CSE 5311 Notes 18: NP-Completeness

(Elementary Concepts)

Satisfiability: \( (p \lor q) \land (p \lor \bar{q}) \land (\bar{p} \lor q) \land (\bar{p} \lor \bar{q}) \)

Is there an assignment? (Decision Problem)

Similar to debugging a logic circuit - Is there an input case that turns on the output LED?

Aside: Evaluating one input setting for a circuit is P-complete \( \Rightarrow \) hard to massively parallelize.

NP-complete means (informally):

1. The problem may be computed (“decided”) in nondeterministic polynomial time.
   a. Guess a solution (polynomial time - easy to get)
   b. Check the solution in polynomial time (deterministic).

Checking (“verification”) is easier than computing.

2. All problems in NP may be transformed (“reduced”) to this problem in polynomial time.

   If an instance of one problem is transformed to an instance of another, the new problem has a solution iff the old problem has a solution.

Instances of all problems in NP \[\rightarrow\] Instances of new problem
Showing that all NP problems reduce to new problem is unnecessary. Instead, find another NP-complete problem:

![](image)

**Instances of all problems in NP**

**Instances of old NP-complete problem**

**Instances of new problem**

**Difficult proof technique** - required for “first” NP-complete problem (satisfiability - Cook’s theorem)

**“Easy” proof technique**

Property 1 without property 2 - problem is just in NP. Example: Is a table sorted?

Property 2 without property 1 - problem is said to be “NP-hard” (at least as difficult as all other problems in NP). Note: property 1 is usually trivial to establish and is often omitted in proofs.

Significance of a problem being NP-complete

No polynomial-time algorithm is known for any NP-complete problem. (Only exponential time)

If a polynomial-time algorithm is known for one NP-complete problem, then there is a polynomial-time algorithm for every NP-complete problem.

Exponential lower bound has never been shown.

If difficult instances of an NP-complete problem arise in practice, then approximation schemes with bounds on the quality of the solution are needed.

*What about playing chess?*

Example Problems:

Satisfiability

Graph (Vertex) Coloring

Job Scheduling with Penalties - durations, deadlines, penalties (single processor)

Bin Packing - how many fixed-sized bins are needed to hold variable-sized objects?

Knapsack - how many objects with different profits and sizes should go into a knapsack?

Subset Sums - is there a subset whose sum is a particular value?
Hamiltonian Path - does a graph (or digraph) have a path including each vertex exactly once?

Hamiltonian Circuit - is there a cycle including each vertex exactly once?

Traveling Salesperson - minimize distance for Hamiltonian circuit

Steiner subgraph - is there a connected subgraph (tree) that includes designated terminal vertices and whose total weight does not exceed a given value? (Euclidean version is NP-hard)

REDUCTIONS


Suppose you know directed Hamiltonian circuit is NP-complete. Show that undirected Hamiltonian circuit is NP-complete:

1. Replace each vertex $x$ by three vertices $x_1, x_2, x_3$ connected as:

   ![Graph](image)

2. Include an edge $\{u_3, v_1\}$ for each edge $(u, v)$ in the directed graph.

![Graph](image)

Leaving out $x_2$ will not work - allows going in wrong direction:
Show 3-satisfiability is NP-complete by reduction from conjunctive normal form satisfiability.

In CNF an expression is a conjunction of several *clauses* (disjunctions).

Each clause has several *literals* which may be asserted or negated.

The reduction is based on replacing each clause with \( k > 3 \) literals by \( k - 2 \) clauses for 3-satisfiability and introducing \( k - 3 \) new variables:

\[
A \lor B \lor C \lor D \lor E \lor F \lor G \\
\overline{A}, \overline{B}, \overline{C}, \overline{D}, \overline{E}, \overline{F}, \overline{G}
\]

\[
A \lor B \lor X_1 \\
\overline{X}_1 \lor C \lor X_2 \\
\overline{X}_2 \lor D \lor X_3 \\
\overline{X}_3 \lor E \lor X_4 \\
\overline{X}_4 \lor F \lor G \\
\overline{A}, \overline{B}, \overline{C}, \overline{D}, \overline{E}, \overline{F}, \overline{G}
\]

Note, however, that 2-satisfiability is in P. Convert to a graph problem by replacing each \( P \lor Q \) by \( P \rightarrow Q \) and \( \overline{Q} \rightarrow P \) based on \( A \rightarrow B \equiv \overline{A} \lor B \).

If there is a path from \( \overline{X} \) to \( X \), then \( X \) is true. If there is a path from \( X \) to \( \overline{X} \), then \( X \) is false.

If \( X \) and \( \overline{X} \) are in a cycle, then the expression is unsatisfiable.

Consider \( A, B, \overline{A} \lor \overline{B} \):

![Diagram](image)

Show that graph 3-colorability is NP-complete by a reduction from 3-sat.

This reduction is fairly difficult. Others are much worse.

Conceptually, we will call the 3 colors True, False, and Red.

Since coloring is usually viewed as assigning the numbers 0, 1, 2 to the vertices, for any successful coloring there are five renamings based on permutations.
The reduction starts with a triangle to establish which number has which color:

\[ \text{TRUE} \quad \text{FALSE} \quad \text{RED} \]

For each variable \( X \), another triangle is needed to constrain the value:

\[ \overline{X} \quad X \quad \text{RED} \]

For each clause \( X \lor Y \lor Z \) the following pattern is used. At least one of \( X \), \( Y \), and \( Z \) is forced to be true.

Observe:
1. \( X, Y, \) and \( Z \) must have the same color as \( \text{TRUE} \) or \( \text{FALSE} \).
2. One of \( D \) and \( E \) has the same color as \( \text{FALSE} \), the other the same color as \( \text{RED} \).
3. If \( E \) has the same color as \( \text{FALSE} \), then \( Z \) has the same color as \( \text{TRUE} \).
4. If \( E \) has the same color as \( \text{RED} \), then \( D \) has the same color as \( \text{FALSE} \).
### Summary:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TRUE</strong></td>
<td><strong>RED</strong></td>
<td>???</td>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
</tr>
<tr>
<td><strong>TRUE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>TRUE</strong></td>
</tr>
<tr>
<td><strong>TRUE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
</tr>
<tr>
<td><strong>FALSE</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>RED</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
</tr>
<tr>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>FALSE</strong></td>
</tr>
<tr>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>FALSE</strong></td>
<td><strong>RED</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>TRUE</strong></td>
<td><strong>TRUE</strong></td>
</tr>
</tbody>
</table>

Unsatisfiable instance - graph will require 4 colors

If \( \overline{A} \) is removed, 3 coloring is possible.
$k$-clique - complete subgraph with $k$ vertices

Show $k$-clique is NP-complete by a reduction from 3-sat.

Each literal becomes a vertex.

Connect each vertex to the vertices for all other clauses, except for

Is there a clique with one vertex per clause (i.e. $k$ is the number of clauses)?

\[
\overline{A} \\
\overline{B} \\
B \lor \overline{C} \\
A \lor B \lor C
\]

Vertex Cover = set of vertices such that every edge has at least one incident vertex in cover.

Is there a vertex cover with no more than $p$ vertices?

Reduce from $k$-clique:

1. Take complement of graph. (Edge is in complement iff edge is not in graph.)
2. Is there a $|V|$ - $k$ cover? (Choose vertices not in the $k$-clique.)

From $k$-clique example - Is there a 3-cover?
Consider clique when first clause is removed.

\[
\begin{align*}
\overline{B} \\
B \lor \overline{C} \\
A \lor B \lor C
\end{align*}
\]

3-vertex cover

Show Steiner subgraph is NP-complete by a reduction from 3-sat.

Steiner vertex for each possible literal on \( n \) propositions.

Terminal vertex for each of \( m \) clauses, \( u \), and \( v \).

Unit-weight edges in subgraph with \( u \), \( v \), and literal vertices (for choosing assignment).

Edges with weight \( 2n + 1 \) between each clause vertex and vertices for its literals

Is there a subgraph with total weight not exceeding \( 2n + m(2n + 1) \)?
Goal: Performance guarantees for optimization (NP-hard) problems corresponding to NP-complete problems.

1. How fast?

2. Approach:
   - Greedy
   - Online
   - Preprocessing (e.g. MST, DFS)
   - Randomization
   - Restricted cases (e.g. spare or dense graphs)
   - (Parallelism)

3. Quality of solution

   \[
   \text{max ratio} = \frac{\text{Optimal}}{\text{Solution}} \geq 1 \text{ (e.g. knapsack)}
   \]

   \[
   \text{min ratio} = \frac{\text{Solution}}{\text{Optimal}} \geq 1 \text{ (e.g. TSP)}
   \]
4. Generality

Approximation Algorithm - achieve max/min ratio in $O(n^k)$ time ($k$ fixed)

Approximation Scheme - flexible ratio $1 + \varepsilon$ in $O(f(n,\varepsilon))$

Polynomial-time Approximation Scheme - $O(n^{f(\varepsilon)})$

Fully PTAS - $O\left(n^k \left(\frac{1}{\varepsilon}\right)^l\right)$ time

Examples are presented in ascending order of min/max ratio

Edge Coloring (http://ranger.uta.edu/~weems/NOTES5311/misraGriesNew.c)

An unusually optimistic situation . . .

[Graph images]

Vizing’s Theorem

$\Delta(G) \leq X'(G) \leq \Delta(G) + 1$ (Required number of colors is either degree (“Class 1”) or degree + 1 (“Class 2”). For bipartite graphs, $\Delta(G) = X'(G)$.)

NP-complete to test if $\Delta(G) = X'(G)$, but takes only $O(VE)$ to color with $\Delta(G) + 1$ colors ($\Delta(G)$ for bipartite) in an incremental fashion. Thus:

$$\min \text{ ratio} \leq \frac{\Delta(G) + 1}{X'(G)} \leq \frac{\Delta(G) + 1}{\Delta(G)}$$
So, steal a color from another edge and give that edge a new color.

HOW?

Aside: The simpler problem of coloring a bipartite graph using $\Delta$ colors.


Now we describe a method for coloring due to Vizing. Originally each edge of $G$ is uncolored; it must be assigned one of $\Delta$ possible colors. An uncolored edge $(v, w)$ is colored as follows. At most $\Delta - 1$ edges incident to $v$ are colored, so some color $a$ is missing at $v$; similarly, some color $b$ is missing at $w$. Construct an “alternating $(a, b)$ path” starting at $w$, as follows. The path begins with the edge incident to $w$ that is colored $a$ (if it exists). Consecutive edges in the path are alternately colored $a$ and $b$. The path ends at the vertex $z$ where the next color is missing. It is easy to see that $z \neq v, w$ if the graph is bipartite.

Interchange colors along the path, switching $a$ to $b$ and $b$ to $a$. This makes color $a$ missing at both $v$ and $w$, since $z \neq v, w$. Edge $(v, w)$ can now be colored $a$. 
Back to non-bipartite . . .

Most general case - maximal fan with free(X) \( \cap \) free(f) = \( \emptyset \).

If no edge incident to X is colored with last free color d, immediately “rotate” fan:
While maximizing the fan, suppose the next color (d) is already on an earlier fan edge:

1. Find (alternating) dc-path starting with X-C. 

dc-path (above) reaches no more than one of B or Z. (Why?)

2. Invert colors (c ↔ d) along entire dc-path.

   a. Neither B or Z reached - fan stops at B.
b. Z reached - fan stops at B.

3. Rotate fan.

c. B reached - fan keeps all vertices.
Vertex Cover - Approximation Algorithm

\[ VC := \emptyset \]
for each edge \{u, v\} // arbitrary order
    if \( u \not\in VC \) and \( v \not\in VC \)
        \( VC := VC \cup \{u, v\} \)

1. At termination, \( VC \) is a vertex cover.

2. Polynomial time - obvious.

3. a. \( VC^{OPT} \) must cover the set of edges processed based on the “if”.

    b. \( VC^{OPT} \) must include at least one of \{u, v\} for each of these edges, so:

\[
\frac{1}{2} |VC| \leq |VC^{OPT}| \quad \text{min ratio} \leq \frac{|VC|}{|VC^{OPT}|} \leq 2
\]

Aggressive strategy of choosing one vertex from an uncovered edge is vulnerable to “stars”.

Minimum Bipartite Vertex Cover - Exact Solution (not NP-complete . . . unless P = NP)

Instance of max-flow is isomorphic to bipartite matching max-flow, except capacities from \( V_1 \) to \( V_2 \) are \( \infty \).

Set of edges from \( V_1 \) to \( V_2 \) with (unit) flow after max-flow is found is a maximum matching. The size of a maximum matching gives an (obvious) lower bound on the size of a minimum bipartite vertex cover. (Showing that the size of a maximum matching is also an upper bound is more involved and omitted).

**Theorem:** If a minimum \( S-T \) cut is known, then \((V_1 \cap T) \cup (V_2 \cap S)\) is a minimum bipartite vertex cover.

Proof: Suppose the bipartite graph has an edge \( \{v_1, v_2\} \) with \( v_1 \in V_1 \) and \( v_2 \in V_2 \) and \( v_1 \in S \). Since the capacity of \( \{v_1, v_2\} \) is \( \infty \), \( \{v_1, v_2\} \) is an edge in the residual network and \( v_2 \in S \) to prevent \( \{v_1, v_2\} \) from being uncovered. ***
$S = \{s\} \quad T = \{t, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \quad V_1 \cap T = \{0, 1, 2, 3\} \quad V_2 \cap S = \emptyset$

$S = \{s, B, C, H\} \quad T = \{t, A, D, E, F, G, I, J, K, L, M\} \quad V_1 \cap T = \{A, D, E\} \quad V_2 \cap S = \{H\}$

Bin Packing (one dimensional)

Minimize number of unit size bins to hold objects with sizes $0 < s_i \leq 1$.

Next Fit

Online

Use one bin, then seal when next item doesn’t fit.

Worst case sequence

$\frac{1}{2} \cdot 2N \cdot \frac{1}{2} \cdot 2N \cdot \frac{1}{2} \cdot 2N \cdot \frac{1}{2} \cdot 2N \cdots \quad \text{Repeated to get } 4N \text{ elements}$

OPT (offline) \hspace{2cm} Next Fit \hspace{2cm} $\frac{NF}{OPT} \leq 2$
First-fit Decreasing

Sort $n$ sizes descending

For each object, go through bins “left-to-right” to find first bin that object fits in.

Achieves $\frac{FF}{OPT} \leq 1.5$ (not hard to improve ratio to 4/3, difficult to get 11/9)

Claim: Objects placed in extra bins have size $\leq 1/2$

Proof: Suppose otherwise.

Claim: Number of objects in extra bins $\leq OPT(S) - 1$

Proof: Suppose $OPT(S)$ extra objects are used.

1. Waste in every $OPT$ bin $< \text{size of every object in extra bins}.$

2. Consider $\sum_{i=1}^{OPT} (OPT_i \text{ sum} + \text{Extra Object}_i)$

But since each $OPT_i \text{ sum} + \text{Extra Object}_i > 1$, this sum exceeds $OPT$, a contradiction.

Based on the two claims, the number of extra bins $\leq OPT/2$ and the ratio is 1.5.

Set Cover

Input: Set $S$ and subsets such that $S = \bigcup_{i=1}^{n} S_i$

Output: Small set of subsets covering $S$.

Greedy Technique:

Choose subset with largest number of uncovered elements.

(Implementation: Doubly-linked list for each element in $S$. Doubly-linked list for each subset. Ordered table for priority queue.)

Achieves: $\frac{Greedy}{OPT} \leq \ln(\text{largest subset}) + 1$

See CLRS, p. 1119-1121 for detailed proof.
Example to motivate logarithmic approximation ratio:

Traveling Salesperson (complete graph)

No $\rho$-approximation in P time (unless P = NP).

Suppose a graph is to be tested for a Hamiltonian cycle:

Weight each “real” edge with 1.

“Imaginary” edges are weighted with $\rho|V| + 1$.

If $\rho$-approximation gives TSP with length $|V|$, then performance is better than guaranteed and have a Hamiltonian cycle.

If $\rho$-approximation gives TSP with length $> \rho|V|$, then performance guarantee has not been met.

If edge weights obey triangle inequality, the scale-up problem is avoided.
2-approximate

1. Find minimum spanning tree.
2. Depth-first search - order vertices by discovery time.
3. Return to start vertex.
4. Remove edge crossings - optional

2-approximate proof:

1. $|MST| \leq |OPT|$

   Best case - removing largest edge in OPT (a cycle) gives MST.

2. $|T_{\Delta}| \leq 2|MST|$

   Since MST short cuts are no longer than subpath skipped.

   $\frac{1}{2}|T_{\Delta}| \leq |MST| \leq |OPT|$, so $|T_{\Delta}| \leq 2|OPT|$

   (Aside: see http://en.wikipedia.org/wiki/Christofides_algorithm for a 3/2-approximate technique.)
Subset Sums - exact solution by dynamic programming (ssNew.c)

Given \( n \) numbers \( S_1, \ldots, S_n \), find a subset (e.g. an index set chosen from \( 1 \ldots n \)) such that sum is an exact value \( C \).

1. Use table with \( C \) entries

   \[ A[i] = j, \text{ where } j \text{ is the smallest index such that } i = S_j + \text{values w/index} < j \]

2. Initialize table and then go forward

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & C=36 \\
2 & 3 & 6 & 11 & 15 & 25 \\
\end{array}
\]

\[
A: \begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 2 & 3 & & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\
4 & & & & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\
6 & & & & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
31 & 32 & 33 & 34 & 35 & 36 & & & & \\
\end{array}
\]

Now suppose each \( S_i \) is multiplied by 1000. D.P. table grows by factor of 1000.
Ordinary Subset Sum (CLRS, p. 1128-1129)

Saves space by maintaining lists of reachable sums.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
<td>11</td>
<td>15</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

Target=36

L0={0/0}

L1={0/0, 2/1}

L2={0/0, 2/1, 3/2, 5/2}

L3={0/0, 2/1, 3/2, 5/2, 6/3, 8/3, 9/3, 11/3}

L4={0/0, 2/1, 3/2, 5/2, 6/3, 8/3, 9/3, 11/3, 13/4, 15 17 18 20 21 23 24 26 28 14/4, 16/4, 17/4, 19/4, 20/4, 22/4} 29 31 32 34 35 37?

L5={0/0, 2/1, 3/2, 5/2, 6/3, 8/3, 9/3, 11/3, 13/4, 14/4, 15/5, 16/4, 17/4, 18/5, 19/4, 20/4, 21/5, 22/4, 23/5, 24/5, 26/5, 28/5, 29/5, 31/5, 32/5, 34/5, 35/5}

L6={0/0, 2/1, 3/2, 5/2, 6/3, 8/3, 9/3, 11/3, 13/4, 14/4, 15/5, 16/4, 17/4, 18/5, 19/4, 20/4, 21/5, 22/4, 23/5, 24/5, 25/6, 26/5, 27/6, 28/5, 29/5, 30/6, 31/5, 32/5, 33/6, 34/5, 35/5, 36/6}
Subset Sum with Range of Target Values

Uses intervals to achieve approximation to within $\epsilon$ of desired value.

CLRS, p. 1130-1133, gives fully PTAS based on $\epsilon$.

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
2 & 3 & 6 & 11 & 15 & 25
\end{array}
\]

Target=33..36

$L_0={0..3/0}$

\[2..5\]

$L_1={0..3/0, 3..5/1}$

\[3..6 \quad 6..8\]

$L_2={0..3/0, 3..5/1, 5..8/2}$

\[6..9 \quad 9..11 \quad 11..14\]

$L_3={0..3/0, 3..5/1, 5..8/2, 8..14/3}$

\[11..14 \quad 14..16 \quad 16..19 \quad 19..25\]

$L_4={0..3/0, 3..5/1, 5..8/2, 8..14/3, 14..25/4}$

\[15..18 \quad 18..20 \quad 20..23 \quad 23..29 \quad 29..40?\]

$L_5={0..3/0, 3..5/1, 5..8/2, 8..14/3, 14..25/4, 25..36/5}$

\[25..28 \quad 28..30 \quad 30..33 \quad 33..39?\]

$L_6={0..3/0, 3..5/1, 5..8/2, 8..14/3, 14..25/4, 25..36/5}$
Graph (Vertex) Coloring - no efficient heuristic (unless P = NP)

Suppose a P time algorithm exists to color every graph $G$ with $X(G) \geq k$ using fewer than $\frac{4}{3} X(G)$ colors, then 3-colorability would be in P.

Proof:

1. $C_k$ is the complete graph with $k$ vertices.

2. $G$ is an instance of 3-colorability.


4. If $X(G) = 3$, each copy of $G$ requires 3 colors, so $X(H) = 3k$.

   Algorithm must use fewer than $\frac{4}{3}(3k) = 4k$ colors for $G$ to be 3-colorable.

   Otherwise, $X(G) > 3$ so each copy of $G$ needs at least 4 colors. $X(H) \geq 4k$

   Algorithm must then use at least $4k$ colors to color $H$.

Thus, such an algorithm is “unlikely”.

Each vertex in a copy of $G$ is connected to all vertices in all other copies of $G$.

"G" and "C_k" are depicted in the diagram.