COMP2230/COMP6230
Algorithms

Lecture 11

Professor Ljiljana Brankovic

Lecture Overview

P and NP:
- Text, Chapter 10

Coping with NP-Completeness
- Text, Chapter 11

Lecture based on:

- Text (R. Johnsonbaugh and M. Schaefer. *Algorithms*.)
- A. Levitin. *Introduction to the Design and Analysis of Algorithms*, Chapter 12.3
- Garey and Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.*

Lecture Overview

- Motivating Examples
- Proving NP-Completeness
- Coping with NP-completeness: restriction, approximation algorithms and heuristics

Example 1

Once again, we consider a travelling salesperson problem.

> Given a map containing cities, roads between cities, and the length of each road, design a minimum length route that starts at a given city, visits each city exactly once and returns to the starting city.

This problem can also be expressed as a decision problem.

> Given a map containing cities, roads between cities, the length of each road, and a parameter k, is there a route that starts at a given city, visits each city exactly once and returns to the starting city, such that the total length of the route is at most k?

Proving NP-Completeness

There are three basic strategies for proving that a problem is NP-complete.

1. Restriction

2. Local Replacement

3. Component Design

Restriction

Restriction is the simplest technique and it is applicable when a special case of our problem is already known to be NP-Complete.

All we need to do is the following:

- Prove that our problem is in NP
- Take an arbitrary instance of a special case A of our problem B (that special case is already known to be NP-complete) and reduce it to an instance of our problem – this is trivial, as it is already an instance of our problem.
- The above two steps together prove that $A \leq_p B$

Example 2. Weighted Vertex Cover.
Given a weighted graph G where every vertex has a weight associated with it, is there a vertex cover of total weight at most k?

Local Replacement

In local replacement, each "component" of the problem is replaced locally by a collection of components; importantly, there is no added interaction between components.

An example of local replacement is the reduction from SAT to $3SAT$.

Another example is reduction from Hamiltonian circuit to TSP.

Example 3. Hamiltonian circuit is known to be NP-complete. Prove that the decision version of TSP is also NP-Complete.

Proof: TSP is in NP, since if we know of a route of size at most k we can verify it in polynomial time.

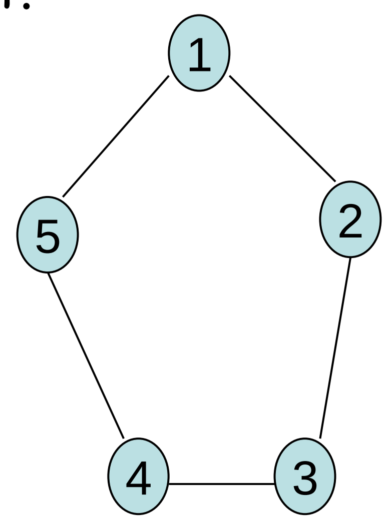
Local Replacement

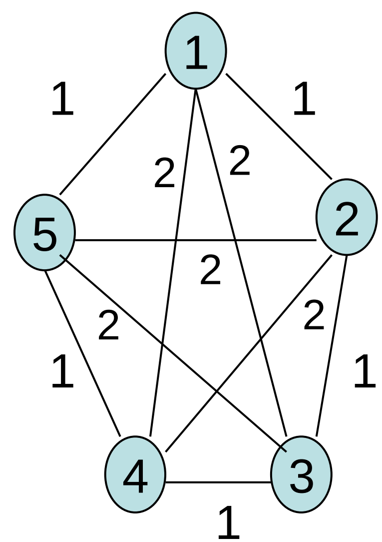
Proof (cont'd):

Take an arbitrary instance of Hamiltonian circuit – a graph $G(V,E)$ and construct an instance of TSP as follows :

- Construct a complete weighted graph $G' = (V,W)$.
- Each vertex in G has a corresponding vertex in G'.
- The edge weights in G' are as follows:
  - $w(i,j) = 1 \;\; if \;\; (i,j) \in E$
  - $w(i,j) = 2 \;\; if \;\; (i,j) \notin E$

Then G' has a route of length $|V|$ if and only if G has a Hamiltonian circuit.





Component Design

Component design refers to the more complicated types of reduction, where there is an interaction between construction components (gadgets).

Example 4. Reduction of $3SAT$ to graph 3-colourability.

Coping with NP-Completeness

The decision version of the TSP problem is known to be NP-complete.

Thus, our hopes to find a polynomial time algorithms for solving travelling salesperson problem are slim.

On the other hand, there exists a brute force algorithm for this problem. One possible approach is to try to make such (exponential) algorithm as efficient as possible. However, we are still limited to small instances.

Techniques for coping with NP-completeness:

1. Approximation algorithms
2. Heuristics
3. Restriction
4. Parameterised algorothms

Approximation algorithms

One possible approach would be to look for an *approximation algorithm*, that runs in polynomial time and finds a solution that is *close to optimal*.

In the case of optimisation version of Travelling Salesperson Problem, such algorithm would find a route that is not necessarily minimum, but it is *close* to minimum.

We need to define more precisely what “close to optimal” means.

Approximation algorithms

Let s_A be a solution obtained by an approximation algorithm, and let s_O be an optimal solution. Further to this, let f be a function that we want to optimise. In the case of Travelling Salesperson Problem we want to optimise (minimise) the total length of the route.

We define the *performance ratio* of the approximation algorithm for a particular solution as

- $R(s_A) = \frac{f(s_A)}{f(s_O)}$ for minimisation problems, and
- $R(s_A) = \frac{f(s_O)}{f(s_A)}$ for maximisation problems.

A polynomial *constant factor approximation algorithm* runs in polynomial time and has the performance ratio bounded by a constant for all instances of the problem:

$$R(s_A) \leq const$$

Example 5. *A factor 2 approximation algorithm for a minimisation problem always finds a solution that is at most twice as big as the minimum solution.*

Approximation algorithms

Example 6: Vertex Cover

Given a graph G, is there a subset of vertices (cover) of size k that covers all the edges (that is, every edge has at least one end vertex in the cover) ?

Vertex cover problem is NP-complete; however, there is a factor 2 approximation algorithm that solves the vertex cover problem in polynomial time:

```
V_C = ∅;
while there are still edges left do {
     select an edge uv;
     V_C = V_C ∪ u ∪ v;
     delete vertices u and v and all their incident edges}
```

Interestingly, this is the best known constant factor approximation algorithm.

Approximation algorithms

Unfortunately, for some problems there does not exist any constant factor approximation algorithms unless $P = NP$, e.g., for the Travelling Salesperson Problem.

Theorem: If $P \neq NP$, then there does not exist a constant factor approximation algorithm for the Travelling Salesperson Problem.

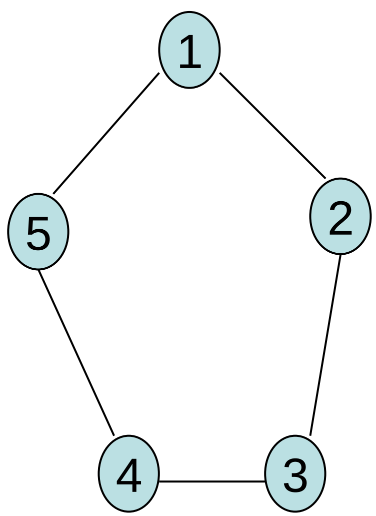
Proof: We use a proof by contradiction. Assume that such an algorithm exists, and that $R(s_A) \leq c$ for all instances of the problem. We show that then we could use the algorithm to solve Hamiltonian circuit problem in polynomial time, which would imply that $P = NP$.

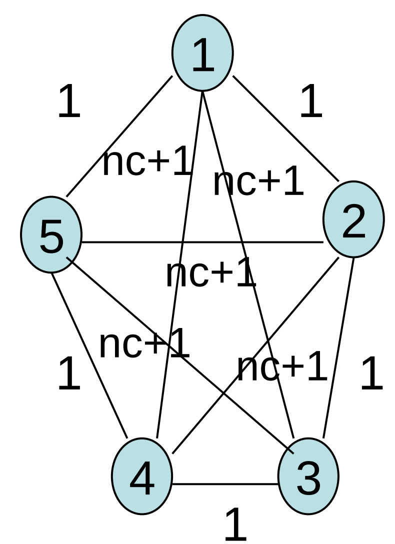
Approximation algorithms

Proof (cont'd): Let a graph G with n vertices be an instance of the Hamiltonian circuit problem.

We map G into an instance of Travelling Salesperson Problem, that is, we map G into a complete weighted graph G', in the following way.

We assign weight 1 to each edge in G' that also exists in G, and weight $cn + 1$ to each edge in G' that does not exist in G.





Approximation algorithms

Proof (cont'd): If there exists a Hamiltonian circuit in G, then the optimal solution s_O to the Travelling Salesperson Problem in G' is that Hamiltonian circuit and its total length $f(s_O)$ is n.

Then the approximate solution s_A obtained by the constant factor approximation algorithm has the total length $f(s_A) \leq cn$, which implies that s_A does not contain any edge which does not exist in G, as the weight of each such edge is $nc + 1$.

If G does not contain Hamiltonian circuit, then the shortest tour in G' has to contain at least one edge which is not in G and thus $f(s_a) \geq cn + 1$.

Thus we can solve the Hamiltonian Circuit problem in polynomial time by first constructing a graph G' and then applying the constant factor approximation algorithm for TSP to it; if the total length of the approximate solution is less than or equal to cn then G has a Hamiltonian circuit; and if the total length is at least $\mathrm{cn} + 1$, then G does not have a Hamiltonian circuit.

Heuristics

In addition to approximation algorithms, one can also use heuristics when faced with problems with no known polynomial time algorithms. A heuristic is an algorithm that is based on exploration and trial and error. <u>Heuristics typically do not have performance guarantees, that is, heuristics cannot guarantee that the solution will always be close to optimal.</u>

There are some heuristic approaches that are not problem specific but rather applicable to different problems; they include local search, tabu search, genetic algorithms and simulated annealing.

Heuristics

Example 7. The following is a heuristic for the TSP.

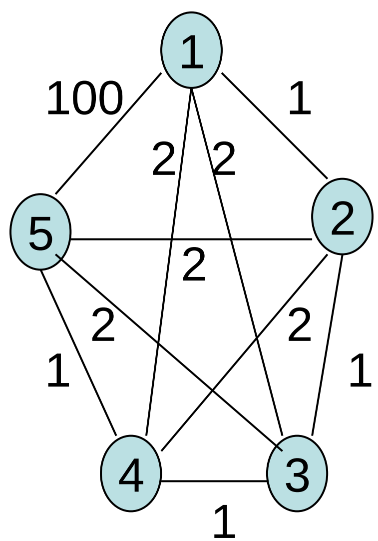
Algorithm A

```
Arbitrarily choose the start city;
While there are unvisited cities do {
   Go to a nearest unvisited neighbour of the current city}
```

This heuristic is actually a greedy algorithm!

Heuristics

The following is an example where the above greedy algorithm would not perform very well. If we pick the vertex 1 as the starting point, then the heuristic would yield the tour of length 104, while the minimum tour is 7.



Restriction

Yet another way to deal with NP-complete problems and their corresponding optimisation problems is to restrict them to special cases which do have either a polynomial time algorithm or a polynomial time constant factor approximation algorithm.

Example 8. We can restrict the TSP instances to Euclidean, i.e., to those where for any 3 cities i, j and k, we always have

$$d(i,j) \leq d(i,k) + d(k,j),$$

In other words, the direct road between two cities is no longer than a road going through another city.

Restriction

If we restrict instances of the TSP to Euclidean instances, then there exist a constant factor approximation algorithm for solving this problem.

Algorithm A

1. Find a minimum spanning tree of the given graph (this can be done in polynomial time).
2. Arbitrarily select a vertex and then use a Depth First Search to perform a walk around the minimum spanning tree, traversing each edge exactly once; record in a list the traversed vertices.
3. Go through the list of traversed vertices and remove all but first occurrence of the same vertex. Vertices remaining on the list form a Hamiltonian circle. Note that removing repeated occurrences of vertices corresponds to taking a "shortcut" in the walk around the tree. Because graph is Euclidean, a shortcut will always be at most as long as the part of the walk it replaces.

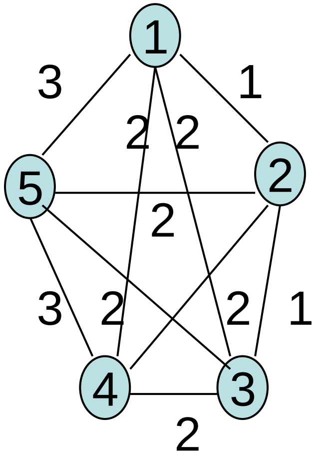
Restriction

The above algorithm finds a TSP tour that is at most twice as long as the minimum tour.

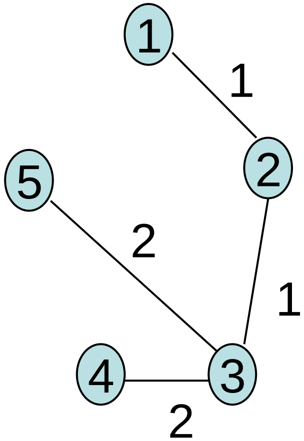
First of all, the total length of a minimum spanning tree is less than the total length of TSP tour; if that were not the case, we could simply delete an edge from the TSP tour and obtain a spanning tree that whose total length is smaller than that of a minimum spanning tree – a contradiction.

Restriction

Example 9



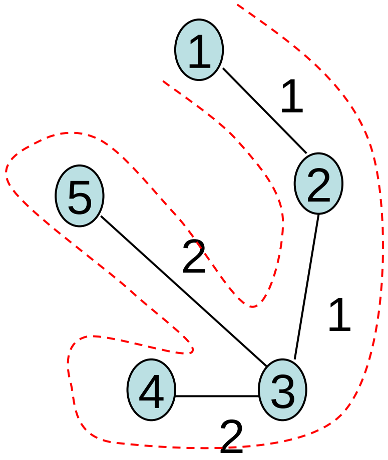
1. Find a minimum spanning tree of the given graph (this can be done in polynomial time).



Restriction

Example 11 :

2. Arbitrarily select a vertex and then use a Depth First Search to perform a walk around the minimum spanning tree, traversing each edge exactly once; record in a list the traversed vertices.

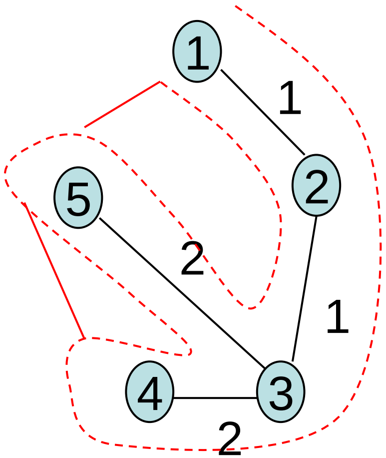


List of traversed vertices: 1,2,3,4,3,5,3,2,1

Restriction

Example 12:

3. Go through the list of traversed vertices and remove all but first occurrence of the same vertex. Vertices remaining on the list form Hamiltonian circle.



$1, 2, 3, 4, \underline{3}, 5, \underline{3}, \underline{2}, 1$

(the repeated occurrences to be removed are underlined)

Thus TSP tour is $1, 2, 3, 4, 5, 1$, and it is not longer than twice the minimum.

Parameterization

- Recall that we can always apply brute force, but . . . we will only be able to solve very small instances of the problem.
- Parameterized complexity focuses on instances that are not necessarily small, but have a small *parameter*.
- For example, a relevant parameter in the vertex cover problem would be the size k of the cover. If k is small we might be able to solve the problem instance efficiently, regardless of how big the instance itself is.

Parameterization

- We can always check whether a given graph G contains a vertex cover of size k by selecting a subset of k vertices and removing it from the graph, and then checking whether there are any edges left:
  - if no, the removed k-subset was indeed a vertex cover (not necessarily minimal!), and the answer to our question is yes, G has a vertex cover of size k;
  - if yes, we proceed by selecting another k-subset of vertices in G.

Parameterization

- There are in total $\binom{n}{k}$ subsets of size k out of n

  vertices, and $\binom{n}{k} \in \mathrm{O}(n^k)$

- Checking whether there are any edges left after removing the k-subset takes $\mathrm{O}\big((n-k)^2\big)$ – we can further improve on this if we use adjacency lists instead of adjacency matrix.

- Thus the total time taken by our algorithm is $\mathrm{O}(n^{k+2})$

Parameterization

- If k is small and we can treat it as a constant, then $O(n^{k+2})$ in polynomial in n ! ! !

- However, since we have n^k we can deal only with very small values for k.

- Ideally, we would hope to find an algorithm with a running time $O(f(k)n^c)$, where c is a small constant – does not depend on neither n nor k. The function $f(k)$ can be even exponential – we can deal with that when k is small.

- Problems for which such an algorithm exists are called

  Fixed Parameter Tractable ($\boldsymbol{FPT}$ ***).***

Algorithm 11.4.4 Vertex Cover

This algorithm determines whether a graph $G = (V, E)$ has a vertex cover of size at most k.

Input Parameter: $G = (V, E)$
Fixed Parameter: k
Output Parameters: $None$

```
vertex_cover(G,k) {
   if ((k == 0) || (E == ∅))
        return E == ∅
   else {
        pick first e = (u,v) in E
               G1  = (V-{u}, E-{(u,w) | w ∈ V})
               G2  = (V-{v}, E-{(v,w) | w ∈ V})
               return vertex_cover(G1, k-1) || vertex_cover(G2, k-1)
   }
}
```

Complexity: $O(2^k(|V| + |E|))$

Thus Vertex Cover is FPT.

Example 13: