Lecture 14: Overview

- NP-Completeness (Chapter 13)
CNF-SAT is NP-Complete

- CNF=Conjunctive normal form
  \((!x_1+x_2+x_4+!x_7)(x_3+!x_5)(!x_2+x_4+!x_6+x_8)\)
- In NP - easy to see...given correct answer, can verify that a CNF problem has a solution in polynomial time
- NP-hard
  - Show by giving polynomial reduction procedure from Circuit satisfiability problems
Cont’d

- Idea: add more variables to circuit satisfiability problem
- Add one variable for the output of each gate
- Convert each gate into a if and only if clause
- Use truth table to convert these into CNF clauses
  - Write DNF formula for when the iff clause is false
  - Use De Morgan’s law to get CNF formula for clause
3-SAT is NP-Complete

• 3-SAT=Conjunctive normal form with 3 literals per clause

\[ (!x_1 + x_2 + x_4)(x_3 + !x_5 + x_4)(!x_2 + !x_6 + x_8) \]

• In NP - easy to see...given correct answer, can verify that a CNF problem has a solution in polynomial time

• NP-hard
  • Show by giving polynomial reduction procedure from CNF-SAT
Reduction

- If the original clause was a single term $C_i=(a)$
  - Replace $C_i$ with $S_i=(a+b+c)(a+!b+c)(a+b+!c)(a+!b+!c)$
    where $b$ and $c$ are new variables not used anywhere else

- If the original clause contained two terms $C_i=(a+b)$
  - Replace $C_i$ with $S_i=(a+b+c)(a+b+!c)$
    where $c$ is a new variable not used anywhere else
cont’d

• If the original clause contained three terms $C_i = a + b + c$, just use it
  • $S_i = a + b + c$

• If the original clause contains more than three terms, $C_i = a_1 + a_2 + a_3 + \ldots + a_k$ for $k > 3$, then we replace $C_i$ with
  • $S_i = (a_1 + a_2 + b_1)(!b_1 + a_3 + b_2)(!b_2 + a_4 + b_3)\ldots(!b_{k-3} + a_{k-1} + a_k)$, where $b_1, b_2, \ldots, b_{k-1}$ are new variables not used anywhere else
Vertex Cover is NP-Complete

• A vertex cover in an undirected graph is a subset $C$ of the vertices of at most size $k$ such that for each edge $(v, w)$ we have $v \in C$ or $w \in C$

• It is NP (given $C$, straightforward to check this property for each edge)

• NP-Hard by giving a reduction from 3SAT problems to vertex covers
Reduction of 3SAT to VERTEX COVER

• For each variable $x_i$ in the 3SAT formula, add two vertices to $G$. One labeled $x_i$ and the other labeled $\neg x_i$. Also add the edge $(x_i, \neg x_i)$ to the graph. Vertex cover must include at least one of these vertices. (truth-setting component)

• For each clause $C_i = (a+b+c)$ in $S$, form a triangle consisting of three vertices $i1$, $i2$, and $i3$ and the three edges $(i1, i2)$, $(i2, i3)$, and $(i3, i1)$

• Note that a vertex cover will have to include at least two of the vertices in the triangle

• Add the edges $(i1, a)$, $(i2, b)$, $(i3, c)$

• Set $k = n + 2m$ where $n =$ number of variables, and $m =$ number of clauses
Solution to 3-SAT problem => Vertex Cover

- Look at each clause in 3-SAT problem
- SAT solution provides variable settings, construct set $C$ that contains the vertices corresponding to these variables in the truth setting component
- If the SAT problem has a solution, the solution must make one or more of the literals in each clause true
- Choose exactly one of these true literals for each clause
- Include other two “clause” vertices in $C$

$\Rightarrow$ Vertex covering of size $n + 2m$
Vertex covering => satisfying assignment

• Must contain at least one vertex in each truth setting component
• Must contain at least two vertices in each clause setting component
• Leaves one edge incident to a clause-satisfying component that is not covered by a vertex in the clause-satisfying component
  • This edge must be covered by a literal vertex
• Can assign covered literal vertices value of 1, and satisfy all clauses
Approximation Algorithms

- Problem:
  - Have NP-complete problem
  - Problem is important to solve
  - Can accept near optimal problem
- Approximation ratio
  - Let $C$ be the cost of solution produced by an algorithm and $C^*$ be the cost of an optimal solution
  - Define $\rho(n) \geq \max(C/C^*, C^*/C)$
  - We call such an algorithm a $\rho(n)$ approximation algorithm
Traveling salesman problem

- Complete undirected graph $G = (V, E)$ that has a nonnegative integer cost $c(u, v)$ associated with each edge $(u, v)$ in $E$
- Want to find Hamiltonian cycle of $G$ with minimum costs
- Let $c(A)$ denote the total cost of the edges in the subset $A \subseteq E$
- We assume the triangle inequality $c(u, w) \leq c(u, v) + c(v, w)$
Approx-TSP-Tour(G, c)

1 select vertex \( r \in V[G] \) to be the root vertex
2 compute minimum spanning tree \( T \) for \( G \) from root \( r \) using MST-Prim(G, c, r)
3 let \( L \) be the list of vertices visited in a pre-order tree walk of \( T \)
4 return the hamiltonian cycle \( H \) that visits the vertices in the order \( L \)
Traveling Salesman Problem

• Approx-TSP-Tour is a polynomial-time 2-approximation algorithm for the TSP problem with the triangle inequality.

• Let $H^*$ denote an optimal tour for the given set of vertices. Since a MST tree has the smallest total weight of any set of edges that connects all the vertices, the weight of the MST $T$ is a lower bound on the optimal tour.

• Full walk of $T$ lists vertices when they are first visited and when they are returned to after a visit to the subtree.
Since full walk traverses each edge of $T$ exactly twice, we have

- $c(W) = 2c(T)$
- $c(W) \leq 2c(H^*)$

Cost of $W$ is within a factor of two of the cost of an optimal tour.