Análisis y Diseño de Algoritmos

Miguel Ángel Romero González

Jueves 18 de Octubre de 2018

Pontificia Universidad Javeriana de Cali
Ingeniería de Sistemas y Computación

NP-complete problems [1]

Class P

$$P = \{L \subseteq \{0,1\}^* : \quad \text{there exists an algorithm } A \text{ that decides } L \text{ in polynomial time}\}.$$

In fact, P is also the class of languages that can be accepted in polynomial time.

$P = \{L : L \text{ is accepted by a polynomial-time algorithm}\}$.

Class NP

$$L = \{x \in \{0,1\}^* : \quad \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\}.$$

We say that algorithm A **verifies** language L in **polynomial time**.

Class co-NP

We can define the complexity class co-NP as the set of languages L such that $\bar{L} \in NP$.

Reducibility

A language L_1 is **polynomial-time reducible** to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in L_1$ if and only if $f(x) \in L_2$.

We call the function f the **reduction function**, and a polynomial-time algorithm F that computes f is called a **reduction algorithm**.

Reducibility

If $L_1, L_2 \subseteq \{0,1\}^*$ are languages such that $L_1 \leq_P L_2$, then $L_2 \in P$ implies $L_1 \in P$.

That is, if $L_1 \leq_P L_2$, then L_1 is not more than a polynomial factor harder than L_2.

NP-complete

A language $L \subseteq \{0,1\}^*$ is **NP-complete** if

1. $L \in NP$, and
2. $L' \leq_P L$ for every $L' \in NP$.

If a language L satisfies property 2, but not necessarily property 1, we say that L is **NP-hard**.

SAT problem

The problem of determining whether a boolean formula is satisfiable has the historical honor of being the first problem ever shown to be NP-complete.

SAT problem

We formulate the **(formula) satisfiability** problem in terms of the language SAT as follows. An instance of SAT is a boolean formula ϕ composed of

1. n boolean variables: $x_1, x_2, \ldots, x_n$;
2. m boolean connectives: AND, OR, NOT, implication, if and only if; and
3. parentheses.

SAT problem

A **truth assignment** for a boolean formula ϕ is a set of values for the variables of ϕ, and a **satisfying assignment** is a truth assignment that causes it to evaluate to 1. A formula with a satisfying assignment is a **satisfiable** formula.

SAT problem

The satisfiability problem asks whether a given boolean formula is satisfiable; in formal-language terms,

$SAT = \{\langle\phi\rangle : \phi \text{ is a satisfiable boolean formula}\}$.

3-CNF-SAT problem

We define 3-CNF satisfiability using the following terms. A **literal** in a boolean formula is an occurrence of a variable or its negation. A boolean formula is in **conjunctive normal form**, or **CNF**, if it is expressed as an AND of **clauses**, each of which is the OR of one or more literals. A boolean formula is in **3-conjunctive normal form**, or **3-CNF**, if each clause has exactly three distinct literals.

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

3-CNF-SAT problem

In 3-CNF-SAT, we are asked whether a given boolean formula ϕ in 3-CNF is satisfiable.

Supposed: Satisfiability of boolean formulas in 3-conjunctive normal form is NP-complete.

The clique problem

A **clique** in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. In other words, a clique is a complete subgraph of G. The **size** of a clique is the number of vertices it contains.

The clique problem

The **clique problem** is the optimization problem of finding a clique of maximum size in a graph. As a decision problem, we ask simply whether a clique of a given size k exists in the graph. The formal definition is

$$\text{CLIQUE} = \{\langle G, k\rangle : G \text{ is a graph with a clique of size } k\}.$$

The clique problem

A naive algorithm for determining whether a graph $G = (V, E)$ with $|V|$ vertices has a clique of size k is to list all k-subsets of V, and check each one to see whether it forms a clique.

As one might suspect, an efficient algorithm for the clique problem is unlikely to exist.

The clique problem

Theorem
The clique problem is NP-complete.

The clique problem

Proof
To show that CLIQUE $\in$ NP, for a given graph $G = (V, E)$, we use the set $V' \subseteq V$ of vertices in the clique as a certificate for G. Checking whether V' is a clique can be accomplished in polynomial time by checking whether, for each pair $u, v \in V'$, the edge (u, v) belongs to E.

The clique problem

Proof
We next prove that 3-CNF-SAT $\leq_P$ CLIQUE, which shows that the clique problem is NP-hard.

The clique problem

Proof
The reduction algorithm begins with an instance of 3-CNF-SAT. Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ be a boolean formula in 3-CNF with k clauses.

For $r = 1, 2, \ldots, k$, each clause C_r has exactly three distinct literals l_1^r, l_2^r, and l_3^r.

We shall construct a graph G such that ϕ is satisfiable if and only if G has a clique of size k.

The clique problem

Proof
For each clause $C_r = (l_1^r \wedge l_2^r \wedge l_3^r)$ in ϕ, we place a triple of vertices v_1^r, v_2^r, and v_3^r into V. We put an edge between two vertices v_i^r and v_j^s if both of the following hold:

- v_i^r and v_j^s are in different triples, that is, $r \neq s$, and
- their corresponding literals are **consistent**, that is, l_i^r is not the negation of l_j^s.

¿Preguntas?

Bibliografía


T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson.
Introduction to Algorithms.
McGraw-Hill Higher Education, 2nd edition, 2001.

¡Gracias!