Proving NP-Completeness

- A is NP-complete if
  - A is in NP
  - Some known NP-complete problem is reducible to A in polynomial time

3-CNF-Satisfiability

- Input: A Boolean formula F with at most 3 literals per clause.
- Output: Determine if F is satisfiable.

3-CNF-Satisfiability is NP-complete
- This is probably the most used NP-complete problem in reduction proofs showing other decision problems are NP-hard.

Reduction by Example

Given \( F = (x_1 \lor \neg x_2 \lor x_3 \lor \neg x_4) \land F' \)

Construct \( H = (x_1 \lor z_2) \land (\neg x_2 \lor \neg z_1 \lor z_3) \land (x_1 \lor x_2 \lor z_2) \land (\neg x_4 \lor \neg z_1) \land F' \)

\( F \) is satisfiable if and only if \( H \) is satisfiable.

\( x_1 = 0 \) satisfies the first clause of \( F \).
\( z_2 = 1, z_1 = 0, z_3 = 0 \) satisfy clauses 1, 3, and 4 of \( H \) and \( x_2 = 0 \) satisfies the clause 2 of \( H \).

3-Colorability

- Input: Graph G = (V,E) and a number k.
- Output: Determine if all vertices can be colored with 3 colors such that no two adjacent vertices have the same color.

3-CNF-Sat \( \leq_p \) 3-Color

- Given a 3-CNF formula \( F \) we have to show how to construct in polynomial time a graph \( G \) such that:
  - \( F \) is satisfiable implies \( G \) is 3-colorable
  - \( G \) is 3-colorable implies \( F \) is satisfiable
The Gadget

- This is a classic reduction that uses a "gadget".
- Assume the outer vertices are colored at most two colors. The gadget is 3-colorable if and only if the outer vertices are not all the same color.

Properties of the Gadget

- Three colorable if and only if outer vertices not all the same color.

Reduction by Example

\[ F = (x \lor y \lor z) \land (\neg x \lor y \lor z) \land (\neg x \land \neg y \lor \neg z) \]

Satisfaction Example

\[ F = (x \lor y \lor z) \land (\neg x \lor y \lor z) \land (\neg x \land \neg y \lor \neg z) \land z = 0 \]

\[ x = 1 \]

\[ y = 1 \]

\[ z = 0 \]
Satisfaction Example \[ F = (x \lor y \lor z) \land (\neg x \lor y \lor z) \land (\neg x \lor \neg y \lor \neg z) \]
\[ x = 1 \quad y = 1 \quad z = 0 \]

Non-Satisfaction Example \[ F = (x \lor y \lor z) \land (\neg x \lor y \lor z) \land (\neg x \lor \neg y \lor \neg z) \]
\[ x = 0 \quad y = 0 \quad z = 0 \]

Naming the Gadget

General Construction
\[ F = \bigcap_{i=1}^{k} (a_i \lor a_j \lor a_k) \quad \text{where} \quad a_i \in \{ x, \neg x_1, \ldots, x_n, \neg x_n \} \]
\[ G = (V, E) \quad \text{where} \]
\[ V = \{ r, g, b \} \cup \{ x, \neg x_1, \ldots, x_n, \neg x_n \} \cup \{ O, U, T, I, N, R \} \quad 1 \leq i \leq k \]
\[ E = \{ \{ r, g \}, \{ g, b \}, \{ b, r \} \}
\[ \cup \{ \{ x, \neg x_1 \}, \ldots, \{ x, \neg x_n \} \}
\[ \cup \{ \{ x_1, \neg x_1 \}, \ldots, \{ x_n, \neg x_1 \} \}
\[ \cup \{ \{ r, U \}, \{ r, T \}, \{ I, N \}, \{ N, R \} \} \quad 1 \leq i \leq k \]
\[ \cup \{ \{ r, O \}, \{ r, U \}, \{ r, T \} \} \quad 1 \leq i \leq k \]
\[ \cup \{ \{ r, g \}, \{ g, b \}, \{ b, r \}, 1 \leq i \leq k \} \]

Reductions

Exact Cover
- Input: A set \( U = \{ u_1, u_2, \ldots, u_n \} \) and subsets \( S_1, S_2, \ldots, S_n \subseteq U \)
- Output: Determine if there is set of pairwise disjoint set that union to \( U \), that is, a set \( X \) such that:
  \[ X \subseteq \{ 1, 2, \ldots, m \} \]
  \[ i, j \in X \text{ and } i \neq j \text{ implies } S_i \cap S_j = \emptyset \]
  \[ \bigcup_{i=1}^{n} S_i = U \]
Example of Exact Cover

\[ U = \{a, b, c, d, e, f, g, h, i\} \]
\[ \{a, c, e\}, \{a, f, g\}, \{b, d\}, \{b, f, h\}, \{e, h, i\}, \{f, h, i\}, \{d, g, i\} \]

Exact Cover
\[ \{a, c, e\}, \{b, f, h\}, \{d, g, i\} \]

3-Partition

- Input: A set of numbers \( A = \{a_1, a_2, \ldots, a_n\} \) and number \( B \) with the properties that \( B/4 < a_i < B/2 \) and \( \sum_{j=1}^{n} a_j = mB \).
- Output: Determine if \( A \) can be partitioned into \( S_1, S_2, \ldots, S_m \) such that for all \( i \)

\[ \sum_{j \in S_i} a_j = B. \]

Note: each \( S_i \) must contain exactly 3 elements.

Example of 3-Partition

- \( A = \{26, 29, 33, 33, 33, 34, 35, 36, 41\} \)
- \( B = 100, m = 3 \)
- 3-Partition
  - 26, 33, 41
  - 29, 36, 35
  - 33, 33, 34

Bin Packing

- Input: A set of numbers \( A = \{a_1, a_2, \ldots, a_n\} \) and numbers \( B \) (capacity) and \( K \) (number of bins).
- Output: Determine if \( A \) can be partitioned into \( S_1, S_2, \ldots, S_K \) such that for all \( i \)

\[ \sum_{j \in S_i} a_j \leq B. \]

Bin Packing Example

- \( A = \{2, 2, 3, 3, 3, 4, 4, 4, 5, 5\} \)
- \( B = 10, K = 4 \)
- Bin Packing
  - 3, 3, 4
  - 2, 3, 5
  - 5, 5
  - 2, 4, 4

Perfect fit!

Degree Bounded Spanning Tree

- Input: An undirected graph \( G = (V, E) \).
- Output: A number \( k \) and a spanning tree \( (V, T) \) of degree \( k \). Furthermore, there is no spanning tree of degree < \( k \).
**DBST Decision Problem**

- Input: An undirected graph $G = (V,E)$ and number $k$.
- Output: Determine if $G$ has a spanning tree of degree less than or equal to $k$.
- DBST is NP-complete

**Hamiltonian Path Decision Problem**

- Input: Undirected Graph $G = (V,E)$.
- Output: Determine if there is a path in $G$ that visits each node exactly once.
- Hamiltonian Path is known to be NP-complete

**Hamiltonian Path is Polynomial time Reducible to Spanning Tree of Degree 2**

- Given a graph $G$, $G$ has a Hamiltonian path if and only if $G$ itself has a spanning tree of degree $\leq 2$
- Why? a Hamiltonian path is a spanning tree of degree 2

**5-Color**

- Input: Undirected graph $G$.
- Output: Determine if the vertices of $G$ can be colored in 5 colors with no two adjacent vertices the same color.
- 5-Color is NP-complete
Reduction of 3-Color to 5-Color

G is 3-colorable if and only if G’ is 5-colorable

Exercise – Argue NP-completeness

1. Independent Set
   - Input: Undirected graph G = (V,E) and a number k.
   - Output: Determine if there is an independent set of size k. X, contained in V, is independent if for all i,j in X there is no edge in G from i to j.

2. Equal Subset-Sum
   - Input: \{a_1, a_2, ..., a_n\} positive integers
   - Output: Determine if there is a set I such that \[ \sum_{i \in I} a_i = \sum_{j \notin I} a_j \]

Coping with NP-completeness

• You have encountered a Hard Problem
• Maybe it is NP-hard
  - Books
  - Garey and Johnson
  - Websites
    - http://www.nada.kth.se/~viggo/problemlist/compendium.html
  - Research papers
  - Maybe you’ll have to do your own reduction
• Can’t determine NP-hardness, then it is probably hard in some way.
• Modify the problem to be more tractable

Boundary Between P and NP

• Satisfiability
  - 2-CNF-SAT is in P
  - 3-CNF-SAT is NP-complete
• Coloring
  - 2-COLOR is in P
  - 3-COLOR is NP-complete
• Planar Colorability
  - Planar graphs are always 4-colorable
  - 3-PLANAR-COLOR is NP-complete

Boundary Continued

• Independent Set
  - Maximum independent set is NP-hard
  - Maximal independent set is in P
• Cutting a graph
  - Maximum cut in a graph is NP-hard
  - Minimum cut in a graph is in P (equivalent to Max Flow)
• Spanning Tree
  - Minimum spanning tree is in P
  - Degree constrained spanning tree is NP-hard
  - Bounded diameter spanning tree is NP-hard

Lessons When Coping

• Lesson 1. Any problem that is in NP may be NP-complete.
• Lesson 2. Any problem in NP may be in P.
• Lesson 3. You may not be able to determine either
  - factoring is open
  - graph isomorphism is open
Solving NP-Hard Problems Exactly

- Pseudo Polynomial Time Algorithms
- Branch-and-Bound Algorithm
- Use general packages
  - SAT solvers
  - GSAT, WalkSAT
  - Integer programming solvers
  - CPLEX

Pseudo Polynomial Time Algorithm for an NP-Complete Problem

- Subset Sum
  - Input: Integers \( a_1, a_2, \ldots, a_n, b \)
  - Output: Determine if there is subset \( X \subseteq \{1, 2, \ldots, n\} \) with the property \( \sum_{i \in X} a_i = b \)
  - Algorithm:
    - Let \( A[0..b] \) be a Boolean array of size \( b + 1 \) initialized as follows: \( A[0] = 1 \) and \( A[i] = 0 \) for \( 1 < i < b \).
    - After scanning the input \( a_1, a_2, \ldots, a_k \), maintaining the invariant that \( A[i] = 1 \) if and only if some subset of \( a_1, a_2, \ldots, a_k \) adds up to \( i \).
    - Time Complexity is \( O((b+1)n) \)
    - Polynomial time?
      - No
    - How about finding the subset?
      - Keep track of index that made \( A[x] = 1 \) for the first time.

Example of the Algorithm

<table>
<thead>
<tr>
<th>3, 5, 2, 7, 4, 2, b = 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>1 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 1 0 0 0 1 0 1 0 0</td>
</tr>
<tr>
<td>1 0 1 1 0 1 0 1 1 0 1 0</td>
</tr>
<tr>
<td>1 0 1 1 0 1 0 1 1 1 1 0</td>
</tr>
<tr>
<td>1 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Load Balanced Spanning Tree Cost Criteria

- Given a graph \( G = (V,E) \) and a spanning tree \( T \):
  - \( d(T) = \max \) degree of any vertex of \( T \)
  - \( c(T) = \sum \) of the squares of the degrees

\[
d(T) = 3 \\
c(T) = 4^2 + 1^2 + 2^2 = 26
\]

Advantage of \( c(T) \) is that it has finer gradations.

Deriving \( c(T) \)

- Every spanning tree on \( n \) vertices has \( n-1 \) edges. Hence, the average number of edges per vertex is \( d = 2(n-1)/n \), about 2.
- Let \( d_i \) be the degree of vertex \( i \). The variance in degree is
  \[
  \sum_{i=1}^{n} (d_i - d)^2 / n = \sum_{i=1}^{n} d_i^2 / n - d^2 / n
  \]
- Minimizing the variance is equivalent to minimizing
  \[
  \sum_{i=1}^{n} d_i^2
  \]
Examples of \( c(T) \)

\[
c(T) = 9 \times 1^2 + 1 \times 9^2 = 90 \\
c(T) = 2^11^2 + 8 \times 2^2 = 34
\]

Another Example

\[
c(T) = 3 \times 1^1 + 3 \times 4 + 1 \times 9 = 24 \\
c(T) = 2 \times 1^1 + 5 \times 4 = 22
\]

Load Balanced Spanning Tree with Minimum Variance

- Input: Undirected graph \( G = (V,E) \).
- Output: A spanning tree that minimizes the sum of the squares of the degrees of the vertices in the tree.

Branch and Bound

- Start with an initial tree \( T \) with cost \( c(T) \).
- Systematically search through all forests by recursively (branching) adding new edges to the current forest.
- Discontinue a search if the forest cannot be contained in a spanning tree of smaller cost. (This is the bounding step).
- This is better than exhaustive search, but it is still only valuable on very small problems.

Example of Branch and Bound

Bounding Condition

- Let \( c(F) \) be the cost of the current forest of \( k \) trees where tree \( T \) had minimum degree vertex \( d \) sorted smallest to largest. Let \( B \) be the best cost of any tree so far.
- The lowest possible cost of any tree containing \( F \) is

\[
m(F) = c(F) + \sum_{i=1}^{k} (d_i + 1)^2 - 2 \sum_{i=1}^{k} d_i^2 - \sum_{i=1}^{k} (d_i + 1)^2 + \sum_{i=1}^{k} d_i^2
\]

- If \( m(F) \geq B \) then do not continue searching from \( F \).
Graphic of Bounding Condition

Example of Bounding

\[ d_1 < d_2 < d_3 < d_4 < d_5 \]

\[ d_1 = 0,1,1,1 \]
\[ c(F) = 1*0 + 8*1 + 1*16 = 24 \]
\[ m(F) = 24 + 2(1*1 + 1*4) - 2(1*0 + 1*1) \]
\[ = 36 \]

Branch and Bound Control

Notes on Branch and Bound

- Branch and bound is still an exponential search. To make it work well many efficiencies should be made.
  - No copy on recursive call.
  - Maintain cost of partial solution \( F \) and its sequence of minimum degrees to make computation of \( m(F,i) \) fast.
  - Use disjoint union-find for cycle checking.
  - Reduce use of expensive bounding checks when possible.
  - Add more bounding checks

Approximate Solutions

Local Search Algorithms

- Start with an initial solution that is usually easy to find, but is not necessarily good.
- Repeatedly modify the current solution to a nearby one looking for better ones.
Neighborhood of a Solution

- Cut an edge breaking tree into two trees.
- Add an edge joining the two parts together again.

Equivalent Move

- Add an edge forming a cycle.
- Delete an edge in the cycle.

Recall Cost

- $c(T) = 26$
- $c(T) = 24$
- $c(T) = \text{sum of squares of the degrees of the vertices in } T$

Solution Space

- Moves are reversible.
- $O(n^m)$ neighbors

Every Spanning Tree is Reachable from Every Other

- Let $T$ and $T'$ be two spanning trees. We can move $T$ closer to $T'$ by adding an edge from $T$ to $T$ and removing an edge in the cycle formed that is not in $T'$.

Potential of Local Search

- Since every spanning tree is reachable from any other, then starting with an arbitrary spanning tree we can possibly move to an optimal one using local search.
- Impediment: there usually are exponentially many spanning trees. The search space is exceedingly large.
- In what direction do we search?
- Do we know we are at an optimum
Greedy Local Search

- Find the best neighbor and continue.

\[ T := \text{an initial tree}; \]
\[ \text{best-cost} := c(T); \]
\[ \text{repeat} \]
\[ \text{cost} := \text{best-cost}; \]
\[ \text{for each neighbor } T' \text{ of } T \text{ do} \]
\[ \text{if } c(T') < \text{best-cost} \text{ then} \]
\[ T := T'; \]
\[ \text{best-cost} := c(T); \]
\[ \text{until (best-cost = cost)} \]
\[ \text{return}(T) \]

Greedy Example (1)

Greedy Example (2)

Analysis of Greedy Local Search

- Assume \( n \) vertices, \( m \) edges and \( D \) is the sum of the squares of the degrees in \( G \). \( D \leq n^2 \).
- There are at most \( D \) iterations in the algorithm.
- Each iteration consists of looking at each edge in the spanning tree and replacing it with some other edge, and checking for a cycle and computing costs. This is roughly \( O(n^2m) \) time per iteration.
- Total time is \( O(Dn^2m) = O(n^4m) \) (worst case).

Notes on Greedy Local Search

- Can be very effective for some problems. The worst case time is not that bad.
- Examining all the neighbors and choosing the best is sometimes called “steepest decent”.
- An alternative is “random decent”. Randomly choose a neighbor and move to it if its cost is smaller.
- Another alternative is “first decent”. Try the neighbors in order and move to the first improvement.

Local Minimum Problem

- Greedy local search leads to a local minimum in the solution space, not necessarily a global minimum.
Avoiding Local Minima

- Systematic probing
- Mixing in random moves
- Simulated Annealing
- Random restarts
- Genetic algorithms

Systematic Probing

- Move from the current solution by probing systematically, even if the visited solutions are not better than the current one.
- After the probe choose the solution that was found to be the best.

Systematic Probing Example

- Switch and mark
Using Randomness to Avoid Local Minima

- We maintain a trial solution.
  - Generate a random move from the trial solution.
  - If the move would beat the trial solution then accept it as the new trial solution.
  - If the move does not improve the solution then accept it with some small probability.
- This enables us to navigate the entire solution space and not get caught in a local minimum.

Mixing in Random Moves

- WalkSAT
  - Maximize the number of clauses satisfied in a CNF formula

  | Maintain the best assignment and a current assignment |
  | Repeat |
  | Randomly pick an unsatisfied clause in the current assignment; |
  | Flip a biased coin with probability of heads = p; if heads then flip a random variable in the clause; (walk step) |
  | if tails then flip all the variables in the clause to find an assignment that maximizes the number of satisfied clauses; (greedy step) |

Simulated Annealing

- Kirkpatrick (1984)
- Analogy from thermodynamics.
- The best crystals are found by annealing.
  - First heat up the material to let it bounce from state to state.
  - Slowly cool down the material to allow it to achieve its minimum energy state.

Heating and Cooling Helps (1)

- Local minimum
- Global minimum

Heating and Cooling Helps (2)

- Local minimum
- Global minimum

Heating and Cooling Helps (3)

- Local minimum
- Global minimum
Annealing Concepts

- Solution space S, x in S is a solution
- E(x) is the energy of x
- x has a neighborhood of nearby states
- T is the temperature
- Cooling schedule, in step t of the algorithm
  - Fast cooling $T = ae^{-bt}$
  - Slower cooling $T = at^{-k}$
Metropolis Algorithm

- Initialize $T$ to be hot.
- Choose a starting state.
- Repeat
  - Generate a random move.
  - Evaluate the change in energy $\Delta E$.
  - If $\Delta E < 0$, then accept the move.
  - Else, accept the move with probability $e^{-\Delta E/T}$.
- Update $T$ until $T$ is very small (frozen).

Applied to Load Balanced Spanning Tree

- A state is a spanning tree.
- $T'$ is a neighbor of $T$ if it can be obtained by deletion of an edge in $T$ and insertion of an edge not in $T$.
- Energy of a spanning tree $T$ is its cost, $c(T)$.
- $T'$ is a neighbor of $T$, $\Delta E = c(T') - c(T) > 0$.
- Probability of moving to a higher energy state is $e^{(c(T') - c(T))/T}$.
  - Higher if either $c(T') - c(T)$ is small or $T$ is large.
  - Lower if either $c(T') - c(T)$ is large or $T$ is small.

Notes on Simulated Annealing

- Not a black box algorithm.
  - Requires tuning the cooling parameter.
  - Has been shown to be very effective in finding good solutions for some optimization problems.
- Known to converge to optimal solution, but time of convergence is very large. Most likely converges to local optimum.
- Very little known about effectiveness generally.