

## 24 NP-Complete Problems

In this section, we discuss a number of NP-complete problems, with the goal to develop a feeling for what hard problems look like. Recognizing hard problems is an important aspect of a reliable judgement for the difficulty of a problem and the most promising approach to a solution. Of course, for NP-complete problems, it seems futile to work toward polynomial-time algorithms and instead we would focus on finding approximations or circumventing the problems altogether. We begin with a result on different ways to write boolean formulas.

**Reduction to 3-satisfiability.** We call a boolean variable or its negation a *literal*. The *conjunctive normal form* is a sequence of clauses connected by  $\wedge$ s, and each *clause* is a sequence of literals connected by  $\vee$ s. A formula is in *3-CNF* if it is in conjunctive normal form and each clause consists of three literals. It turns out that deciding the satisfiability of a boolean formula in 3-CNF is no easier than for a general boolean formula. Define  $3\text{-SAT} = \{\varphi \in \text{SAT} \mid \varphi \text{ is in 3-CNF}\}$ . We prove the above claim by reducing SAT to 3-SAT.

**SATISFIABILITY LEMMA.**  $\text{SAT} \leq_P 3\text{-SAT}$ .

**PROOF.** We take a boolean formula  $\varphi$  and transform it into 3-CNF in three steps.

**Step 1.** Think of  $\varphi$  as an expression and represent it as a binary tree. Each node is an operation that gets the input from its two children and forwards the output to its parent. Introduce a new variable for the output and define a new formula  $\varphi'$  for each node, relating the two input edges with the one output edge. Figure 110 shows the tree representation of the formula  $\varphi = (x_1 \implies x_2) \iff (x_2 \vee \neg x_1)$ . The new formula is

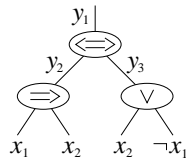


Figure 110: The tree representation of the formula  $\varphi$ . Incidentally,  $\varphi$  is a tautology, which means it is satisfied by every truth assignment. Equivalently,  $\neg\varphi$  is not satisfiable.

$$\begin{aligned}\varphi' &= (y_2 \iff (x_1 \implies x_2)) \\ &\quad \wedge (y_3 \iff (x_2 \vee \neg x_1)) \\ &\quad \wedge (y_1 \iff (y_2 \iff y_3)) \wedge y_1.\end{aligned}$$

It should be clear that there is a satisfying assignment for  $\varphi$  iff there is one for  $\varphi'$ .

**Step 2.** Convert each clause into disjunctive normal form. The most mechanical way uses the truth table for each clause, as illustrated in Table 6. Each clause

| $y_2$ | $x_1$ | $x_2$ | $y_2 \iff (x_1 \implies x_2)$ | prohibited                                 |
|-------|-------|-------|-------------------------------|--|
| 0     | 0     | 0     | 0                             | $\neg y_2 \wedge \neg x_1 \wedge \neg x_2$ |
| 0     | 0     | 1     | 0                             | $\neg y_2 \wedge \neg x_1 \wedge x_2$      |
| 0     | 1     | 0     | 1                             |  |
| 0     | 1     | 1     | 0                             | $\neg y_2 \wedge x_1 \wedge x_2$           |
| 1     | 0     | 0     | 1                             |  |
| 1     | 0     | 1     | 1                             |  |
| 1     | 1     | 0     | 0                             | $y_2 \wedge x_1 \wedge \neg x_2$           |
| 1     | 1     | 1     | 1                             |  |

Table 6: Conversion of a clause into a disjunction of conjunctions of at most three literals each.

has at most three literals. For example, the negation of  $y_2 \iff (x_1 \implies x_2)$  is equivalent to the disjunction of the conjunctions in the rightmost column. It follows that  $y_2 \iff (x_1 \implies x_2)$  is equivalent to the negation of that disjunction, which by de Morgan's law is  $(y_2 \vee x_1 \vee x_2) \wedge (y_2 \vee x_1 \vee \neg x_2) \wedge (y_2 \vee \neg x_1 \vee \neg x_2) \wedge (\neg y_2 \vee \neg x_1 \vee x_2)$ .

**Step 3.** The clauses with fewer than three literals can be expanded by adding new variables. For example  $a \vee b$  is expanded to  $(a \vee b \vee p) \wedge (a \vee b \vee \neg p)$  and  $(a)$  is expanded to  $(a \vee p \vee q) \wedge (a \vee p \vee \neg q) \wedge (a \vee \neg p \vee q) \wedge (a \vee \neg p \vee \neg q)$ .

Each step takes only polynomial time. At the end, we get an equivalent formula in 3-conjunctive normal form.  $\square$

We note that clauses of length three are necessary to make the satisfiability problem hard. Indeed, there is a polynomial-time algorithm that decides the satisfiability of a formula in 2-CNF.

**NP-completeness proofs.** Using polynomial-time reductions, we can show fairly mechanically that problems are NP-complete, if they are. A key property here is the transitivity of  $\leq_P$ , that is, if  $L' \leq_P L_1$  and  $L_1 \leq_P L_2$  then  $L' \leq_P L_2$ , as can be seen by composing the two polynomial-time computable functions to get a third one.

**REDUCTION LEMMA.** Let  $L_1, L_2 \subseteq \{0, 1\}^*$  and assume  $L_1 \leq_P L_2$ . If  $L_1$  is NP-hard and  $L_2 \in \text{NP}$  then  $L_2 \in \text{NPC}$ .

A generic NP-completeness proof thus follows the steps outline below.

Step 1. Prove that  $L_2 \in \text{NP}$ .

Step 2. Select a known NP-hard problem,  $L_1$ , and find a polynomial-time computable function,  $f$ , with  $x \in L_1$  iff  $f(x) \in L_2$ .

This is what we did for  $L_2 = 3\text{-SAT}$  and  $L_1 = \text{SAT}$ . Therefore  $3\text{-SAT} \in \text{NPC}$ . Currently, there are thousands of problems known to be NP-complete. This is often con-

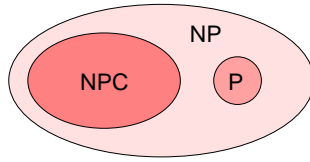


Figure 111: Possible relation between P, NPC, and NP.

sidered evidence that  $P \neq \text{NP}$ , which can be the case only if  $P \cap \text{NPC} = \emptyset$ , as drawn in Figure 111.

**Cliques and independent sets.** There are many NP-complete problems on graphs. A typical such problem asks for the largest complete subgraph. Define a *clique* in an undirected graph  $G = (V, E)$  as a subgraph  $(W, F)$  with  $F = \binom{W}{2}$ . Given  $G$  and an integer  $k$ , the CLIQUE problem asks whether or not there is a clique of  $k$  or more vertices.

**CLAIM.** CLIQUE  $\in$  NPC.

**PROOF.** Given  $k$  vertices in  $G$ , we can verify in polynomial time whether or not they form a complete graph. Thus CLIQUE  $\in$  NP. To prove property (2), we show that  $3\text{-SAT} \leq_P \text{CLIQUE}$ . Let  $\varphi$  be a boolean formula in 3-CNF consisting of  $k$  clauses. We construct a graph as follows:

- (i) each clause is replaced by three vertices;
- (ii) two vertices are connected by an edge if they do not belong to the same clause and they are not negations of each other.

In a satisfying truth assignment, there is at least one true literal in each clause. The true literals form a clique. Conversely, a clique of  $k$  or more vertices covers all clauses and thus implies a satisfying truth assignment.  $\square$

It is easy to decide in time  $O(k^2 n^{k+2})$  whether or not a graph of  $n$  vertices has a clique of size  $k$ . If  $k$  is a constant, the running time of this algorithm is polynomial in  $n$ . For the CLIQUE problem to be NP-complete it is therefore essential that  $k$  be a variable that can be arbitrarily large. We use the NP-completeness of finding large cliques to prove the NP-completeness of large sets of pairwise non-adjacent vertices. Let  $G = (V, E)$  be an undirected graph. A subset  $W \subseteq V$  is *independent* if none of the vertices in  $W$  are adjacent or, equivalently, if  $E \cap \binom{W}{2} = \emptyset$ . Given  $G$  and an integer  $k$ , the INDEPENDENT SET problem asks whether or not there is an independent set of  $k$  or more vertices.

**CLAIM.** INDEPENDENT SET  $\in$  NPC.

**PROOF.** It is easy to verify that there is an independent set of size  $k$ : just guess a subset of  $k$  vertices and verify that no two are adjacent.

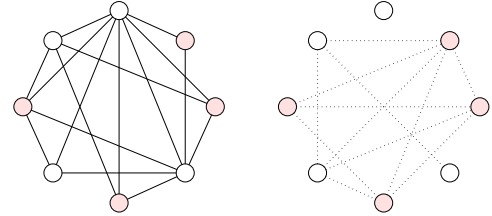


Figure 112: The four shaded vertices form an independent set in the graph on the left and a clique in the complement graph on the right.

We complete the proof by reducing the CLIQUE to the INDEPENDENT SET problem. As illustrated in Figure 112,  $W \subseteq V$  is independent iff  $W$  defines a clique in the complement graph,  $\overline{G} = (V, \binom{V}{2} - E)$ . To prove CLIQUE  $\leq_P$  INDEPENDENT SET, we transform an instance  $H, k$  of the CLIQUE problem to the instance  $G = \overline{H}, k$  of the INDEPENDENT SET problem.  $G$  has an independent set of size  $k$  or larger iff  $H$  has a clique of size  $k$  or larger.  $\square$

**Various NP-complete graph problems.** We now describe a few NP-complete problems for graphs without proving that they are indeed NP-complete. Let  $G = (V, E)$  be an undirected graph with  $n$  vertices and  $k$  a positive integer, as before. The following problems defined for  $G$  and  $k$  are NP-complete.

An  $\ell$ -coloring of  $G$  is a function  $\chi : V \rightarrow [\ell]$  with  $\chi(u) \neq \chi(v)$  whenever  $u$  and  $v$  are adjacent. The CHROMATIC NUMBER problem asks whether or not  $G$  has an  $\ell$ -coloring with  $\ell \leq k$ . The problem remains NP-complete for fixed  $k \geq 3$ . For  $k = 2$ , the CHROMATIC NUMBER problem asks whether or not  $G$  is bipartite, for which there is a polynomial-time algorithm.

The *bandwidth* of  $G$  is the minimum  $\ell$  such that there is a bijection  $\beta : V \rightarrow [n]$  with  $|\beta(u) - \beta(v)| \leq \ell$  for all adjacent vertices  $u$  and  $v$ . The BANDWIDTH problem asks whether or not the bandwidth of  $G$  is  $k$  or less. The problem arises in linear algebra, where we permute rows and columns of a matrix to move all non-zero elements of a square matrix as close to the diagonal as possible. For example, if the graph is a simple path then the bandwidth is 1, as can be seen in Figure 113. We can transform the

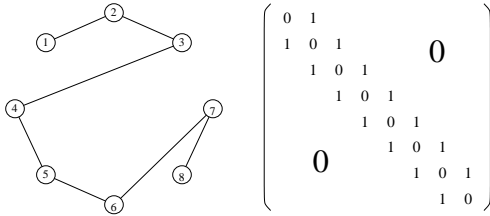


Figure 113: Simple path and adjacency matrix with rows and columns ordered along the path.

adjacency matrix of  $G$  such that all non-zero diagonals are at most the bandwidth of  $G$  away from the main diagonal.

Assume now that the graph  $G$  is complete,  $E = \binom{V}{2}$ , and that each edge,  $uv$ , has a positive integer weight,  $w(uv)$ . The TRAVELING SALESMAN problem asks whether there is a permutation  $u_0, u_1, \dots, u_{n-1}$  of the vertices such that the sum of edges connecting contiguous vertices (and the last vertex to the first) is  $k$  or less,

$$\sum_{i=0}^{n-1} w(u_i u_{i+1}) \leq k,$$

where indices are taken modulo  $n$ . The problem remains NP-complete if  $w : E \rightarrow \{1, 2\}$  (reduction to HAMILTONIAN CYCLE problem), and also if the vertices are points in the plane and the weight of an edge is the Euclidean distance between the two endpoints.

**Set systems.** Simple graphs are set systems in which the sets contain only two elements. We now list a few NP-complete problems for more general set systems. Letting

$V$  be a finite set,  $C \subseteq 2^V$  a set system, and  $k$  a positive integer, the following problems are NP-complete.

The PACKING problem asks whether or not  $C$  has  $k$  or more mutually disjoint sets. The problem remains NP-complete if no set in  $C$  contains more than three elements, and there is a polynomial-time algorithm if every set contains two elements. In the latter case, the set system is a graph and a maximum packing is a maximum matching.

The COVERING problem asks whether or not  $C$  has  $k$  or fewer subsets whose union is  $V$ . The problem remains NP-complete if no set in  $C$  contains more than three elements, and there is a polynomial-time algorithm if every set contains two elements. In the latter case, the set system is a graph and the minimum cover can be constructed in polynomial time from a maximum matching.

Suppose every element  $v \in V$  has a positive integer weight,  $w(v)$ . The PARTITION problem asks whether there is a subset  $U \subseteq V$  with

$$\sum_{u \in U} w(u) = \sum_{v \in V - U} w(v).$$

The problem remains NP-complete if we require that  $U$  and  $V - U$  have the same number of elements.