CS430

LECTURE NOTE NO.8

NP-COMPLETE PROBLEMS

We first define some complexity terminology:
Time Complexity– number of steps required by a machine in the worst case. The time complexity is expressed as a function of the input size i.e the length of the string representing the input.

The constants are typically absorbed by using the O notation.

A TIME COMPLEXITY class, $TIME(t(n))$ is defined as: $TIME(t(n)) = \{qL|L$ is a language decided by a $O(t(n))$ time Turing Machine.

The class P

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

Examples of problems in $\mathcal{P}$.

Problem 1.:The Path problem: Given a directed graph, determine if there is a path between s and t in the graph. This can be solved by a depth first search. Depth First search can be implemented in $O(n)$ steps assuming immediate knowledge of nodes adjacent to a particular node.

Problem 2.: Triangulation a polygon. A dynamic programming approach was described before. A table is used to find the cost of triangulation sub-polygons. This required $O(n^3)$ steps.

The class $\mathcal{NP}$: This is the class of problem solvable in polynomial time on a non-deterministic Turing Machine.

$NTIME(t(n)) = \{L|L$ is a languages that can be recognized by a non-deterministic Turing Machine in $O(t(n))$.

$\mathcal{NP} = \cup_k NTIME(n^k)$

Problems in NP:

CLIQUE problem: Given a graph $G = (V, E)$ is there a complete subgraph of size k.

There is an alternate view of $\mathcal{NP}$ in terms of verification algorithms.

A $VERIFIER$ for a language A is an algorithm V where V accepts the pair $(w, c), w \in A$ and c is a certificate for w.

The notion of a certificate is a proof of membership of w in A.

A language A is polynomial time verifiable if it has a verifier that requires time polynomial in w.

Example 1:

The Hamiltonion path problem can be verified in polynomial time where w is an instance of the problem (i.e. a graph) and c is a path.

A path in a graph can be easily verified to be hamiltonion or not.

Example 2:

Consider the problem:

$COMPOSITES = \{x|x = pq$ for some integeres $p, q > 1\}$.

Compositeness is also polynomial time verifiable.

Given the certificates p and q it is easy to determine if w is a composed of p and q. Thus this problem is polynomial time verifiable.

Lemma 0.1. *NP is the class of languages that have polynomial time verifiers.*

Proof*: Convert a polytime ND-Turing Machine to a polynomial time verifier.

Given the input (w, c) to the verifier, run the NDTM with input w choosing c to be the choices in the execution of the non-deterministic machine. Accept if the NDTM accepts.

Coversely, given a verifier the following machine accepts the string in non-deterministic polynomial time. On input w, simply guess a certificate c and run the verifier on (w, c). Accept if V accepts.

We will restrict our attention to decision problems. This is without loss of generality since decision and optimization problems are inter-related.

1 Polynomial Time Reducibility and NP-Completeness.

A problem A is polynomially time reducible to B, written as $A \leq_P B$, if there exists a polynomial time computable function $f : \Sigma^* \to \Sigma^*$, such that for every w, w has a yes answer $\iff$ $f(w) \in B$.

The reduction is called a polynomial time reduction.

Theorem 1.1. *If $A \leq_B B$ and $B \in P$, then $A \in P$.*

A language B is NP-Complete if it satisfies
1. B is in NP, and
2. every A in NP is polynomial time redcuible to B.

Consequences

If B is NP-Complete and $B \in P$ then $P = NP$.
If B is NP-Complete and $B \leq_P C$ for C in NP then C is NP-Complete.

The first problem to be shown NP-Complete is the SATISFIABILITY PROBLEM: A Boolean formula is an expression involving Boolean variables and operations. For example:

$$\phi = (x' \wedge y) \vee (x \wedge z')$$

is an example of a Boolean formula. A Boolean formula is satisfiable if some truth assignment to the variable makes the formula evaluate to true.

The satisfiability problem:

$$SAT = \{\phi | \phi \text{ is satisfiable}\}$$

Cook-Levin Theorem:

$$SAT \in P \iff P = NP$$

Theorem 1.2. *SAT is NP-Complete*

Proof*: To show this we need to design a boolean formula that simulates the working of a Turing Machine.

Given a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, f_A, q_F)$ and an inpute $w = w_1 w_2 w_3 \dots w_n$ we would like to construct a boolean formula where sets of clauses represent the configuration sequence $C_0, C_1 \dots C_f$ which leads to the acceptance of an input w to the machine M:

Let M decide w in N^k steps. The configurations can be represented in a $n^k \times n^k$ table.

The following conditions are required to be satisfied:

(i) Each cell of the table represents a valid symbol of a configuration.

(ii) The first row is the starting configuration.

(iii) The final row is an accepting configuration.

(iv) the $i$the row of the table follows from the $i-1$st row.

Formula to represent (i)

We have a variable $x(i,j,k)$ which is true when the (i,j) element of the table contains the kth symbol in γ.

$$\phi_1 = \bigwedge_{i,j}[(\bigvee_k x(i,j,k) \wedge (\bigwedge_{s.t.\ s \neq t} (\overline{x(i,j,k)} \vee \overline{x(i,j,t)})]$$

Formula to represent (ii)

$\phi_I = x(1,1,\#) \wedge x(1,2,q_0) \wedge x(1,3,w_1)$
$\ldots x(1,n,w_n) \wedge x(1,n+1,<bl>) \ldots x(1,n^k,<bl>$

Formula to represent (iii)

$$\phi_A = \bigvee_{i,j} x(i,j,q_A)$$

Formula to represent (iv)

$$\phi_M = \bigwedge_{i,j} \phi(i,j)$$

where $\phi(i,j)$ represents that the (i,j)th symbol is legal.

$\phi(i,j) = \bigvee_{(a,b,c,d,e,f)}(x(i,j,a) \wedge x(i,j+1,b) \wedge x(i,j+2,c)$
$\wedge x(i+1,j,d) \wedge x(i+1,j+1,e) \wedge x(i+1,j+2,e))$

where (a,b,c,d,e,f) is a valid 6-tuple representing the productions.

A valid 6-tuple follows the production rules:

Each row may not contain more than one state.

The second row follows from the first as follows:

(i) if a is a state q and $\delta(q,b) \rightarrow (q',x,R)$ then $d = x, e = q', f = c$.

(ii) if a is a state q and $\delta(q,b) \rightarrow (q',x,L)$ then d can be anything, $e = x$ and $f = c$

(iii) if b is a state q and $\delta(q,c) \rightarrow (q',x,R)$ then $d = a, e = x, f = q'$.

(iv) if b is a state q and $\delta(q,c) \rightarrow (q',x,L)$ then $d = q', e = a, f = x$.

(v) if c is a state q then $d = a$ and e and f can be anything.

(vi) if no states are present then $d = a$, $e = b$ and $f$can be either c or a state.

Size of the formula generated. The largest size formula is ϕ_M which is of size $O(n^{2k})$.

Thus the formula and the reduction is polynomial.

REDUCTIONS

We show a number of problems to be NP-Complete in this section:

CLIQUE is NP-Complete

This is achieved by a reduction from 3-SAT. 3-SAT is polynomial time reducible to the CLIQUE Problem

Given a Boolean formula composed of k clauses

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \ldots (a_k \vee b_k \vee c_k)$$

we create a graph such that for each occurrence of a literal in the clause there is a vertex. i.e. $V = \{x_{ij} |$ the ith clause contains the jth literal $\}$. and the edge set connects two vertices such that the vertices do not correspond to the same clause or to negated literals, i.e. $E = \{(x_{ij}, x_{kl}) | i \neq k$ and the literal corresponding to x_{ik} is not a negation of the one corresponding to $x_{kl}\}$.

It follows that the formula ϕ is satisfiable iff the graph has a clique of size k.

Vertex-Cover is NP-Complete

The Vertex-Cover Problem: Given a graph $G = (V, E)$ find a set of vertices S. of size k such that all the edges of the graph have at least one edge incident to a vertex in S.

The NP-Hardness is shown by a reduction from 3-Sat. Given ϕ a 3-SAT formula, a graph is created as follows:
Vertex Set:
Type 1: (i) A vertex is created for each variable and it complement, v_i and $\overline{v_i}$.
Type 2: (ii) A vertex is also created for each clause-literal pair.

Edge Set:
(i) An edge exists between a variable and its complement ($v_i \overline{v_i}$).
(ii) A clique exists between all literals in a clause-literal pair
(iii) Edges exist between the variable and its occurences in the clauses or between its complement and their occurences in the clauses.

A satisfying assignment generates a variable in the vertex cover according to the variable or its complement being true. Each of the clauses will have the two other literals in the vertex cover. This gives a vertex cover of size $n + 2m$, where n is the number of variables and m the number of clauses.

A vertex cover must have two vertices from each clause– to cover the clique edges. And one vertex from the variable-complement pair. This is a satisfying assignment provided no other vertex is chosen in the vertex cover.

Hamiltonion Cycle Problem:

Given a directed graph $G = (V, E)$ the hamiltionion cycle problem asks for directed cycle that traverses each vertex once and exactly once.

We reduce the vertex cover problem to this problem.

We introduce k special vertices, $s_1, s_2 \dots s_k$, which have edges to a gadget G_i which is sequence of vertices $u_1, u_1', u_2, u_2', \dots u_l$ representing a vertex u in the graph which has l edges incident to it. Each edge-gadget comprises a pair of vertices, say u_i and u_i' and represents the ith edge (u, v) incident to u. The edge-gadget is connected to the corresponding edge-gadget for the other vertex incident to the edge (shown in the figure) and allows the path to visit the other gadget and return to this gadget. In this graph a hamiltonion cycle can visit only visit k gadgets starting from the first vertex.

If there is a vertex cover of size k then picking the gadgets corresponding to the vertices in the vertex cover for traversal by the hamiltonion cycle. Since all the edges are incident to the vertices of this vertex cover every edge-gadget corresponding to vertices not in the cover can also be visited.

Conversely if there is a hamiltonion path then it can traverse only k gadgets (because of $v_1, v_2 \dots v_k$) and thus all the edge-gadgets not in the traversed gadgets, are traversed via a corresponding edge-gadget in a traversed gadget. Thus the traversed gadget correspond to vertices which cover all the edges.

It can be verified that the reduction is of polynomial size.

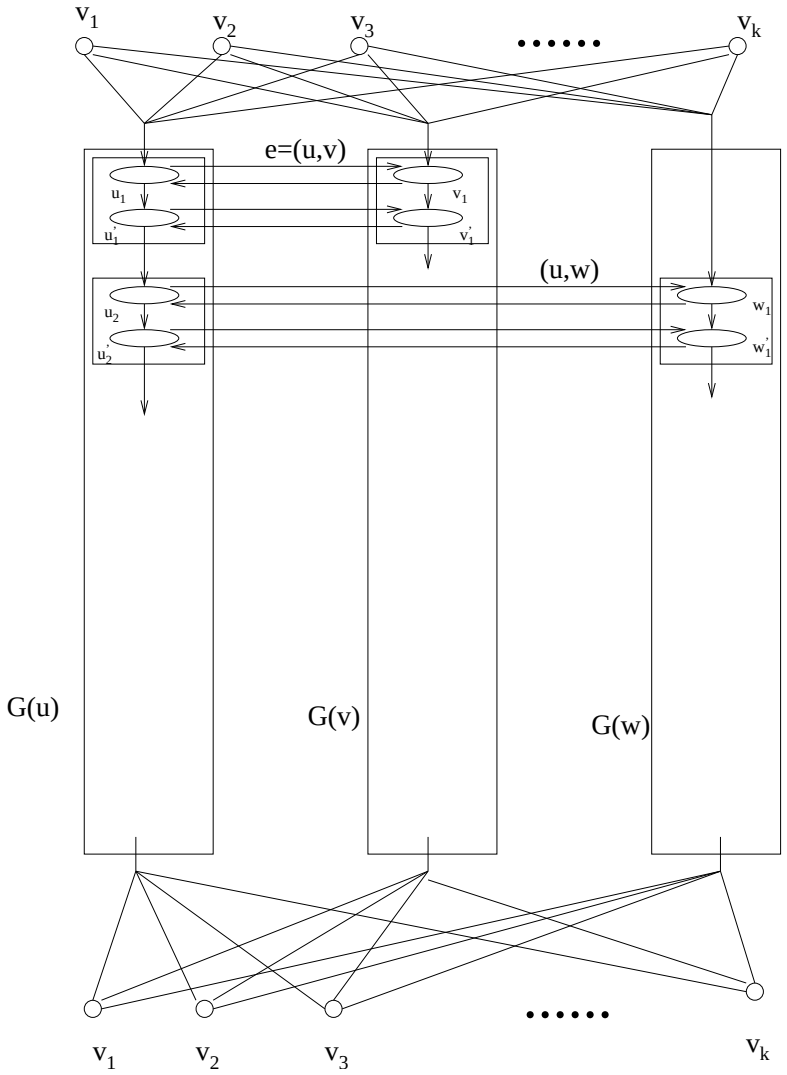

Figure 1: Hamiltonion Cycle is NP-Complete

2 Space Complexity:

If M is a non-deterministic Turing Machine where all branches halt on all inputs, we define its space complexity $f(n)$ to be the maximum number of tape cells that M scans on any branch of its computation for an input of length n.

For deterministic machine the space complexity is simply $f(n)$, the number of tape cells used on an input of length n.

So $NSPACE(f(n)) = \{L|L$ is a language decided by an $O(f(n))$ space nondeterministic Turing machine $\}$.

And $DSPACE((f(n)) = \{LL$ is a language decided by an $O(f(n)$ space deterministic machine $\}$.

PSPACE Complete Problems:

PSPACE is the class of languages decidable in polynomial space on a deterministic Turing Machine.

A Language B is PSPACE complete if

(i) B is in PSPACE

(ii) every language in PSPACE is polynomial time reducible to B.

Quantified Boolean Formula is PSPACE Complete:

The following formula simulates a PSPACE Turing Machine:

We want to specify a quantified Boolean Formula which simulates a correct sequence of configurations:

$\phi_{c_1,c_2,t} = \exists c[\phi_{c_1,c,t/2} \wedge \phi_{c,c_2,t/2}]$

This formula will require space corresponding to exponential time. To reduce the size we modify it to the following formula:

$\phi_{c_1,c_2,t} = \exists c \forall (c_3, c_4) \in (c_1, c), (c, c_1)[\phi_{c_3,c_4,t/2}]$.

3 APPROXIMATIONS

Consider a minimization problem for which the corresponding decision problem has been shown to be NP-Hard. Let $OPT(I)$ be value of optimum solution.

Further, consider an algorithm A which approximates the solution to the minimization problem. If the value is $vA(I)$, then $vA(I)/OPT(I)$ is the approximation ratio that we are able to achieve.

TSP

We consider approximating the TSP. We consider the TSP in graphs which satisfy triangle inequality. The following algorithm is proposed:

1. Find the MST of the graph.
2. Double the edges and construct an eulerian tour of the graph E.
3. Shortcut the eulerain tour to eliminate duplication of vertices and obatin a tour A.

What is the cost of this tour?

$$cost(MST) \leq OPT(TSP)$$

.

$$Cost(E) \leq 2cost(MST)$$

.

$$Cost(A) \leq Cost(E)$$

Thus we get an approximation ratio of 2.

Can this be extended to graphs where triangle inequality is not satisfied? No. In fact the problem of approximating to within any constant factor is NP-Hard. (Assign weight 1 to edges $\in E$ and $n\epsilon + 2$ to edges $\notin E$).

3.1 Knapsack

Consider the Knap-Sack problem, $Knap\text{-}Sack(\{a_1 \ldots a_n\}, M, S, P)$ where S is the size function and P the profit function on the set of elements $\{a_1, a_2 \ldots a_n\}$ and M is the maximum size of the knapsack.

The idea is to use the dynamic programming solutions that has show to have a time complexity of $O(nP)$. In this solution the recurrence is

$$KP(j, P) = \min\{KP(j-1, P-p_j) + s_j, KP(j-1, P)\}$$

where $KP(j, P)$ represents the minumum sized knapsack which can achieve a profit P using items $a_1, a_2 \ldots a_j$. The time complexity is $O(NP^*)$ since the maximum value of $P*$ achievable is to be found. Note that $P^* \leq NP_{max}$ and $\geq P_{max}$ where P_{max} is the largest profit item.

We use the technique of scaling to solve this problem. Reducing the profit of each item to $\lfloor p_j/k \rfloor$, for some parameter k, and solving via the above approach gives a solution within a time bound of $N^2 P_{max}/k$. Choose these items as a solution, A.

What is the approximation achieved. Consider the optimum solution OPT comprising items $a_{o1}, a_{o2} \ldots a_{ol}$. Scaling the profits of these items down and then scaling back by multiplying with k gives an error due to the floor operations. Its cost when scaled profits are used is at least $OPT - kl$ since every item can give an error of k when scaled back.

However the solution found has profit $\geq SOPT$ where SO is the scaled profit of the solution OPT. Thus $profit(A) \geq SOPT \geq OPT - kl$. Thus the relative error is $\leq kl/P_{max} \leq Nk/P_{max}$. Choosing $k = \epsilon P_{max}/N$ gives a relative error of atmost ϵ. Thus $profit(A) \geq OPT(1 - \epsilon)$.