NP-Completeness

- Polynomial-time algorithms are tractable.
  - low-order poly time is fast
  - invariant under many models of computation (RAM, TM, parallel)
  - invariant under +, *, and composition
- Algorithms that require more than polynomial time are intractable
- NP-complete problems are probably intractable
- NP-complete problems are candidates for approximation

Problems

- An abstract problem $Q$ is a binary relation on a set $I$ of problem instances and a set $S$ of problem solutions.
  - SHORTEST-PATH finds a shortest path between two given vertices in an unweighted, undirected graph $G = (V,E)$.
  - An instance is a triple consisting of a graph and two vertices.
  - A solution is a sequence of vertices in the graph, with the empty sequence denoting no path.
  - A given instance may have more than one solution.

- A decision problem is an abstract problem with a yes or no answer.
  - Given two vertices in an unweighted, undirected graph $G = (V,E)$ and a non-negative integer $k$, is there a path of length at most $k$?
  - An instance is a quadruple consisting of a graph, two vertices, and a non-negative integer.
  - A solution is either 0 (no) or 1 (yes)
- NP-Completeness theory uses decision problems.

Accepting and Deciding

- Algorithm $A$ accepts input $x$ if the algorithm outputs $A(x) = 1$. A rejects $x$ if $A(x) = 0$.
- The language accepted by $A$ is the set of all inputs accepted by $A$.
- A language $L$ is accepted in poly-time by $A$ if, for any input $x$ in $L$ of length $n$, $A$ accepts $x$ in time $O(n^k)$ for some constant $k$.
- An algorithm may crash or loop forever and therefore fail to accept or to reject an input.
- A language is decided by $A$ if every input is either accepted or rejected.
- A language is decided in poly-time by $A$ if, for any input $x$ of length $n$, $A$ decides $x$ in time $O(n^k)$ for some constant $k$.
- To accept, $A$ need only look at strings in $L$; to decide, $A$ must look at every string in $Σ^*$.
- There are problems for which an accepting algorithm, but no decision algorithm, exists.
- Complexity class $P$ is the set of languages that can be decided in polynomial time.
- Theorem:
  $P = \{L: L \text{ is decided by a poly-time algorithm}\}$

Hamiltonian-Cycle Problem

- undirected graph $G=(V,E)$
- an hamiltonian cycle is a simple cycle that contains each vertex in $V$.
- hundred-year old problem
- the vertices and edges of a dodecahedron form a Hamiltonian graph.
- no bipartite graph with an odd number of vertices is Hamiltonian.
- The Hamiltonian-cycle problem, "does graph $G$ have a Hamiltonian cycle?" can be defined as a language:
  $\text{ham-cycle} = \{<G>: G \text{ is a Hamiltonian graph}\}$
  where $<G>$ is the standard encoding of graph $G$.

Deciding vs. Verifying

- Deciding if a graph is Hamiltonian
  - no known poly-time algorithm (in fact, the problem is NP-complete)
  - naive algorithm: test each permutation of vertices to see if it is a simple cycle.
  - complexity: Use adjacency matrix as encoding.
    Number of vertices is $m = Ω(\sqrt{n})$, where $n$ is the length of the encoding.
    $Ω(m!) = Ω(\sqrt{n}!) = Ω(2\sqrt{n})$, which is not poly-time.
- Verifying that a graph is Hamiltonian
  - given a list of vertices purported to form a Hamiltonian cycle.
    - this list is called a certificate.
    - there is a poly-time algorithm $A$ that can be used to check if the list is a Ham-cycle: check that the list is a permutation of the graph vertices, then check that consecutive vertices in the list are connected by graph edges.
    - algorithm $A$ verifies ham-cycle in poly time.

The complexity class NP is the class of languages that can be verified by a poly-time algorithm.

More Examples

- Is positive integer $n$ composite? A certificate for $n$ could be a list of its factors.
**Graph Coloring**
Can the vertices of graph $G$ be colored with $k$ colors so that no two adjacent vertices have the same color? A certificate could be a list of the colors associated with each vertex.

**Satisfiability**
Can the variables in a Boolean expression be assigned values true or false so the expression is true? A certificate could be a list of truth values associated with each variable.

**Traveling Salesperson Problem (TSP)**
- **Optimization problem:** given a weighted graph, find a minimum-weight Hamiltonian cycle
- **Decision problem:** given a weighted graph and integer $k$, is there a Hamiltonian circuit of total weight at most $k$?

**Reducibility**
Transforming one problem to another

- **Language $L_1$ is polynomial-time reducible to language $L_2$, written $L_1 \leq_P L_2$, if there is a poly-time computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ so that, for all $x \in \{0,1\}^*$, $x \in L_1$ if and only if $f(x) \in L_2$.
- **Example:** the Hamiltonian cycle problem is poly-time reducible to the Travelling Salesperson Problem. Idea: convert an instance of the ham-cycle problem to an 'equivalent' instance of TSP. Let $G=(V,E)$ be a graph with $n$ nodes (ham-cycle problem). Construct an instance of TSP as follows: Let $H=(V,V \times V)$ be a complete graph on the vertices of $G$. The edge weights $c(u,v)$ are 1 if the edge $(u,v)$ is in $E$ and 2 otherwise. The bound $k$ is $n$. Then a Hamiltonian cycle in $G$ is a tour in $H$ with cost $n$. If there are no Hamiltonian cycles in $G$, any tour in $H$ must cost at least $n+1$. So $G$ is in ham-cycle if and only if $f(G)$ is in TSP.

**NP-Completeness**
Show problem is in NP and that some NP-complete problem is transformable to it.

**Assume:** The 3CNF-SAT problem is NP-complete. A logic formula in 3-CNF is the AND of clauses, each of which is the OR of three distinct variables, for example, $(a \lor b \lor c) \land (d \lor a \lor \neg c)$

**Clique Problem**

- **undirected graph $G=(V,E)$
- an $k$-clique of graph $G$ is a complete subgraph of $G$ with $k$ vertices
- **Clique problem:** Does an undirected graph have a clique of size $k$?
- **in NP:** guess clique, verify it in $O(n^2)$ time by checking if every pair of vertices in the clique is in $E$.
- **NP-complete:** The 3-CNF-SAT problem is polynomially transformable to the Clique Problem.
- **create a vertex for each appearance of a variable. Connect vertices if they are in different clauses and not negatives of each other. Then the formula is satisfiable if and only if the graph has a clique of size $k$, where $k$ is the number of clauses in the formula.

**Vertex-Cover Problem**

- **undirected graph $G=(V,E)$
- A vertex cover of $G$ is a subset $S$ of $V$ with the property that each edge of $G$ is incident on some vertex in $S$.
- **Vertex Cover Problem:** Does an undirected graph have a vertex cover of size $K$?
- **in NP:** guess vertex cover, test endpoints of each edge in $E$ to see if at least one is in the vertex cover.
- **NP-complete:** transform Clique Problem to a Vertex Cover problem.
• Idea: use graph complement. The complement of an undirected graph $G$ has the same set of vertices as $G$ and exactly the edges that are not in $G$.

• Given a graph $G$ with $n$ vertices for the clique problem, construct its complement. Then $G$ has a clique of size $k$ if and only if its complement has a vertex cover of size $n-k$.

**Approximation Algorithms**

Poly-time algorithms with performance bounds

• Bounds
  – $C$ is the cost of the approximate solution
  – $C^*$ is the cost of an optimal solution
  – ratio bound is $\rho(n)$ if
    \[
    \max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).
    \]
  – relative error bound is $\epsilon(n)$ if
    \[
    \left|\frac{C - C^*}{C^*}\right| \leq \epsilon(n).
    \]

• Vertex Cover can be approximated with a ratio bound of 2

• If the weights in TSP obey the triangle inequality, TSP can be approximated with a ratio bound of 2 using Prim’s MST algorithm.

**Summary**

• $P$ is the set of decision problems that can be *solved* in polynomial time

• NP is the set of decision problems that can be *verified* in polynomial time

• $P$ is in NP, but it is unknown if $P=NP$.

• Problem $P_1$ is polynomial-time *reducible* to problem $P_2$ if there is a poly-time computable function $f$ with the property that, the result of $P_1$ on input $x$ is the same as the result of $P_2$ on input $f(x)$.

• a problem $L$ is NP-complete if it is in NP and if every problem is NP can be poly-time reduced to $L$. (decision problems only)

• a problem $L$ is NP-hard if if every problem is NP can be poly-time reduced to $L$. (optimization problems allowed)

• a problem $L$ can be shown to be NP-hard by showing that one problem known to be NP-complete can be poly-time reduced to $L$. To show $L$ is NP-complete, also show it is in NP.

• Solutions to NP-complete problems can be *approximated*. 