Tractable & Intractable Problems

- We will be looking at :
  - What is a P and NP problem
  - NP-Completeness
  - The question of whether P=NP
  - The Traveling Salesman problem again

Polynomial Time (P)

- Most of the algorithms we have looked at so far have been **polynomial-time algorithms**
- On inputs of size n, their worst-case running time is $O(n^k)$ for some constant k
- The question is asked can all problems be solved in polynomial time?
- From what we've covered to date the answer is obviously no. There are many examples of problems that cannot be solved by any computer no matter how much time is involved
- There are also problems that can be solved, but not in time $O(n^k)$ for any constant k

Non-Polynomial Time (NP)

- Another class of problems are called NP problems
- These are problems that we have yet to find efficient algorithms in Polynomial Time for, but given a solution we can verify that solution in polynomial time
- Can these problems be solved in polynomial time?
- It has not been proved if these problems can be solved in polynomial time, or if they would require superpolynomial time
- This so-called P != NP question is one which is widely researched and has yet to be settled

Tractable and Intractable

- Generally we think of problems that are solvable by polynomial time algorithms as being tractable, and problems that require superpolynomial time as being intractable.

- Sometimes the line between what is an ‘easy’ problem and what is a ‘hard’ problem is a fine one.

- For example, “Find the shortest path from vertex *x* to vertex *y* in a given weighted graph”. This can be solved efficiently without much difficulty.

- However, if we ask for the longest path (without cycles) from *x* to *y*, we have a problem for which no one knows a solution better than an exhaustive search

Deterministic v Non-Deterministic

- Let us now define some terms
  - **P:** The set of all problems that can be solved by deterministic algorithms in polynomial time

- By *deterministic* we mean that at any time during the operation of the algorithm, there is only one thing that it can do next

- A nondeterministic algorithm, when faced with a choice of several options, has the power to “guess“ the right one.

- Using this idea we can define NP problems as,
  - **NP:**The set of all problems that can be solved by nondeterministic algorithms in polynomial time.

Is P = NP?

- Obviously, any problem in P is also in NP, but not the other way around
- To show that a problem is in NP, we need only find a polynomial-time algorithm to check that a given solution (the guessed solution) is valid.
- But the idea of nondeterminism seems silly. A problem is in NP if we can ‘guess’ a solution and verify it in polynomial time!!
- No one has yet been able to find a problem that can be proven to be in NP but not in P
- Is the set P = NP? We don’t know. If it is, then there are many efficient algorithms out there just waiting to be discovered.
- Most researchers believe that P != NP, but a proof remains to be shown

NP-Completeness

- **NP-complete** problems are set of problems that have been proved to be in NP
- That is, a nondeterministic solution is quite trivial, and yet no polynomial time algorithm has yet been developed.
- This set of problems has an additional property which does seem to indicate that P = NP
- If any of the problems can be solved in polynomial time on a deterministic machine, then all the problems can be solved in NP(Cook's Theorem)
- It turns out that many interesting practical problems have this characteristic

Examples of NP-Complete

- **Travelling Salesman Problem**: Given a set of cities and distances between all pairs, find a tour of all the cities of distance less than M.
- **Hamiltonian Cycle**: Given a graph, find a simple cycle that includes all the vertices.
- **Partition**: Given a set of integers, can they be divided into two sets whose sum is equal?
- **Integer Linear Programming**: Given a linear program is there an integer solution?
- **Vertex Cover**: Given a graph and an integer N, is there a set of fewer than N vertices which touches all the edges?

Solving These Problems

- At present no algorithms exist that are guaranteed to solve any of the NP-complete problems efficiently
- Remember if we could find one then we could solve all the NP-Complete problems
- In the meantime can we find ‘adequate’ solutions?
- One approach is to seek an approximate solution which may not be the optimal but is close to the optimal
- Another approach is to focus on the average case and develop an algorithm that works for most, but not all, cases

Approximation Algorithms

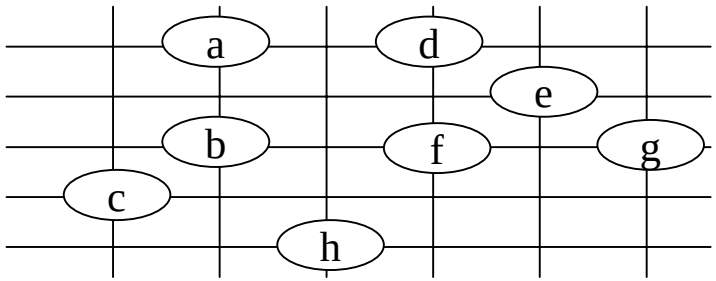
- Here is an approximation algorithm for the travelling salesman problem,

Approx-TSP-Tour(G, c)
- select a vertex r ∈ V[G] to be a “root” vertex
- grow a minimum spanning tree T for G from root r using MST-Prim(G,c,r)
- Let L be the list of vertices visited in a preorder tree walk of T
- return the Hamiltonian cycle H that visits the vertices in the order L

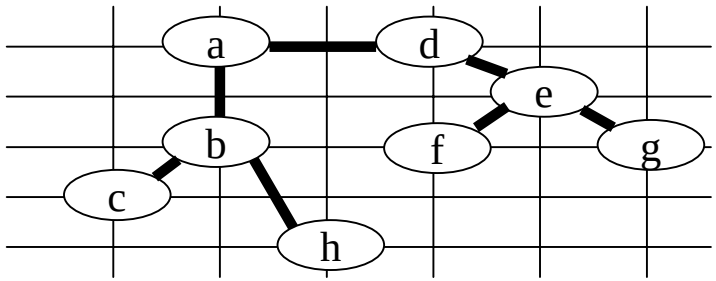
endalg

- This approximation if implemented correctly returns a tour whose cost is not more than twice the cost of an optimal tour

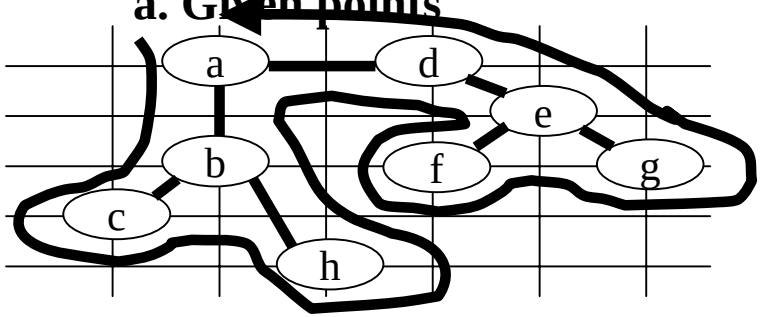
TSP Example



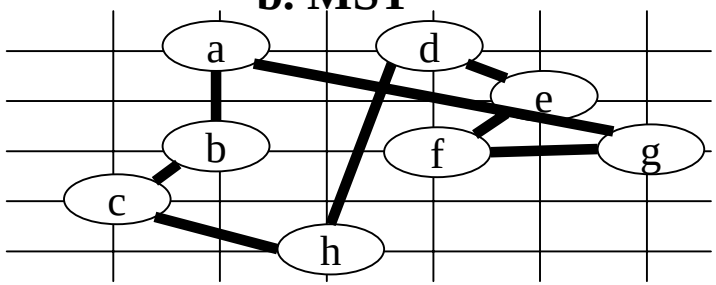
a. Given points



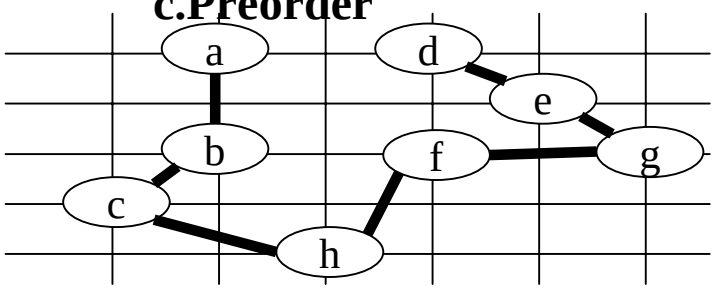
b. MST



c.Preorder



d. Tour by preorder



e. Optimum Tour

Summary

- Polynomial problems are problems for which algorithms can be found that solve the problem in polynomial time
- Non deterministic Polynomial problems are problems for which non-deterministic algorithms can be found
- NP-Complete problems are problems that are in NP, but which have the added property that if one can be solved, they all can be solved
- It is thought that NP-Complete problems may be insoluble using deterministic methods
- One approach is to develop approximation algorithm, as we did with TSP