Tools of AI

Hard Computational Problems (4)

Marcin Sydow

Web Mining Lab/PJII T
Topics covered by this lecture:

- Examples of proofs of NP-completeness
- Pseudo-polynomial Algorithms, Strong NP-completeness
- Example of Exponential Algorithm
More examples of NP-complete problems

- CLIQUE
- VERTEX COVER
- INDEPENDENT-SET
- SET-COVER
- SET-PACKING
- HAM-CYCLE
- TSP
- SUBSET-SUM
CLIQUE optimisation problem: given an undirected graph $G = (V, E)$ find the maximum *clique* in $G$. (a maximisation problem)

(A *clique* in $G$ is a full subgraph of $G$)

The *decision-version* of the CLIQUE problem: CLIQUE: given an undirected graph $G = (V, E)$ and $k \in \mathbb{N} \setminus \{0\}$ is there a clique of size $k$ in $G$?

If $k$ is a constant, the brute-force algorithm (checking all the possible $k$-element subsets of $V$ whether they form a $k$-clique) formally has the polynomial complexity $\Theta(k^2(n!/k!(n-k)!))$, however if $k$ is close to $|V|/2$ it becomes exponential.
CLIQUE (decision version) is NP-complete

(Reduction from 3-CNF-SAT)

Obviously, CLIQUE \( \in \text{NP} \) (verifying if a given subset of a graph is a clique can be done in polynomial time)

Reduction: given a 3-CNF-SAT instance \( \phi \) consisting of \( k \) OR-clauses construct a \( 3k \)-vertex graph \( G \) that has a clique iff the corresponding 3-CNF formula is satisfiable.

Idea: For each OR-clause of \( \phi \) create a triple of 3 vertices representing its literals. The edges are added only between any vertex and all consistent\(^1\) vertices in other triples. Now, the literals that evaluate to 1 in the truth-assignment of \( \phi \) constitute a clique of size \( k \) in \( G \). On the other hand, any \( k - \text{clique} \) in \( G \) corresponds to a truth-assignment of \( \phi \). The construction can be done in polynomial time.

\(^1\)two literals are consistent except the situations one is a negation of another
VERTEX-COVER problem

VERTEX-COVER optimisation problem: given a graph $G = (V, E)$ find a minimum vertex cover in $G$. (a minimisation problem)

A vertex cover is such a subset of vertices $V' \subseteq V$ that each edge in $E$ is adjacent with some vertex from $V'$ (is “covered”).

VERTEX-COVER (decision version): given a graph $G = (V, E)$ and a natural positive number $k$ answer if there is a vertex cover in $G$ of size $k$. 
VERTEX-COVER (decision version) is NP-complete

(reduction from CLIQUE)

Whether a given set of vertices ("certificate") is a vertex cover can be easily verified in polynomial time, so VERTEX-COVER $\in$ NP

Let's explain that it is also NP-hard:
Given an instance $C$ of k-clique (decision) problem, construct a graph that has a vertex cover if and only if $C$ has a k-element clique.

Idea: An undirected graph $G = (V, E)$ has a k-clique if and only if a complement graph $\bar{G} = (V, \bar{E})$ has a vertex cover of size $|V| - k$

(A complement of an undirected graph $G = (V, E)$ has the same set of vertices $V$ and has an edge $(u, v)$, $u, v \in V$ only if $(u, v) \notin E.$)
INDEPENDENT-SET problem

Given a graph $G = (V, E)$, find a maximum subset $V' \subseteq V$ such that no pair of vertices from $V'$ are adjacent in $G$. (maximisation problem)

INDEPENDENT-SET decision version: Given a graph $G = (V, E)$ and positive natural $k$ answer whether there exists an independent set of size at least $k$

INDEPENDENT-SET (decision version) is NP-complete

Idea (reduction from VERTEX-COVER): a set of $k$ vertices $V' \subseteq V$ is independent in $G$ if and only if $V \setminus V'$ is a ((|V| − $k$)-element) VERTEX-COVER in $G$. 
SET-COVER (decision) problem

Given a set $U$ of $n$ elements and a family $S = \{S_1, ..., S_m\}$ of subsets of $U$ and positive natural $k$, answer whether there exists a subfamily of $S$ of at most $k$ subsets such that their union is equal to ("covers") $U$

Possible interpretation: select the minimum set of people that have all desired skills in total, etc.

SET-COVER is NP-complete

Idea (reduction from VERTEX-COVER): given an instance $(G = (V, E), k)$ of VERTEX-COVER we translate it to a SET-COVER instance as follows. Each vertex $v \in V$ corresponds to the set of edges from $E$ that are incident to $v$, and $U$ is the set of all edges $E$. 
SET-PACKING (decision) problem

Given a set $U$ of $n$ elements and a family $S = \{S_1, \ldots, S_m\}$ of subsets of $U$ and positive natural $k$, answer whether there exists a subfamily of $S$ of at least $k$ subsets such that no pair of subsets intersect

Possible interpretation: subsets may correspond to processes that need non-sharable resources from a set $U$ of all resources of the system; can we run at least $k$ processes in parallel?

SET-PACKING is NP-complete

Idea: reduction from INDEPENDENT-SET, analogously as SET-COVER can be reduced from VERTEX-COVER.
A *hamiltonian cycle* in a graph is a cycle that uses each vertex exactly once\(^2\).

**HAM-CYCLE (decision) problem:** given a graph \(G = (V, E)\) answer whether there exists a hamiltonian cycle in it.

**HAM-CYCLE** is NP-complete (it can be reduced from VERTEX-COVER, for example, see Cormen et al. for a proof).

\(^2\)A cycle with no repeated vertices is called a *simple* cycle.
Travelling Salesman Problem (TSP)

TSP optimisation problem: Given a graph $G = (V, E)$ with non-negative weights on edges find a hamiltonian cycle $C \subseteq E$ in $G$ with minimum weight (defined as the sum of weights of the edges in $C$). (minimisation problem)

TSP decision version:
Given a graph $G = (V, E)$ with weights on edges and $k \in N$ answer whether there exists a hamiltonian cycle in $G$ with weight of at most $k$.

TSP (decision) problem is NP-complete.

Idea (reduction from HAM-CYCLE): given an instance $G = (V, E)$ of HAM-CYCLE, construct an instance of TSP as $G' = (V, E')$ such that $E' = \{(i, j) : i, j \in V \land i \neq j\}$ and weight for each edge $(i, j)$ defined as $w(i, j) = \lceil (i, j) \notin E \rceil$ (assume self loops exist in $G'$ and have weights of 1). The corresponding instance of TSP is then whether there exists a hamiltonian cycle of weight (at most) 0 in $G'$. 
Reduction types

- Karp Reduction: single reference to the reduction target
- Cook Reduction: multiple reference, more general (also called Turing reduction, i.e. a reduction that is polynomial iff the target sub-routine is polynomial)

Karp reduction (showing $Y \leq_P X$):

- prove $X \in NP$
- choose $Y$ that is NPC
- take an arbitrary instance $s_Y$ of $Y$ and show how to construct in polynomial time an instance $s_X$ of $X$ so that:
  - if $s_Y$ is “yes” instance of $Y$ then $s_X$ is “yes” instance of $X$
  - if $s_X$ is “yes” instance of $X$ then $s_Y$ is “yes” instance of $Y$
Yet another distinction among reduction types

(after Garey and Johnson’s classic textbook: “Computers and Intractability”, 1979)

- Restriction (the simplest, similar to the “Karp” type), e.g.:
  - VERTEX-COVER $\leq_P$ HITTING-SET (read as “VERTEX-COVER is a restriction of HITTING-SET”)
  - EXACT-3-COVER $\leq_P$ MIN-COVER

- Local Replacement (“medium complexity”) e.g.: SAT $\leq_P$ 3-SAT

- Component Design (the most complex) e.g.:
  - 3-SAT $\leq_P$ HAMILTON-CYCLE
  - 3-SAT $\leq_P$ 3-DIM-MATCHING
  - 3-SAT $\leq_P$ 3-COLORING
  - 3-DIM-MATCHING $\leq_P$ SUBSET-SUM
Graph k-Coloring

Assign each vertex a color (1 of k) so that neighbours have different colors
(resource allocation, map coloring, etc.)

is 2-coloring NP-complete? (it is equivalent to checking whether a graph is bi-partite what can be done in polynomial time. How? With BFS in $O(|V| + |E|)$ time

3-coloring is NP-complete (reduction from 3-SAT, for example)

for $k \geq 3$ k-coloring is NP-complete (reduction from 3-coloring: original graph + $k - 3$-element clique connected to all original nodes)
Why NP-completeness is useful for practical algorithm-designers?

Assume, we want to solve a new problem. Now assume that somebody proved it to be an NP-complete problem.

Does it make sense to look for a fast algorithm solving it? Not really, because unless $P = NP$ there is no such algorithm.

In this situation it is possibly much better to invest the effort in a different way than looking for a fast exact solution.
For a problem proved to be NP-complete there are the following alternatives for proceeding (since finding an exact fast algorithm is unlikely):

- trying to design exponential-time algorithm that is as fast as possible
- focus on special cases, and find fast algorithms for them
- work towards a fast approximation algorithm, that does not solve the problem exactly, but you can prove some bounds on the solution quality
- (if you cannot do the above) use one of the fast heuristics to approximately solve the problem, without guarantees on the quality (this is possibly the least “ambitious” approach, but sometimes necessary)
- device a randomised algorithm, that is expected to be fast
Example

Let's consider another decision problem:

**PARTITION:** A finite set of $n$ items, each item $a \in A$ has weight $w(a) \in \mathbb{Z}^+$.

**OUTPUT:** Can $A$ be partitioned into 2 parts of identical weights? (i.e. does there exist $A' \subseteq A$, such that $\sum_{a \in A'} w(a) = B/2$, where $B = \sum_{a \in A} w(a)$)

**On one hand:** PARTITION is NP-complete ($3\text{SAT} \leq_P 3\text{-DIM-MATCHING} \leq_P \text{PARTITION}$)

**On the other hand:** There exists an algorithm that solves PARTITION in time $\Theta(nB)$

Does this mean we have just found the first polynomial-time algorithm for an NP-complete problem!?
Dynamic-programming algorithm for PARTITION

If $B$ is odd answer no.

Otherwise, for integers $0 < i \leq n$, $0 \leq j \leq B/2$ let $t(i,j)$ be "true"/"false" according to the following statement: "there is a subset of $\{a_1, a_2, \ldots, a_i\}$ for which the sum of weights is exactly $j$".

The $t(i,j)$ table is filled row by row, starting with $t(1,j) = true$ iff $j = 0$ or $w(a_1) = j$. For $i > 1$ we set $t(i,j) = [t(i-1,j) = true \lor (w(a_i) \leq j \land t(i-1,j-w(a_i))]$. The answer is “yes” iff $t(n,B/2) = true$.

Time complexity of this (correct!) algorithm is: $\Theta(nB)$.

Is it a polynomial of the data size? Not really.
Pseudo-polynomial algorithms

Since the numbers in the task are represented in binary form ("reasonable encoding" assumption), $\Theta(nB)$ can be actually exponential function of the data size.

The algorithm is polynomial only if the numerical values in the input are small enough (i.e. polynomial) in the data size.

Such algorithms are called pseudo-polynomial algorithms.
Let’s consider an instance $i$ of a given problem.

Let $\text{length}(i)$ and $\text{max}(i)$ specify integer functions (interpreted as “data size” and “maximum numeric value” in the input).

(Two pairs of functions $(\text{length}, \text{max})$ and $(\text{length}', \text{max}')$ are \textit{polynomially related} iff $\text{length}(i) \leq p'(\text{length}'(i))$, $\text{length}'(i) \leq p(\text{length}(i))$ and $\text{max}(i) \leq q'(\text{max}'(i), \text{length}'(i))$, $\text{max}'(i) \leq q(\text{max}(i), \text{length}(i))$ for all instances $i$ and some polynomials $p, p', q, q'$.)

An algorithm is \textit{pseudo-polynomial} iff it is bounded by a polynomial of $\text{length}()$ and $\text{max}()$.

A problem is called \textit{number problem} if there exists no polynomial $p$ such that $\text{max}(i) \leq p(\text{length}(i))$ for all instances $i$. 

Strong NP-completeness

We call the problem **strongly NP-complete** if it contains a subproblem that is NP-complete and satisfies polynomial bound on \( \text{Max} \).

PARTITION is not strongly NP-complete (as there exists a pseudo-polynomial algorithm for it)

Observations:

- If problem is NP-complete and is not a number problem then it cannot be solved by a pseudo-polynomial algorithms *unless* \( P \neq NP \)
- If a problem is strongly NP-complete, then it cannot be solved by a pseudo-polynomial algorithms *unless* \( P \neq NP \)
Example: Vertex Cover

Vertex Cover, decision version (VC):

Given a graph $G = (V, E)$, where $|V| = n$ and $k \in N$, is there a vertex cover of size at most $k$? (i.e. such a set of vertices $S \subseteq V$, $|S| \leq k$ that each edge $e \in E$ has at least one end in $S$).

VC is NP-complete (SAT -> 3-SAT -> IS (independent set) -> VC), so that no polynomial algorithm is likely to exist.
However, if $k$ is fixed and small (e.g. $k = 3$), the method of checking all $3^k$ possible subsets of size $k$ has complexity $O(kn \cdot n^k)$ that is polynomial (of $n$).

Notice: even for relatively small values of $n, k$ this polynomial algorithm is impractical: e.g. $n = 1000, k = 10$ would take more than the age of the Universe on a PC.

Interestingly, there exists an exponential algorithm that would be faster for small values of $n, k$!

Observations:
if $G$ has at most $k$-element vertex cover, then:

- $|E| \leq kd$, where $d$ is maximum degree of a node
- $|E| \leq k(n - 1)$

\(^3\)In general, checking all potential solutions is called brute force method.
Exponential Algorithm for VC

Assume $e = (u, v) \in E$. $G$ has at most $k$-element vertex cover iff at least one of the graphs $G \setminus \{u\}$ or $G \setminus \{v\}$ has a vertex cover of size at most $k - 1$

If $|E| = 0$ then answer ‘‘yes’’, if $|E| > kn$ then answer ‘‘no’’
Else, take any edge $e = (u, v)$
   recursively check if either $G \setminus \{u\}$ or $G \setminus \{v\}$ has vertex cover $T$ of size $k - 1$
   if neither has, then answer ‘‘no’’
   else $T + \{u\}$ or $T + \{v\}$ is $k$-element vertex cover of $G$

Time complexity of the above algorithm is $O(2^k kn)$

Thus, for our previous example ($n = 1000, k = 10$) the algorithm will find the solution very quickly

-------------------------------------------------

$^4$all the edges incident to a node are also removed
Tools of AI
Marcin Sydow
Examples of NP-complete problems
Strong NP-completeness
Exponential Algorithms

Time complexity analysis of the algorithm

Explanation: The recursion tree has height of $k$, thus the number of recursive calls is bounded by $2^{k+1}$. Each recursive call (except the leaves) takes at most $O(kn)$ time.

Proof (by induction on $k$): $T(n, k) = O(2^k kn)$. Assume $c \in N$ is a constant:

$T(n, 1) \leq cn$

$T(n, k) \leq 2T(n, k - 1) + ckn$

Assume the thesis is true for $k - 1$, then:

$T(n, k) \leq 2T(n - 1, k - 1) + ckn \leq 2c \cdot 2^{k-1} (k - 1)n + ckn = c2^k kn - c2^k n + ckn \leq c \cdot 2^k kn$

Of course, this algorithm is not practical for higher values of $k$ (as it is exponential in $k$)
Literature

- Cormen et al. “Introduction to Algorithms”, chapters 34,35 (3rd edition)
- Kleinberg, Tardos “Algorithm Design”, chapters 8,10,11
- Garey, Johnson “Computers and Intractability” (1979, difficult to get nowadays)
- Papadimitriou “Computational Complexity” (first chapters) (more advanced textbook)
Thank you for attention