NP-Completeness

Definition SAT = { $\langle\phi\rangle$ | ϕ is a satisfiable Boolean formula }

Example $((x \wedge \neg y) \vee \neg x)$ is satisfiable by the assignment x = 1 and y = 0.

Theorem SAT is NP-complete [proof next week]

Definition 3SAT = { $\langle\phi\rangle$ | ϕ is a satisfiable 3CNF formula }

A 3CNF formula has the form $(s_{11} \vee s_{12} \vee s_{13}) \wedge (s_{21} \vee s_{22} \vee s_{23}) \wedge \ldots$
where literals s_{ij} are Boolean variables or negated Boolean variables.

Theorem 3CNF is NP-complete [proof next week]

Theorem $L \in$ NPC

Proof

I. Show that $L \in$ NP
   - A. Certificate: [Describe certificate]
   - B. Algorithm: [Construct verification algorithm V]
   - C. Runtime: [Argue that V runs in polynomial time]

II. Show that $S \leq_P L$ [Find a reduction from S to L, where $S \in$ NPC]
   - A. Algorithm: [Construct reduction algorithm M]
     1. Input: [Give form of input]
     2. Output: [Give form of output]
   - B. Reduction: [Show that M is a reduction]
     1. ($\Rightarrow$): [$x \in S \Rightarrow M(x) \in L$]
     2. ($\Leftarrow$): [$x \notin S \Rightarrow M(x) \notin L$]
   - C. Runtime: [Argue that M runs in polynomial time]

SAT ∈ NPC

SAT ∈ NP

certificate: A satisfying assignment of 0/1 to variables
algorithm:
V = "On input ⟨ϕ, c⟩:
    Scan f, replace variables with 0/1 according to c
    Run E to evaluate the formula
E = "On input ⟨ϕ⟩:
    Find the primary operator
    $\neg \phi_0 \Rightarrow$ run E on input ϕ_0; calculate $\neg \phi_0$
    $\phi_1 \wedge \phi_2 \Rightarrow$ run E on input ϕ_1 and then run E on input ϕ_2; calculate $\phi_1 \wedge \phi_2$
    $\phi_1 \vee \phi_2 \Rightarrow$ run E on input ϕ_1 and then run E on input ϕ_2; calculate $\phi_1 \vee \phi_2$
runtime: E's recursion must run in polynomial time – there are less than n operators in a formula and the recursion only runs once for each operator.

SAT is NP-hard

$\forall$ languages A ∈ NP, A $\leq_P$ SAT

Let A be an arbitrary language in NP.
Let N be a NDTM that decides A in n^k time.

Suppose w ∈ A
$\Leftrightarrow$ N has an accepting branch whose length is $\leq n^k$
$\Leftrightarrow$ $\exists$ an accepting ***tableau*** for N on w

A ***tableau*** of N on w is an $n^k \times n^k$ table whose rows are configurations on a branch of N's computation for w.

#	q_0	w_1	w_2	–	…	w	–	…	–	#	start configuration
#										#	second configuration
#										#	
						window					
#										#	n^k-th configuration

Goal: Generate a formula ϕ (in polynomial-time) that has a "true" assignment iff $\exists$ an accepting tableau for N on w.

Let Q and Γ be the set of states and the tape alphabet of N.
Let $C = Q \cup \Gamma \cup \{\#\}$.
Let $x_{i,j,s}$ be a variable in the formula ϕ that corresponds to cell (i, j) in the tableau and some $s \in C$.
Note: There are a polynomial number (in n) of such variables: $n^k \times n^k \times k'$ [k and k′ are both constants].

Let ϕ be the formula $\phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$, where

- ϕ_{cell} = Every cell in the tableau holds exactly one symbol from C.
- ϕ_{start} = The start configuration is # q0 w_1 w_2 … w_n – … – #
- ϕ_{move} = All transformations in the tableau (from one configuration to the next) are consistent with N's transition function.
- ϕ_{accept} = Some cell in the tableau contains q_{accept}.

$\phi_{cell} = \forall\ i, j \in [1..n^k]$, ($\exists\ s \in C \ni x_{i,j,s} = 1$ **and** $\forall\ s, t \in C$ **where** $s \neq t$, $x_{i,j,s} = 0$ **or** $x_{i,j,t} = 0$)

Let $m = n^k$.

$$\forall\ i, j \in [1..m],\ P(x_{i,j,s}) \equiv \begin{array}{l} P(x_{1,1,s}) \wedge P(x_{1,2,s}) \wedge \ldots \wedge P(x_{1,m,s}) \wedge \ldots \wedge \\ P(x_{2,1,s}) \wedge P(x_{2,2,s}) \wedge \ldots \wedge P(x_{2,m,s}) \wedge \ldots \wedge \\ \ldots \\ P(x_{m,1,s}) \wedge P(x_{m,2,s}) \wedge \ldots \wedge P(x_{m,m,s}) \end{array}$$

$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q0} \wedge x_{1,3,w1} \wedge x_{1,4,w2} \wedge \ldots \wedge x_{1,n+2,wn} \wedge x_{1,n+3,–} \wedge \ldots \wedge x_{1,m-1,–} \wedge x_{1,m,\#}$

$\phi_{accept} = \forall\ i, j \in [1..m],\ x_{i,j,q\text{-}accept}$

$\phi_{move} = \forall\ i, j \in [1..m]$, the (i, j) window is legal (it *might* appear based on the transition)

$\forall$ legal windows $a_1..a_6$, $x_{i-1,j,a1} \wedge x_{i,j,a2} \wedge x_{i+1,j,a3} \wedge x_{i-1,j+1,a4} \wedge x_{i,j+1,a5} \wedge x_{i+1,j+1,a6}$

[Iterating through all legal windows may be costly if C is large, but it will *not* depend on input size, and therefore it can be considered constant]

Runtime of a machine that creates ϕ:

ϕ_{cell} : $O(n^{2k})$
ϕ_{start} : $O(n^k)$
ϕ_{accept} : $O(n^{2k})$
ϕ_{move} : $O(n^{2k})$

Example

Let $\delta(q_1, a) = \{ (q_1, b, R) \}$ and $d(q_1, b) = \{ (q_2, c, L), (q_2, a, R) \}$

Legal windows

a	q_1	a
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

b	b	b
c	b	b

Illegal windows

a	b	a
a	a	a

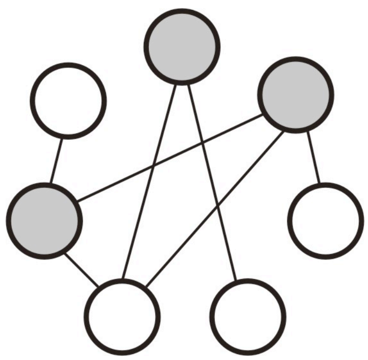
a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

Vertex cover

Let G = (V, E) be an undirected graph.
A **vertex cover** is a subset V' ⊆ V of vertices such that
if (u, v) ∈ E then u ∈ V' or v ∈ V' (or both)

Example



VERTEX-COVER = { ⟨G, k⟩ : graph G has a vertex cover of size k }

Theorem VERTEX-COVER ∈ NPC

Proof
(1) VERTEX-COVER ∈ NP
certificate: subset of V' ⊆ V of vertices
algorithm:
check of |V'| = k
for each edge (u, v) in E
    check if u ∈ V or v ∈ V
poly-time: size check takes O(V); edge check takes O(E)

(2) VERTEX-COVER is NP-hard
(we show CLIQUE $\leq_P$ VERTEX-COVER)

Recall that CLIQUE = { ⟨G, k⟩ | G has a clique of size k }

(A) Algorithm F: ⟨G, k⟩ → ⟨G', k'⟩
construct G' as G* = (V, E*), E* is the compliment of E [set of all edges not in E]
k' = |V| – k

(B) F runs in poly time
$O(V^2)$ time to change 0 ↔ 1 in adjacency matrix.

(C) F computes a reduction

$\langle G, k\rangle \in$ CLIQUE $\Leftrightarrow \langle G^*, |V| - k\rangle \in$ VERTEX-COVER

($\Rightarrow$)
G has clique V' of size k
Let (u, v) be an arbitrary edge in G*
Since (u, v) is not an edge in G,
at least one of u, v must be outside clique V'
Therefore, at least one of u, v must be in V – V'
V – V' covers (u, v)
Since (u, v) is arbitrary, V – V' covers all (u, v) in G*
V – V' is a vertex cover for G* – it has size |V| – k

($\Leftarrow$)
G* has vertex cover V' with size |V| – k
Let both u and v be arbitrary vertices in V – V'
Suppose (u, v) is in G*
Then at least one of u, v must be in vertex cover V' ($\Rightarrow\Leftarrow$)
So (u, v) is in G
Since u and v are arbitrary,
for all u, v in V – V', (u, v) is in G
Therefore, V – V' is a clique of size k for G

[return to the picture – the complement graph has a clique of size four]

<u>Example</u> Show that the subgraph isomorphism problem in NP-complete.

SUB-ISO = { $\langle$G1, G2$\rangle$ | G1 is isomorphic to a subgraph of G2 }

Step 1. Show SUB-ISO $\in$ NP

certificate: An isomorphic mapping *m* from the vertices of G1 to a subset of the vertices of G2.

Verification algorithm V
V = "On input $\langle\langle$G1, G2$\rangle$, y$\rangle$:
for each vertex v in G1
  for each vertex u in G1
    if (u, v) is in G1 and (f(u), f(v)) is not in G2 **then** return false
    if (u, v) is not in G1 and (f(u), f(v)) is in G2 **then** return true
return true
Runtime: V takes O(V2) time

Step II. Show CLIQUE $\leq_P$ SUB-ISO

Reduction algorithm M: $\langle G, k\rangle \to \langle G1, G2\rangle$

M = "On input ⟨G, k⟩:
Construct G2 = G
Construct G1 = complete graph with k vertices

Runtime of M: Polynomial in E + V

Correctness of M: ⟨G, k⟩ ∈ CLIQUE ⇔ ⟨G1, G2⟩ ∈ SUB-ISO
Assume ⟨G, k⟩ ∈ CLIQUE
⇔ G has a clique of size k
⇔ G has a subgraph of size k
⇔ G2 has a subgraph of size k isomorphic to G1
⇔ ⟨G1, G2⟩ ∈ SUB-ISO

Definition CLIQUE = { ⟨G, k⟩ | G is an undirected graph with a clique (a complete subgraph) of k nodes }

Definition VERTEX-COVER = { ⟨G, k⟩ | G is an undirected graph with a vertex cover (a set of nodes that touches every edge of G) of k nodes }

Definition HAMPATH = { ⟨G, s, t⟩ | G has a path from s to t that goes through each node exactly once }

Definition HAMCYCLE = { ⟨G ⟩ | G contains a cycle that goes through each node exactly once }
