ϕ be a typical instance of SAT, i.e., a Boolean formula in CNF. Let n be the number of variables in ϕ, and r the number of clauses. We must show how to construct, in polynomial time, an undirected graph G such that

$$\phi \text{ is satisfiable} \iff G \text{ is } (n+1)\text{-colourable.} \tag{2}$$

(A graph G is *k-colourable* if there is an assignment of k colours to the vertices of G such that no pair of adjacent vertices share the same colour.)

Let $x_1, x_2, \ldots, x_n$ be the variables of ϕ, and $C_1, C_2, \ldots, C_r$ be the clauses. Let $v_0, v_1, v_2, \ldots, v_n$ be new symbols. The constructed graph $G = (V, E)$ has vertex set

$$V = \{v_0, \ldots, v_n\} \cup \{x_1, \ldots, x_n\} \cup \{\overline{x}_1, \ldots, \overline{x}_n\} \cup \{C_1, \ldots, C_r\},$$

and edge set

$$\begin{aligned} E = &\{\{v_i, x_j\}, \{v_i, \overline{x}_j\} \mid 1 \le i, j \le n \text{ and } i \ne j\} \\ &\cup \{\{v_i, v_j\} \mid 0 \le i < j \le n\} \\ &\cup \{\{v_0, C_k\} \mid 1 \le k \le r\} \\ &\cup \{\{x_i, \overline{x}_i\} \mid 1 \le i \le n\} \\ &\cup \{\{x_i, C_k\} \mid x_i \text{ is } \textit{not} \text{ a literal in clause } C_k\} \\ &\cup \{\{\overline{x}_i, C_k\} \mid \neg x_i \text{ is } \textit{not} \text{ a literal in clause } C_k\}. \end{aligned}$$

(It will be seen that we have confused vertices with variables and clauses: this is notationally very convenient, and ambiguities can be resolved by context. Each vertex $\overline{x}_i$ corresponds, in a sense which will become clear, to the negated variable $\neg x_i$.)

Now suppose that ϕ is satisfiable. We shall demonstrate how to colour the vertices of G with $n+1$ colours, which for convenience we refer to as $0, 1, 2, \ldots, n$. Consider a particular assignment of truth values to the variables x_i which satisfies ϕ. Then the following is a valid $(n+1)$-colouring of G:

(a) Each vertex v_i receives colour i.

(b) For each variable x_i which is *true* under the assignment, vertex x_i receives colour i and vertex $\overline{x}_i$ receives colour 0.

(c) For each variable x_i which is *false* under the assignment, vertex x_i receives colour 0 and vertex $\overline{x}_i$ receives colour i.

(d) From each clause C_k, select a literal, say x_i or $\neg x_i$, which is true under the assignment; then vertex C_k receives colour i.

[Check that the above is indeed a valid $(n+1)$-colouring, i.e., that no pair of adjacent vertices receives the same colour.] This deals with the forward implication in (2).

For the reverse implication, suppose that G is $(n+1)$-colourable. Consider any valid $(n+1)$-colouring of the vertices of G. The vertices $v_0, v_1, \ldots, v_n$ form a clique, and so must acquire $n+1$ distinct colours; let these colours be referred to as $0, 1, \ldots, n$ respectively. In each pair of vertices $\{x_i, \overline{x}_i\}$, one of the vertices must have colour i and the other vertex colour 0. [Why?] The colouring of G defines an assignment of truth values to the variables of ϕ in the following way: for each variable x_i, set x_i to be true if vertex x_i has colour i, and false if vertex x_i has colour 0. We complete the analysis by showing that this truth assignment makes each clause of ϕ true (and hence ϕ itself true). Consider any vertex C_k in G. Since vertex C_k is adjacent to vertex v_0, it must receive a colour, i, other than 0. Thus, by the construction of the edge set of G, the clause C_k must contain a literal, either x_i or $\neg x_i$, which is true under the proposed assignment.

Note that (2) asserts that the function that maps the formula ϕ to the pair $(G, n+1)$ is a reduction from SAT to COLOURABILITY. The reader may readily check that the reduction is polynomial time. □

THEOREM 13.7 INTPROG *is NP-complete.*

PROOF. We exhibit a polynomial-time reduction from SAT to INTPROG. Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_r$ be a CNF Boolean formula in the variables $x_1, x_2, \ldots, x_n$. We construct a system of linear inequalities which has an integral solution if and only if ϕ is satisfiable. For $1 \leq i \leq r$ and $1 \leq j \leq n$ define

$$\alpha_{ij} = \begin{cases} 1, & \text{if the literal } x_j \text{ occurs in the clause } C_i; \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\beta_{ij} = \begin{cases} 1, & \text{if the literal } \neg x_j \text{ occurs in the clause } C_i; \\ 0, & \text{otherwise.} \end{cases}$$

Let $z_1, z_2, \ldots, z_n$ be integer variables and consider the following system of linear inequalities:

$$\begin{aligned} z_j &\geq 0, \quad \text{for } 1 \leq j \leq n; \\ z_j &\leq 1, \quad \text{for } 1 \leq j \leq n; \\ \sum_{j=1}^{n} (\alpha_{ij} z_j + \beta_{ij}(1 - z_j)) &\geq 1, \quad \text{for } 1 \leq i \leq r. \end{aligned}$$

It is straightforward to check that the system has a solution if and only if ϕ is satisfiable. Firstly, note that each variable is constrained to be either 0 or 1. Establish a correspondence between the Boolean variables x_i and the integer variables z_i, under which x_i is true if $z_i = 1$, and x_i is false if $z_i = 0$. Under this correspondence, satisfying assignments of ϕ are associated with solutions to the system of linear inequalities, and vice versa.

The reader should check that the given system of inequalities can be written in the form $A\mathbf{x} \leq \mathbf{b}$, which is the format specified in the definition of IntProg. The demonstration that IntProg $\in$ NP is omitted.[28] [It is instructive to attempt a proof that IntProg $\in$ NP; where does the 'obvious' approach fail?] □

Theorem 13.8 SubsetSum *is NP-complete.*

proof. We employ a reduction from IndSet. Let $G = (V, E)$ be an undirected graph and k a positive integer; taken together, G and k form a typical instance of IndSet. Let $V = \{v_0, \ldots, v_{n-1}\}$ and $E = \{e_0, \ldots, e_{m-1}\}$. Our reduction maps this instance of IndSet to an instance of SubsetSum which has underlying set $X = V \cup E$, sizes

$$\begin{aligned} s(e_i) &= 10^i, \quad \text{for } 0 \leq i \leq m-1; \\ s(v_i) &= 10^m + \sum_{e_j \ni v_i} 10^j, \quad \text{for } 0 \leq i \leq n-1 \end{aligned}$$

(the sum is over all edges e_j incident at vertex v_i), and integer goal

$$b = k \times 10^m + 10^{m-1} + 10^{m-2} + \cdots + 10^2 + 10 + 1.$$

Again, the appearance of v_i and e_i in two contexts is a slight abuse of notation, but one that does indicate an intentional correspondence between the two problem instances. We claim that G has an independent set of size k if and only if X contains a subset A such that $\sum_{x \in A} s(x) = b$.

First, suppose that $U \subseteq V$ is an independent set in G of size k. Let $F \subseteq E$ be the set of edges in G which are *not* incident at some vertex in the independent set U. We claim that $\sum_{x \in U \cup F} s(x) = b$. To see this, imagine that terms on the left hand side of this equation are arranged as a traditional sum in base 10. (There is no particular significance in the number 10, here or in the definition of the reduction; as we shall see, any base greater

[28] A proof that IntProg $\in$ NP can be found in J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, p. 338.

than three would do equally well.) Label the columns of the sum from zero, starting at the column of least significant digits. Note that the reduction sets up a correspondence between columns of the sum and edges of G: column i corresponds to edge e_i, for $0 \leq i \leq m-1$. Now each edge of G is *either* in the set F *or* has one endpoint in U. Thus, all columns other than column m contain a single 1. Also, since U is of size k, column m contains precisely k 1s.

Conversely, suppose there are subsets $U \subset V$, $F \subseteq E$ such that $\sum_{x \in U \cup F} s(x) = b$. Then U must be an independent set of size k. To see this, again consider the sum presented in base 10. No column other than the leftmost column (column m) may contain more than three 1s [why?]. Thus there can be no 'carries' between these columns, and the column sums must all be 1. This in turn implies that no edge of G can have both vertices in U, i.e., that U is an independent set. Also, since there must be exactly k 1s in column m, the set U must have size k.

Thus the function mapping (G, k) to (X, s, b) is indeed a reduction form IndSet to SubsetSum. It is easy to check that the reduction is computable in polynomial time. □

The problems discussed in this note represent a small selection from the many hundreds of problems which are now known to be NP-complete. Virtually all the search problems which have arisen in practice have been shown either to be in the class P or to be NP-complete. The way in which naturally occurring problems tend to cluster into a small number of complexity classes is one of the most intriguing phenomena in computer science, and is even now largely unexplained. However we must not assume that if a problem is in NP and does not seem to be in P then it must be NP complete. The following result, whose proof is beyond the scope of this course[29], is worth bearing in mind:

Theorem 13.9 *If* P $\neq$ NP *then there are languages in* NP *that are not in* P *and are not* NP *complete.*

In fact there are various natural problems whose status is still unknown, one of the most frequently cited is Graph Isomorphism (which is clearly in NP):

Instance: Two undirected graphs $G = (V, E)$ and $G' = (V', E')$.

Question: Are the graphs isomorphic?, i.e., is there a bijection $f : V \to V'$ such that for all vertices $u, v \in V$ we have $\{ u, v \}$ is an edge of G if and only if $\{ f(u), f(v) \}$ is an edge of G'? (In other words G and G' are really the same graph once we have relabeled things appropriately.)

What is puzzling is that there are many hundreds of known natural NP-complete problems but very few whose status is in the same limbo as that of Graph Isomorphism. This in turn can be used as a heuristic when trying to find the status of a problem known to be in NP but that is not immediately seen to be in P: if all attempts to prove that it is

[29]For a proof of the theorem see Ch. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, p. 330. This book is an excellent start for more advanced study.

NP-complete fail then it is worth putting a great deal of effort into developing a polynomial time algorithm for it.

The topic of resource-bounded computation introduced here can be pursued further in the CS4 *Computational Complexity* module.