pp. 292-311. The Class NP-Complete (Sec. 7.4)

- \( P = \{ L \mid L \text{ decidable in poly time} \} \)
- \( NP = \{ L \mid L \text{ verifiable in poly time} \} \)
- Certainly all \( P \) is in \( NP \)
- **Unknown** if \( NP \) is bigger than \( P \)
- (p. 299) \( NP \)-Complete = subset of \( NP \) where if **any** one is solvable in poly time, then **all** in \( NP \)-Complete are
  - No one has found polynomial algorithms for any in it
  - If someone finds such an algorithm for **any problem** in \( NP \)-Complete, then \( NP \) moves to \( P \)
  - **Unknown** if \( NP \)-complete = \( NP \)
- (p 300) **Theorem 7.27** SAT is in \( P \) iff \( P=NP \)
  - 1\(^{st}\) \( NP \) complete problem
  - Will prove any \( NP \) problem convertible into SAT
  - Needs several intermediate theorems first
• (p. 261) Definition: **Language A is Turing-Reducible to B**, written $A \leq_T B$, if A is decidable relative to B using some function $f: A \rightarrow B$ that transforms instances
  
  i.e. any $w_A$ from A can be mapped/reduced to a $w_B$ in B such that B’s decision on $w_B$ can be converted into decision on $w_A$

  ![Diagram](diagram.png)

  • If B decidable, then so is A.

• (p. 300) Definition 7.28: $f: \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if
  
  • Some polynomial time TM exists
  • which when started with $w$ on tape,
  • halts with just $f(w)$ on its tape,
• (p. 300) Def. 7.29: Language A is **polynomial time reducible to** language to B (Written $A \leq_p B$) if
  • There is some polynomial time computable function $f$
  • Where $w$ is in A iff $f(w)$ is in B
  • See Fig. 7.30, p.301
  • Thus for every string $w$ in A there is a string $f(w)$ in B
  • And if $w$ not in A, then $f(w)$ not in B
  • If you can write a polynomial time decider for B
    • then, using $f$, can write a polynomial time solver for A

  ![Diagram showing function $f$ maps $w$ to $f(w)$ in B, Decider $R$ for B, Map decision from B to one for A, Decision for w instance.]

• (p. 301) Theorem 7.3.1. **If $A \leq_p B$ and B in P, then A in P**
  • Given any $w$ in A
    • Compute $w' = f(w)$ – poly time
    • Run Decider for B and output result – poly time
    • Sum of two poly time functions is still poly
• Two sample problems
• (p. 299) SAT: The Satisfiability Problem
  • SAT = \{wff | wff is satisfiable\}
  • Wff = Well-formed-Formula, made up of
    • Boolean Variables (may take on only 0 or 1)
    • Expressions built from AND, OR, NOT
• (p. 302) CNF: a wff is in conjunctive normal form:
  • The AND of a set of clauses (called a conjunction)
    • Where each clause is the OR of a set of literals called
      a disjunction
    • Where each literal is a variable or its complement
• 3SAT = \{wff | wff in CNF has exactly 3 literals\}
  • E.g. (a_1 \lor b_1 \lor c_1) \land (a_2 \lor b_2 \lor c_2) \land \ldots (a_k \lor b_k \lor c_k)

• Also: CLIQUE =\{<G,k>| G includes a k-clique\}
  • Where a k-clique has k vertices with edges to each other
  • CLIQUE known to be in NP (p. 296)
• (p.302) 3SAT is polynomial time reducible to CLIQUE
  • Proof: convert wffs to graphs
    • Wff \( C = C_1 \lor C_2 \ldots \lor C_k \) (i.e. \( k \) clauses)
    • \( C_i = a_i \lor b_i \lor c_i \) where \( a_i, b_i, c_i \) all literals
  • \( f \) converts wff \( C \) to string \(<G,k>\)
    • \( G \) has \( k \) groups of 3 vertices (each group from a clause)
    • Each vertex in a triple corresponds to a literal
      • And named to match
    • All vertices in \( G \) have edges to all other vertices except
      • No edges between vertices in same triple
      • No edge between vertices with opposite labels (i.e. same variable, different signs)
  • See page 303 for example

\[
(w \mid x \mid y) \land \land (\neg x \mid \neg y \mid z) \land \land (\neg z \mid \neg w \mid \neg x)
\]

We'll connect vertices from different clauses if they are consistent.

Consider \( y = \text{false}, x = \text{true}, w = \text{false}, z = \text{true} \)

Is there a clique of size \( m \) where \( m \) is the number of clauses?

http://cs.nmu.edu/~mkowalcz/cs422w09/36/reduction2.jpg
• (p. 303) Wff C is satisfiable iff G has a k-clique
  • =>: If wff has a satisfying assignment, then each clause has at least one literal that is true
    • Choose just one of these in each triple
      • By construction there must be an edge between all selected vertices & thus must be a k-clique
  • <=: If the graph has a k clique
    • Cannot include vertices in same triple (not permitted by construction)
    • Cannot include literals with opposite sides (not permitted by construction)
    • Assign value to variables to make each literal in k-clique true
    • Result is a satisfying assignment

If CLIQUE is solvable in poly time, so is 3SAT and vv

We'll connect vertices from different clauses if they are consistent.

Consider \( y = \text{false}, x = \text{true}, w = \text{false}, z = \text{true} \)

Is there a clique of size \( m \) where \( m \) is the number of clauses?
• (p. 304) Def 7.34. B is NP-complete if both B in NP and every A in NP is polynomial time reducible to B

• (p. 304) Theorem 7.35. If B is in NP-complete and B in P, then P = NP
  • Any member can be converted to any other by series of polynomial time f

• (p. 304) Theorem 7.36. If B in NP-complete, and B ≤p C for some C in NP, then C is also NP-complete
  • Since B is NP-complete, every language in NP is polynomial time reducible to B,
  • But B is polynomial time reducible to C
  • Can compose the functions, so every language in NP is also polynomial time reducible to C
  • Thus C also in NP-Complete
• (p. 304) **COOK-LEVIN Theorem. SAT is NP-complete!**

• First show SAT is in NP
  • A nondeterministic TM N can guess an assignment and then verify in polynomial time. Thus in NP

• Now show any A in NP is polynomial time reducible to SAT

• \( n = |w| \), \( w \) in A

• \( N \) an NTM that decides A in \( O(n^k) \) for some \( k \)
  • Tape used is thus at most \( n^k \) cells in length

• Construct **tableau** (table) of size \( n^k \times n^k \) (p. 305)
  • \( n^k \) rows (one for each step of NTM)
  • Each row is a configuration
    • 1\(^{st}\) row is **starting configuration** of \( N \) on \( w \)
    • Each configuration at most \( n^k \) symbols long (columns – max tape length)
    • For convenience, each config starts & ends with #
    • Each entry in table called a **cell**
      • A state or a symbol
      • Let \( C = Q \cup \Gamma \cup \{#\} \) = state set + tape chars
  • Tableau is **accepting** if some row an accepting config
    • And row i+1 follows row i via valid transition

• Now to show \( N \) accepts \( w \) is eqvt to question “does an accepting tableau exist?”
• Conversion \( f \) from \( A \) to SAT: Each cell in tableau has a symbol from \( C \)
  
• Define a set of \( 2^k \times 2^k \times |C| \) Boolean variables \( x_{i,j,s} \)
  
  • \( i, j \) between 1 and \( 2^k \)
  
  • \( s \) over all symbols in \( C \)
  
  • \( x_{i,j,s} = 1 \) iff cell\([i,j]\) contains symbol \( s \)

• (p. 306) Define a wff made up of AND of 4 sets of clauses
  
  • \( \text{Wff}_{\text{cell}} \) = clauses ensure 1 variable is true for each \( i,j \)
  
  • \( \text{Wff}_{\text{start}} \) = clause that forces variables with \( i=1 \) to have initial config of \( N \)
  
  • \( \text{Wff}_{\text{accept}} \) = clauses that guarantees an accepting configuration appears as some row
  
  • \( \text{Wff}_{\text{move}} \) = clauses that guarantee that a move from the config for row \( i \) to row \( i+1 \) is valid
    
    • See 6 “windows” on p. 308 for rows \( i \) and \( i+1 \)
    
    • Centered around state symbol in row \( i \)

• This conversion can be done in poly time

• Thus any problem in NP can have its decider (if it exists) converted into a SAT problem in poly time

• Solving the SAT problem finds answer for \( A \)
Sample tableau (for deterministic TM accepting \((aa)^n\))

<table>
<thead>
<tr>
<th>state</th>
<th>tape</th>
<th>new state</th>
<th>new tape</th>
<th>dir</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>a</td>
<td>q1</td>
<td>a</td>
<td>R</td>
</tr>
<tr>
<td>q1</td>
<td>a</td>
<td>q0</td>
<td>a</td>
<td>R</td>
</tr>
<tr>
<td>q0</td>
<td>blank</td>
<td>q2</td>
<td>blank</td>
<td>L</td>
</tr>
</tbody>
</table>

Tableau for \((aa)^n\)

<table>
<thead>
<tr>
<th>Steps (each a configuration)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#</td>
<td>q0</td>
<td>a</td>
<td>a</td>
<td>bl</td>
<td>#</td>
</tr>
<tr>
<td>2</td>
<td>#</td>
<td>a</td>
<td>q1</td>
<td>a</td>
<td>bl</td>
<td>#</td>
</tr>
<tr>
<td>3</td>
<td>#</td>
<td>a</td>
<td>a</td>
<td>q0</td>
<td>bl</td>
<td>#</td>
</tr>
<tr>
<td>4</td>
<td>#</td>
<td>a</td>
<td>q2</td>
<td>a</td>
<td>bl</td>
<td>#</td>
</tr>
</tbody>
</table>

3 cells = 4x6x6, 144 variables

Variable Assignments

<table>
<thead>
<tr>
<th>i</th>
<th>s</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>bl</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>q0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>q1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>q2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>bl</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>q0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>q1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>q2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>bl</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>q0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>q1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>q2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>bl</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>q0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>q1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>q2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
• Remember: showing a problem is NP-Complete
  • Show its in NP (i.e. NTM to create certificate & poly verifier)
  • Show some/any NP-Complete problem polynomially maps to it
    • Not always 3SAT!

• Other NP-Complete problems
  • (p. 310) 3SAT
    • Do logic conversions from any SAT wff to 3 var clauses
  • (p. 311) CLIQUE
    • 3SAT reduces to it via Theorem 7.32 (p. 302)
      • 3 vertices for each clause
        • Labelled with literal name
      • Edges between all vertices, except:
        • Between vertices of a clause
        • Any vertex with any other labelled with the vertex’s literal complement
    • P. 303 addresses match of satisfying solution and k-clique
(p. 312) **VERTEX-COVER** = \{<G,k>| G a graph with a subset of k vertices that has every edge in G touching at least one of the subset\}

- 3SAT reduces to (G,k) k=m+2l, m=# variables, l=# clauses
  - For each variable x create *pair* of 2 vertices (labelled x and ~x) with an edge between them
  - Each clause maps to a *triangle* labelled with variables
    - With edges to matching vertices from 1st set
  - Total of 2m + 3l vertices
- Assume satisfying assignment, show k-cover:
  - Include m vertices from pairs that match assignment
    - Covers edges to clause triangles and other of pair
  - Each triangle has at least 1 vertex in assignment, choose other 2 (2l)
- Assume G has a k-cover, show satisfying assignment
  - Cover must have at least one vertex in each pair
    - Otherwise edge between pair not covered
  - Cover must have at least 2 vertices in each triangle
    - Otherwise cannot get edge in triangle covered
  - For k=m+2l, above must be exact
  - M from pair must be satisfying (p. 313)
(p. 314) **HAMPATH**: \(\{G,s,t|\) there is a path from \(s\) to \(t\) that goes thru all vertices exactly once.\}

- 3SAT of \(l\) variables & \(k\) clauses reduces to HAMPATH.
- For each variable in 3SAT construct *diamond* as Fig. 7.47
  - 3\(k+3\) vertices in center row
    - 2-vertex pair for each clause + 1 border per clause
    - Lefthand vertex for “true” assignment
    - Righthand for “False”
  - Multiple paths from top to bottom
    - Left or right from top to center
    - Optionally across the center, in either direction
    - Left or right to lower vertex
  - Diamonds stacked on top of each other (Fig. 7.49)
    - Vertex \(s\) is topmost; vertex \(t\) is bottommost
  - Additionally, add separate vertex for each clause in 3SAT
    - \(K\) of them
    - If literal \(x_i\) appears in clause \(c_j\) (p. 316, Fig. 7.51)
      - Add edge from left vertex of \(j\)’th pair in center of diamond for \(x_i\) to vertex for \(c_j\)
      - Add edge from \(c_j\) to right vertex of \(j\)’th pair
    - If literal \(\neg x_i\) appears in clause \(c_j\), add edges in opposite
If 3SAT is satisfiable, then Hamiltonian path from s to t
- Starts at top, go left if x1 is true, right if false (Fig. 7.53)
- Go across center, then down to top of next diamond
- Repeat
- Exception: for each clause cj pick one satisfying literal
  - Follow the breakout from the appropriate center row
- Result: all vertices touched exactly once

If HAMPATH exists in graph
- If “normal”: top-down and thru center, with bypass, then can read out satisfying assignment
- Fig. 7.54 (p. 318) cannot occur

(p. 319) **UHAMPATH** – HAMPATH with undirected edges
• (p. 319) **SUBSET-SUM** \( S = \{(S,t) \mid S = \{x_1, \ldots\} \) and for some subset \( Q = \{q_1, \ldots\} \) a subset of \( S \), sum of \( y \)'s = \( t \)}

• 3SAT of \( l \) variables and \( k \) clauses reduces to a Subset-Sum problem with
  
  • 2\( l \) members of \( S = \{y_1, \ldots y_l, z_1, \ldots z_l\} \)
    
    • \( y_i \) and \( z_i \) for variable \( x_i \)
  
  • 2\( k \) members of \( Q = \{g_1, \ldots g_k, h_1, \ldots h_k\} \)
    
    • and \( t = a \) # described below

• Create table as on p. 321
  
  • Each row of \( l + k \) #s:
    
    • \( l \) columns: 1 for each variable
    
    • and \( k \) more columns: 1 for each clause
  
  • Total of 2\( l \) + 2\( k \) + 1 rows:
    
    • 2\( l \) of them: variable \( x_i \) has 2 rows, labelled \( y_i \) and \( z_i \)
      
      • For row \( y_i \): all 0's but a 1 in column for \( x_i \) and a 1 in each clause column having \( x_i \) as a literal
      
      • For row \( z_i \): all 0's but a 1 in column for \( x_i \) and a 1 in each clause column having \( \sim x_i \) as a literal
    
    • 2\( k \) of them: 2 for each clause, labelled \( g_i \) and \( h_i \)
      
      • Row is all 0s but a single 1 in column for clause \( i \)
      
      • One row for \( t \): All 1s for variable columns; all 3s for clause columns
• Treat each row as digits of a number
• Assume wff is satisfiable, show subset
  • select Q as follows
    • If xi assigned true, select yi for Q
    • If xi assigned false, select zi for Q
  • Add up the selected rows
    • Exactly 1 for each of 1\textsuperscript{st} l digits
    • Each of last k digits between 1 and 3
  • To make last k digits all 3
    • Select enough gs and hs to add up to 3
• Assume subset exists, show assignment
  • All digits in each # is either 0 or 1
  • Each column in table has at most 5 1’s
    • At most 3 from literals in clause
    • 2 from gs’ and hs’
  • Thus no carries possible
  • Thus for a 1 in each of first l columns, exactly 1 of ys’ and zs’ must be selected
  • This is assignment
**Summary:** from [https://people.eecs.berkeley.edu/~vazirani/algorithms/chap8.pdf](https://people.eecs.berkeley.edu/~vazirani/algorithms/chap8.pdf)

<table>
<thead>
<tr>
<th>Hard problems (NP-complete)</th>
<th>Easy problems (in P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td>2SAT, HORN SAT</td>
</tr>
<tr>
<td>TRAVELING SALESMAN PROBLEM</td>
<td>MINIMUM SPANNING TREE</td>
</tr>
<tr>
<td>LONGEST PATH</td>
<td>SHORTEST PATH</td>
</tr>
<tr>
<td>3D MATCHING</td>
<td>BIPARTITE MATCHING</td>
</tr>
<tr>
<td>KNAPSACK</td>
<td>UNARY KNAPSACK</td>
</tr>
<tr>
<td>INDEPENDENT SET</td>
<td>INDEPENDENT SET on trees</td>
</tr>
<tr>
<td>INTEGER LINEAR PROGRAMMING</td>
<td>LINEAR PROGRAMMING</td>
</tr>
<tr>
<td>RUDRATA PATH</td>
<td>EULER PATH</td>
</tr>
<tr>
<td>BALANCED CUT</td>
<td>MINIMUM CUT</td>
</tr>
</tbody>
</table>

**Figure 8.7 Reductions between search problems.**

```
All of NP
  ↓
  SAT
    ↓
    3SAT
      ↓
      INDEPENDENT SET
        ↓
        3D MATCHING
          ↓
          VERTEX COVER
            ↓
            CLIQUE
              ↓
              ZOE
                ↓
                SUBSET SUM
                  ↓
                  ILP
                    ↓
                    RUDRATA CYCLE
                      ↓
                      TSP
```
From https://en.wikipedia.org/wiki/NP-completeness