

3515ICT: Theory of Computation

1 Computational complexity

1.1 The class \mathcal{P}

Briefly, \mathcal{P} is the class of problems that can be solved deterministically in polynomial time.

Class \mathcal{P} is important because (a) it is invariant over all reasonable models of computation and (b) practical problems in \mathcal{P} have efficient (low-degree polynomial) algorithms.

Important examples of problems in \mathcal{P} include context-free language recognition, primality testing, matrix multiplication and inversion, linear programming, finding shortest paths and minimal spanning trees in graphs, and finding convex hulls and Voronoi diagrams.

1.2 The class \mathcal{NP}

Briefly, \mathcal{NP} is the class of problems that can be verified deterministically in polynomial time.

Equivalently, \mathcal{NP} is the class of problems that can be solved nondeterministically in polynomial time.

The class \mathcal{NP} is also invariant over all reasonable models of computation.

Clearly, $\mathcal{P} \subseteq \mathcal{NP}$, but it is not known whether or not $\mathcal{P} = \mathcal{NP}$.

1.3 The class of \mathcal{NP} -complete problems

Def. Problem P_1 is *polynomially reducible* to problem P_2 ($P_1 \leq_P P_2$) if there exists a polynomial-time algorithm that transforms every instance I_1 of P_1 to an instance I_2 of P_2 such that the answer to I_1 is “yes” ($I_1 \in P_1$) if and only if the answer to I_2 is “yes” ($I_2 \in P_2$).

Note. The relation \leq_P is transitive: $P_1 \leq_P P_2$ and $P_2 \leq_P P_3$ implies $P_1 \leq_P P_3$.

This follows from the fact that the composition of two polynomial functions is another polynomial function.

Note. If $P_1 \leq_P P_2$ and $P_2 \in \mathcal{P}$, then $P_1 \in \mathcal{P}$.

To solve an instance of P_1 in polynomial time, first transform it to an instance of P_2 in polynomial time, and then use the polynomial-time decision procedure for P_2 .

Note. If $P_1 \leq_P P_2$ and $P_1 \notin \mathcal{P}$, then $P_2 \notin \mathcal{P}$.

This follows immediately from the previous note, and provides a way to prove that problems are intractable (not in \mathcal{P}).

If $P_1 \leq_P P_2$, we say that P_2 is *at least as hard* as P_1 .

Def. Problem P is \mathcal{NP} -complete if:

1. $P \in \mathcal{NP}$, and
2. every problem $P' \in \mathcal{NP}$ is polynomially reducible to P .

(We also say a problem P is \mathcal{NP} -hard if (only) condition 2 holds.)

That is, a problem is \mathcal{NP} -complete if it is in \mathcal{NP} and it is at least as hard as every problem in \mathcal{NP} .

Note. If P_1 is \mathcal{NP} -complete and $P_1 \leq_P P_2$, then P_2 is \mathcal{NP} -complete.

Note. If some \mathcal{NP} -complete problem is in \mathcal{P} , then $\mathcal{P} = \mathcal{NP}$.

Proof. By an above note, every problem in \mathcal{NP} is also in \mathcal{P} , and, by definition, $\mathcal{P} \subseteq \mathcal{NP}$.

This provides a potential way to settle the $\mathcal{P} = \mathcal{NP}$ question.

It shows that, if $\mathcal{P} \neq \mathcal{NP}$, then the set \mathcal{P} and the set of \mathcal{NP} -complete problems are disjoint. Moreover:

Ladner's Theorem. If $\mathcal{P} \neq \mathcal{NP}$, then there exists a problem in \mathcal{NP} that is not in \mathcal{P} and is not \mathcal{NP} -complete.

No natural example of such a problem is known. One candidate is GRAPH ISOMORPHISM (described below).

Exercise. Prove that, if $\mathcal{P} = \mathcal{NP}$, then every nontrivial problem in \mathcal{NP} is \mathcal{NP} -complete. (Here, problem P is trivial if $P = \emptyset$ or $P = \Sigma^*$.)

1.4 Satisfiability (SAT)

We present an important example of an \mathcal{NP} -complete problem.

A *boolean formula* is a formula constructed from a set of variables x, y, \dots using parentheses, the unary operator \neg and the binary operators \vee and \wedge , e.g., $x \wedge \neg(y \vee z)$. A (*truth*) *assignment* for a boolean formula F maps each variable in F to *true* (1) or *false* (0). The *value* of a boolean formula F under a truth assignment T for F is the result of replacing the variables in F by their corresponding truth values in T and applying the truth tables for the operators \neg , \vee and \wedge . A boolean formula F is *satisfiable* if there exists some truth assignment T for F such that the value of F under T is *true*.

Example. The formula $x \wedge \neg(y \vee z)$ is satisfiable ($x = 1$ and $y = z = 0$ is the unique satisfying assignment). The formula $x \wedge (\neg x \vee y) \wedge \neg y$ is not satisfiable. The formula $x \vee \neg y$ has three satisfying assignments.

Satisfiability (SAT) Is a given boolean formula satisfiable?

To represent an instance of SAT in a Turing machine, we code the variables in the form $x_1, x_{10}, x_{11}, \dots$. Then the length of an encoded formula is proportional to the number of variable occurrences in the formula.

Theorem. (Cook, Levin) Satisfiability is \mathcal{NP} -complete.

Proof outline. (See Sipser, Theorem 7.37, and Wikipedia entry on Cook's Theorem.) First, given a truth assignment for an formula, a deterministic Turing machine can check in polynomial time whether the value of the formula under the assignment is *true*. Second, for any language L in \mathcal{NP} , there exists a nondeterministic Turing machine M such that $L = L(M)$. That is, every computation of M on input w has the form $\alpha_0 \vdash \dots \vdash \alpha_k$, where k is at most a polynomial function of $|w|$. Every such (M, w) pair can be polynomially transformed into a boolean formula F such that F is satisfiable if and only if M accepts w in at most a polynomial number of steps. \square

We can now use the existence of this particular \mathcal{NP} -complete problem to prove other problems are \mathcal{NP} -complete.

1.5 Satisfiability of CNF formulas (CSAT)

Def. A *literal* is a variable (x) or the negation of a variable ($\neg x$ or \bar{x}). A *clause* is a disjunction (sum) of literals, e.g., $x \vee \bar{y} \vee z$ or $x + \bar{y} + z$. A boolean formula is in *conjunctive normal form* (CNF) if it is a conjunction (product) of clauses, e.g., $((x \vee \neg y) \wedge (\neg x \vee z))$ or $(x + \bar{y})(\bar{x} + z)$.

Satisfiability of CNF formulas (CSAT) Is a given boolean formula in CNF satisfiable?

Theorem. CSAT is \mathcal{NP} -complete.

Proof. CSAT is in \mathcal{NP} , as we can test whether an assignment satisfies a CNF formula or not in polynomial time.

We show $\text{SAT} \leq_P \text{CSAT}$. (Note: We can't simply transform an arbitrary boolean formula F to an equivalent CNF formula F' , because F' may be more than polynomially larger than F . E.g., the formula $a_1 b_1 + \dots + a_n b_n$ with $2n$ literals is equivalent to the CNF formula $(a_1 + \dots + a_n)(a_1 + a_2 + \dots + b_n) \dots (b_1 + \dots + b_n)$ with $n2^n$ literals.)

To transform an arbitrary boolean formula into CNF, first push all \neg operators inside all \vee and \wedge operators, e.g., $\neg((\neg(z + y))(\bar{x} + y)) \Rightarrow (x + y) + (x\bar{y})$, so that the only negated formulas are variables.

Then, recursively transform the resulting formula F to CNF as follows:

1. If F is a literal, return F .
2. If F is a conjunction $F_1 F_2$, recursively transform F_1 and F_2 to CNF formulas G_1 and G_2 , and return $G_1 G_2$.
3. If F is a disjunction $F_1 F_2$, recursively transform F_1 to $G_1 = g_1 \dots g_m$ and $G_2 = h_1 \dots h_n$, where the g_i and h_j are clauses. Let y be a new variable, i.e., one not occurring in G_1 or G_2 . Return $G = (y + g_1)(y + g_2) \dots (y + g_m)(\bar{y} + h_1) \dots (\bar{y} + h_n)$.

For example, if $F = x\bar{y} + \bar{x}(y + z)$, this algorithm should return $(u + x)(u + \bar{y})(\bar{u} + \bar{x})(\bar{u} + v + y)(\bar{u} + \bar{v} + z)$, where u and v are the new variables introduced.

Now, it is straightforward to show that the resulting formula G is only polynomially larger than the initial formula F and that G is satisfiable iff F is satisfiable.

Note. Cook actually proved that CSAT is \mathcal{NP} -complete, so the reduction $\text{SAT} \leq_P \text{CSAT}$ is not really required in the development of the theory.

A satisfiability algorithm for CNF formulas (Davis-Putnam).

```

sat(f) =
  if f is empty, return true;
  if f contains an empty clause, return false;
  choose a literal lit in f;
  if sat(apply(lit, f)), return true;
  else return sat(apply(comp(lit), f));

```

```

apply(lit, f) =
  remove each clause containing lit from f;
  remove each occurrence of comp(lit) from the remaining clauses in f;
  return f;

```

Here, $\text{comp}(\text{lit})$ is the complementary literal of lit .

Heuristics (Logemann, Loveland) for choosing a literal in an formula f are:

- (a) choose a pure literal, one whose complement does not occur in f .
- (b) choose the literal in a unit clause of f , a clause of length 1.
- (c) choose a literal in a shortest clause of f .

1.6 Satisfiability of 3CNF formulas (3SAT) and related problems

Def. A CNF formula is in 3CNF if every clause has exactly 3 literals.

Satisfiability of 3CNF formulas (3SAT) Is a given boolean formula in 3CNF satisfiable?

Theorem. 3SAT is \mathcal{NP} -complete.

Proof. See Sipser, Corollary 7.42. Reduction from CSAT.

Theorem. 2SAT is in \mathcal{P} .

Proof. Exercise. Define $G = (V, E)$ from a 2CNF formula ϕ as follows. Let V be the set of literals in ϕ and their complements. Let E consist of the edges $(\neg x, y)$ and $(\neg y, x)$ for each clause $(x \vee y)$ in ϕ . Then ϕ is unsatisfiable iff there is a variable x with paths from $\neg x$ to x and from x to $\neg x$. But determining whether there is a path from x to y in a directed graph can be done in polynomial time.

HORN SAT Is a given boolean formula in CNF with at most one positive literal in each clause satisfiable?

Theorem. HORN SAT is in \mathcal{P} .

Proof. Consider each clause as an implication (or constraint), and the formula as a logic program P . Compute the least model of P (satisfying the constraints) or, equivalently, the least fixed point of T_P (satisfying the constraints), in a bottom-up fashion.

MAX2SAT Given a boolean formula F in 2CNF and a positive integer k , is there a truth assignment that makes at least k clauses in F true?

Theorem. MAX2SAT is \mathcal{NP} -complete.

Proof. See Papadimitriou, Theorem 9.2. Reduction from 3SAT. For each clause $C = (x \vee y \vee z)$ in the 3CNF formula ϕ , let w be a new variable, and construct the 2CNF formula

$$x \wedge y \wedge z \wedge w \quad \wedge \quad (\bar{x} \vee \bar{y}) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{z} \vee \bar{w}) \quad \wedge \quad (x \vee \bar{w}) \wedge (y \vee \bar{w}) \wedge (z \vee \bar{w})$$

Then C is satisfiable iff at least 7 of these clauses are satisfiable. Repeat this for all m clauses in ϕ and let $k = 7m$.

NAESAT Given a boolean formula in 3CNF, is there a satisfying truth assignment that makes each clause have at least one true literal and at least one false literal?

Theorem. NAESAT is \mathcal{NP} -complete.

Proof. (Sipser, Problem 7.24) $3\text{SAT} \leq_P \text{NAESAT}$. Transform each instance ϕ of 3SAT as follows. Transform each clause $c_i = (x_1 \vee x_2 \vee c_3)$ of ϕ to $d_i = (y_1 \vee y_2 \vee y_3 \vee z)$, where each literal x_i has a corresponding literal y_i and z is a single new variable. Then define $y_i \neq z$ iff x_i is true. Finally, transform d_i to $(y_1 \vee y_2 \vee w) \wedge (\bar{w} \vee y_3 \vee z)$ as in the reduction $\text{CSAT} \leq_P 3\text{SAT}$.

1.7 Basic graph problems

CLIQUE Given a graph G and a positive integer k , does G have a complete subgraph of size k ?

INDEPENDENT SET (IS) Given a graph G and a positive integer k , does G have a set S of k vertices such no pair of vertices in S are adjacent?

VERTEX COVER (VC) (Also called **NODE COVER**.) Given a graph G and a positive integer k , does G have a set C of k vertices such that every edge in G is incident with a vertex in C ?

Theorem. CLIQUE is \mathcal{NP} -complete.

Proof. Use reduction from 3SAT given by Papadimitriou, Theorem 9.4, Figure 9-2. Vertex for each literal, edges between non-complementary literals in different clauses, $k =$ number of clauses.

Corollary 1. IS is \mathcal{NP} -complete.

Proof 1. Reduction from CLIQUE. G has a clique of size k iff the complement of G has an independent set of size k .

Proof 2. (Hopcroft *et al.*, Theorem 10.18) Direct reduction from 3SAT: vertex for each literal, edges between each pair of literals in same clause, edges between complementary literals in different clauses, $k =$ number of clauses.

Corollary 2. VC is \mathcal{NP} -complete.

Proof 1. Reduction from INDEPENDENT SET. I is an independent set of $G = (V, E)$ with k nodes iff $V \setminus I$ is a vertex cover with $n - k$ nodes. (Use contraposition in the proof.)

Proof 2. (Sipser, Theorem 7.4) Direct reduction from 3SAT. Vertex for each clause literal and for each variable and its complement. Edge between each pair of literals in the same clause (triangles), between each pair of complementary literals, and between each clause literal and corresponding literal. (Sipser, Figure 7.45) Let $k = m + 2l$, where m is the number of variables and l is the number of clauses.

SUBGRAPH ISOMORPHISM Given graphs G and H , does G contain a copy of H as a subgraph? *i.e.*, is there an injection from H to G that preserves edges?

Theorem. SUBGRAPH ISOMORPHISM is \mathcal{NP} -complete.

Proof. Exercise. Easy reduction from CLIQUE. Given (G, k) , let H be the complete graph on k vertices.

Note that GRAPH ISOMORPHISM is not known to be in \mathcal{P} or to be \mathcal{NP} -complete.

DOMINATING SET Given a graph G and a positive integer k , does $G = (V, E)$ have a set S of k vertices such every vertex in $V \setminus S$ is adjacent to a vertex in S .

Theorem. DOMINATING SET is \mathcal{NP} -complete.

Proof. Easy reduction from VERTEX COVER. Add a new vertex and two new edges making a triangle for each edge in G .

MAX CUT A *cut* in an undirected graph is a partition of the vertices V into two disjoint subsets S and T . The size of a cut is the number of edges that have one vertex in S and one in T . Then MAX CUT is the question: Given a graph G and an integer k , does G have a cut of size k or greater?

Theorem. MAX CUT is \mathcal{NP} -complete.

Proof. (Sipser, Problem 7.25) Reduction from NAESAT.

Note. The corresponding problem MIN CUT is in \mathcal{P} .

1.8 Colouring problems

3-COLOURING Given a graph G , can the vertices of G be coloured with 3 colours so that no two adjacent vertices have the same colour?

Theorem. 3-COLOURING is \mathcal{NP} -complete.

Proof. See Papadimitriou, Theorem 9.8; Sipser, Problem 7.27. Reduction from NAESAT.

Note that 2-COLOURING and 4-COLOURING are both in \mathcal{P} . (Every planar graph can be 4-coloured.)

1.9 Hamiltonian path problems

HAMILTONIAN PATH Given a directed graph G and two vertices s and t , is there a Hamiltonian path in G from s to t ? (A Hamiltonian path is a path that visits every vertex exactly once.)

Theorem. HAMILTONIAN PATH is \mathcal{NP} -complete.

Proof. See Sipser, Theorem 7.46. Reduction from 3SAT.

Clearly, HAMILTONIAN PATH (HAM PATH) is in \mathcal{NP} . To prove $3\text{SAT} \leq_P \text{HAM PATH}$, let

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_k \vee b_k \vee c_k).$$

First, represent each variable x_i with a diamond-like structure containing a doubly-linked list of nodes in the horizontal row. Connect the bottom of each diamond to the top of the next diamond.

Second, represent each clause $(a_j \vee b_j \vee c_j)$ of ϕ as a single node c_j .

Third, construct the doubly-linked list in each diamond to be of length $2k$, consisting of one pair for each clause c_j .

Fourth, if variable x_i occurs in clause c_j , add edges from the first member of the c_j pair in the i th diamond to c_j and a corresponding edge from c_j to the second member of this pair.

If literal \bar{x}_i occurs in clause c_j , add edges from the second member of the c_j pair in the i th diamond to c_j and a corresponding edge from c_j to the first member of this pair.

It is then straightforward to show that ϕ is satisfiable iff there is a Hamiltonian path from the top of the first diamond (s) to the bottom of the last diamond (t). Note that a Hamiltonian path cannot leave a list from one (clause) pair and return to another (clause) pair, because that would leave some nodes in the list unreachable.

Note. PATH (is there a path in G from s to t ?) is in \mathcal{P} . (And hence 2SAT is in \mathcal{P} .)

UNDIRECTED HAMILTONIAN PATH Given an undirected graph G and two vertices s and t , is there a Hamiltonian path in G from s to t ?

Theorem. UNDIRECTED HAMILTONIAN PATH is \mathcal{NP} -complete.

Proof. See Sipser, Theorem 7.55. Reduction from HAMILTONIAN PATH. Construct an undirected graph G' from G as follows. For vertices s and t , add new vertices s_{out} and t_{in} . For each other vertex u , add new vertices u_{in} , u_{mid} and u_{out} . For each edge (u, v) in G , add a new edge (u_{out}, v_{in}) in G' . Then G has a Hamiltonian path from s to t iff G' has a Hamiltonian path from s_{out} to t_{in} .

LONGEST PATH Given an undirected graph G , vertices s and t , and positive integer k , is there a simple path (*i.e.*, one containing no vertex more than once) in G from s to t of length at least k ?

Proof. Generalisation of UNDIRECTED HAMILTONIAN PATH, which is LONGEST PATH with $k = |V|$.

Note. The dual SHORTEST PATH problem is in \mathcal{P} .

TSP (Travelling salesman problem) Given a directed, weighted graph G , and a cost C , is there a Hamiltonian cycle in G whose cost is at most C ?

Theorem. TSP is \mathcal{NP} -complete.

Proof. See Papadimitriou, Corollary, p. 198. Simple reduction from HAMILTONIAN PATH. Let (G, s, t) be an instance of UNDIRECTED HAMILTONIAN PATH. First, add a new node u between t and s . Then G has a Hamiltonian path from s to t iff the new graph has a Hamiltonian circuit. Then, give each edge in the graph cost 1, and set C to be the number of vertices in the new graph. Then G has a Hamiltonian path from s to t iff the new weighted graph has a Hamiltonian cycle of cost at most C .

1.10 Sets and numbers

TRIPARTITE MATCHING Given three sets A , B and C , each containing n elements, and a ternary relation $R \subseteq A \times B \times C$, is there a set of n triples in R no two of which have a component in common? (*e.g.*, a man-woman-home application.)

Theorem. TRIPARTITE MATCHING is \mathcal{NP} -complete.

Proof. See Papadimitriou, Theorem 9.9. Reduction from 3SAT.

Corollary. EXACT COVER BY 3-SETS, SET COVERING and SET PACKING are \mathcal{NP} -complete.

Exercise. Prove SET COVERING (given a finite universe U , a family S of subsets of U and a positive integer k , is there a subset S' of S of size at most k whose union is U ?) is \mathcal{NP} -complete by reduction from VERTEX COVER.

SUBSET SUM Given a set S of integers and another integer K , is there a subset of S the sum of whose elements is K ?

Theorem. SUBSET SUM is \mathcal{NP} -complete.

Proof. See Sipser, Theorem 7.56. Reduction from 3SAT.

BIN PACKING Given a set S of n positive integers, and two more positive integers B (the number of bins) and C (the capacity), can S be partitioned into B subsets, each of which has sum at most C ?

Theorem. BIN PACKING is \mathcal{NP} -complete.

Proof. See Papadimitriou, Theorem 9.11. Reduction from TRIPARTITE MATCHING.

1.11 Other problems

MINESWEEPER CONSISTENCY Is a partially revealed $m \times n$ Minesweeper grid consistent with a possible placement of mines?

Theorem. MINESWEEPER CONSISTENCY is \mathcal{NP} -complete. Really!

Proof. See Richard Kaye's Minesweeper Pages, and Sipser, Problem 7.30.

1.12 Complements of \mathcal{NP} -complete problems

The class $\text{co}\mathcal{NP}$ is the set of languages that are complements of languages in \mathcal{NP} .

The complement of SAT asks whether a Boolean formula F is *false* for all truth assignments, *i.e.*, whether $\neg F$ is a tautology (*true* for all truth assignments). There is no obvious certificate for $\overline{\text{SAT}}$, so $\overline{\text{SAT}}$ is not obviously in \mathcal{NP} .

Similarly, the complement of TSP asks whether all Hamiltonian cycles in a graph G have cost greater than C . Again, there is no obvious certificate for $\overline{\text{TSP}}$, and $\overline{\text{TSP}}$ is not obviously in \mathcal{NP} .

However, we don't know whether or not $\text{co}\mathcal{NP}$ is different from \mathcal{NP} .

We do know that $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co}\mathcal{NP}$.

We *believe* that no \mathcal{NP} -complete problems are in $\text{co}\mathcal{NP}$ and that no complement of \mathcal{NP} -complete problems are in \mathcal{NP} .

However, if the complement of some \mathcal{NP} -complete problem is in \mathcal{NP} , then $\mathcal{NP} = \text{co}\mathcal{NP}$.

Also, clearly, if $\mathcal{P} = \mathcal{NP}$, then $\mathcal{P} = \mathcal{NP} = \text{co}\mathcal{NP}$.

1.13 Other complexity classes

Def. Let M be a TM that halts on all inputs. The *space complexity* of M is the function $f : \mathcal{N} \rightarrow \mathcal{N}$, where $f(n)$ is the maximum number of tape cells that M scans on any input of length n .

If M is a NTM such that all branches halt on all inputs, then its space complexity $f(n)$ is the maximum number of tape cells that M scans on any branch of its computation for any input of length n .

Let $f : \mathcal{N} \rightarrow \mathcal{N}$ be a function. Then $\text{SPACE}(f(n)) = \{ L \mid L \text{ is a language decided by some TM with } O(f(n)) \text{ space complexity} \}$. $\text{NSPACE}(f(n)) = \{ L \mid L \text{ is a language decided by some NTM with } O(f(n)) \text{ space complexity} \}$.

Space is more powerful than time because space can be reused, but time cannot.

Example. SAT can be solved in linear space, and is hence in PSPACE.

Example. The regular expression equivalence problem,

$$EQ_{\text{REX}} = \{ \langle R, S \rangle \mid R \text{ and } S \text{ are equivalent regular expressions} \}$$

is in PSPACE.

Savitch's Theorem. For any function $f : \mathcal{N} \rightarrow \mathcal{N}$, where $f(n) \geq n$,

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)).$$

Proof. Omitted.

Def. PSPACE is the class of languages that are decidable in polynomial space, *i.e.*,

$$\text{PSPACE} = \bigcup_{k \geq 0} \text{SPACE}(n^k).$$

We can define NPSPACE similarly.

By Savitch's theorem, PSPACE = NPSPACE.

Theorem. $\mathcal{P} \subseteq \mathcal{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXP} \subseteq \text{NEXP} \subseteq \text{EXPSpace}$.

Here, EXP (resp., NEXP) refers to exponential *time* (resp., nondeterministic exponential *time*).

Proof. Omitted.

We know that $\mathcal{P} \subset \text{EXP}$, but we don't know which of the intermediate inclusions is strict!

We also know that $\mathcal{P} = \mathcal{NP}$ implies $\text{EXP} = \text{NEXP}$.

Def. A language B is *PSPACE-complete* if

1. B is in PSPACE, and
2. For every language A in PSPACE, $A \leq_P B$.

QUANTIFIED BOOLEAN FORMULAS (QBF) A (totally) quantified boolean formula is a boolean formula in which every variable is existentially or universally quantified. An example of such an formula is

$$F = \forall x \exists y [(x \vee y) \wedge (\bar{x} \vee \bar{y})].$$

The QBF problem is: Is a given quantified boolean formula *true*?

(Sipser calls this the *TQBF* problem.)

Theorem QUANTIFIED BOOLEAN FORMULAS (QBF) is PSPACE-complete.

Proof. Omitted.

QBF remains PSPACE-complete, even when restricted to formulas in CNF (resp., DNF), and when further restricted to have at most three literals per disjunct (resp., conjunct).

Note that the QBF problem has a two-person game flavour. See Sipser, Example 8.10 and Theorem 8.11.

GENERALIZED GEOGRAPHY (GG) Let G be a directed graph with vertex s . The current vertex is s . Two players alternate in choosing a new vertex that is connected to the current vertex, and making it the current vertex. The first player who cannot move loses. The GG problem is: Given (G, s) , is there a winning strategy for the first player?

Theorem GENERALIZED GEOGRAPHY (GG) is PSPACE-complete.

Proof. Omitted. Reduction from QBF.

REGULAR FORMULA TOTALITY Given a regular expression E over alphabet Σ , does $L(E) = \Sigma^*$?

Theorem. REGULAR EXPRESSION TOTALITY is PSPACE-complete.

Proof. Omitted.

Corollary. REGULAR EXPRESSION EQUIVALENCE is PSPACE-complete.

Proof. Trivial polynomial-time reduction from REGULAR EXPRESSION TOTALITY.

Note. Determining whether two *star-free* regular expressions are equivalent is $\text{co}\mathcal{NP}$ -complete.

See Papadimitriou, Note 20.2.13, for discussion of these results.

Inclusion (resp., equivalence) of XML DTDs is PSPACE-complete.

Generalisations of some familiar problems (*e.g.*, Rush Hour, Sokoban) are PSPACE-complete.

Generalisations of some familiar two-person games (*e.g.*, Draughts, Chess, Go, Hex, Shannon Switching Game) are also PSPACE-complete or harder.

Of course, it's (remotely) possible that $\mathcal{P} = \text{PSPACE}$, so we still don't have an example of any problem that is definitely outside \mathcal{P} .

BOUNDED TURING MACHINE ACCEPTANCE Given a TM M , an input x and a positive integer k , does M halt on x in at most k steps.

Theorem. BOUNDED TURING MACHINE ACCEPTANCE is EXP-complete (and hence outside \mathcal{P}).

Proof outline. Simulating M on x for at most k steps takes $O(k)$ time. But the input k is represented in $\log k$ space, so the time complexity is exponential in the size of the input.

OK, perhaps that example is a bit artificial.

EXTENDED REGULAR EXPRESSION TOTALITY Given a regular expression E , that may use an intersection operator (the extension), over alphabet Σ , does $L(E) = \Sigma^*$?

Theorem. EXTENDED REGULAR EXPRESSION TOTALITY is EXP-complete, and hence outside \mathcal{P} .

Proof. Omitted.

Corollary. EXTENDED REGULAR EXPRESSION EQUIVALENCE is EXP-complete, and hence outside \mathcal{P} .

Proof. Trivial polynomial-time reduction from EXTENDED REGULAR EXPRESSION TOTALITY.

Exercise. Find a class of regular languages that can be represented in exponentially less space using extended regular expressions instead of (normal) regular expressions.

Some variants of Draughts, Chess and Go (which allow indefinite repetition) are EXP-complete, and hence outside \mathcal{P} .

Satisfiability of propositional dynamic logic has exponential-time complexity, and is hence outside \mathcal{P} . This logic makes statements about the effects of propositional programs, *e.g.*, the formula “**after(while E do A end; if E then B end, $\neg E$)**” is *true* for any values of A , B and E .

Some decidable theories of arithmetic over the integers (*e.g.*, Presburger arithmetic, $\text{Th}(\mathbb{N}, +)$) or the reals, with different sets of operators, have been proved to have (at least) exponential complexity, and are hence outside \mathcal{P} . In fact, Presburger arithmetic has been proved to have doubly exponential complexity.

1.14 Managing \mathcal{NP} -complete problems

1. Seek restrictions that make the problem tractable. *e.g.*, 2SAT is in \mathcal{P} ; 2-COLOURING is in \mathcal{P} .
2. Seek parameterisations that make the problem tractable. *e.g.*, VERTEX COVER is fixed-parameter tractable, *i.e.*, there exists an algorithm that solves VERTEX COVER with time complexity $1.4^k k^2 + c|G|$ (where c is a constant independent of k). However, DOMINATING SET is not fixed-parameter tractable.
3. Seek efficient approximation algorithms. *i.e.*, seek polynomial-time algorithms that solve optimisation problems to within some bounded error ϵ . Such algorithms exist for many NP-complete problems (good). However, the exponents are often large multiples of $1/\epsilon$ (bad).
4. Seek efficient probabilistic algorithms. Very useful in practice.
5. Seek efficient parallel algorithms. Less useful in practice.
6. (If really desperate) seek efficient quantum algorithms. Unlikely (?) to be useful in practice.

References

- D. Harel with Y. Feldman, *Algorithmics: The Spirit of Computing, Third Edition*, Addison-Wesley, 2004.
- J.E. Hopcroft, R. Motwani and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation, Second Edition*, Addison-Wesley, 2001.
- C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.
- M. Sipser, *Introduction to the Theory of Computation, Second Edition*, Course Technology, 2006.