P and NP
Lecture 25

COSC 242 – Algorithms and Data Structures

Today’s outline

1. Hard problems
2. Hamilton cycle problem
3. Problem classes
4. NP-Complete
5. Travelling Salesman problem

Today’s outline

1. Hard problems
2. Hamilton cycle problem
3. Problem classes
4. NP-Complete
5. Travelling Salesman problem

Hard problems

In COSC242, pretty much all of the problems and algorithms we've covered so far have been solved in a polynomial number of steps. For example, $O(n^2)$.

That is, we have primarily been concerned with how *easy* a problem is. We have focused on the upper bounds of problem complexity using Big-Oh notation.

Hard problems

Are there problems that are harder than polynomial ones?

Yes, there are many problems that are definitely harder. For example:

- listing all possible subsets of a given set
- determining if there is a winning strategy in generalised games (chess, go etc) requires exponential time. Generalised games have arbitrary sized boards.

Easy or Hard?

There are also a whole class of problems for which we don't know if any efficient algorithms exist. Here are some examples:

- finding a Hamilton cycle in a graph
- factoring an integer into a product of primes
- finding a tour of minimum distance that visits a set of cities once only. Called the travelling salesman problem
- 0-1 knapsack problem (decision problem form) and bin-packing
- timetabling

Easy or Hard

To date, no algorithms that generate a solution to these problems in polynomial time on a deterministic machine have been identified.

These problems fall into a broad category known as “NP”, or “Non-deterministic Polynomial time”.

Understanding the properties of these problems is of interest to researchers in the field of computational complexity theory.

In these NP problems, we are concerned with how *hard* they are, and seek to establish their lower-bound complexity (Big omega, Ω(n)).

Today’s outline

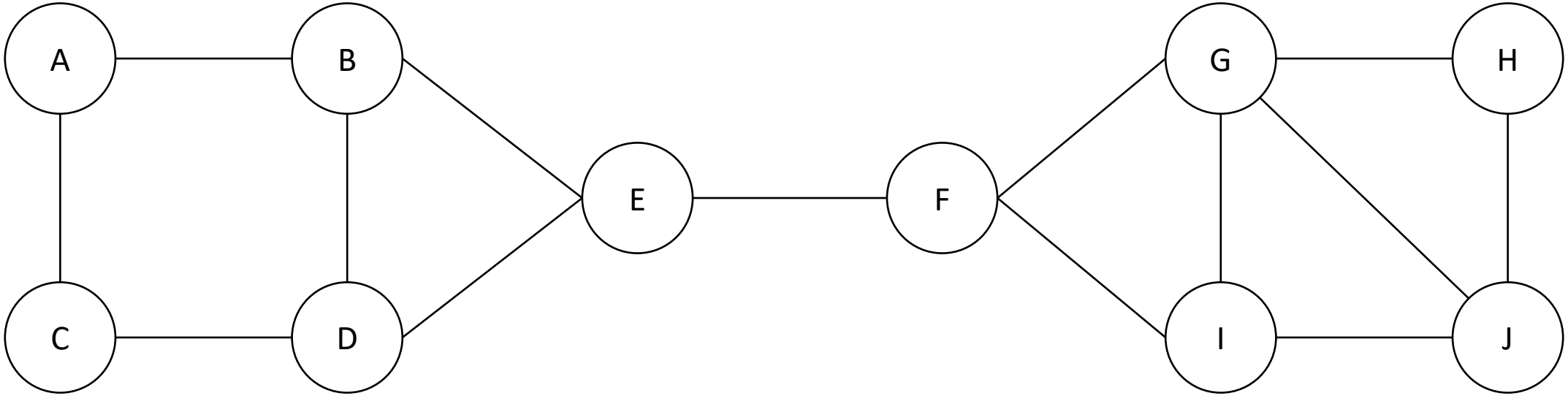
1. Hard problems
2. Hamilton cycle problem
3. Problem classes
4. NP-Complete
5. Travelling Salesman problem

Hamilton path and cycle problems

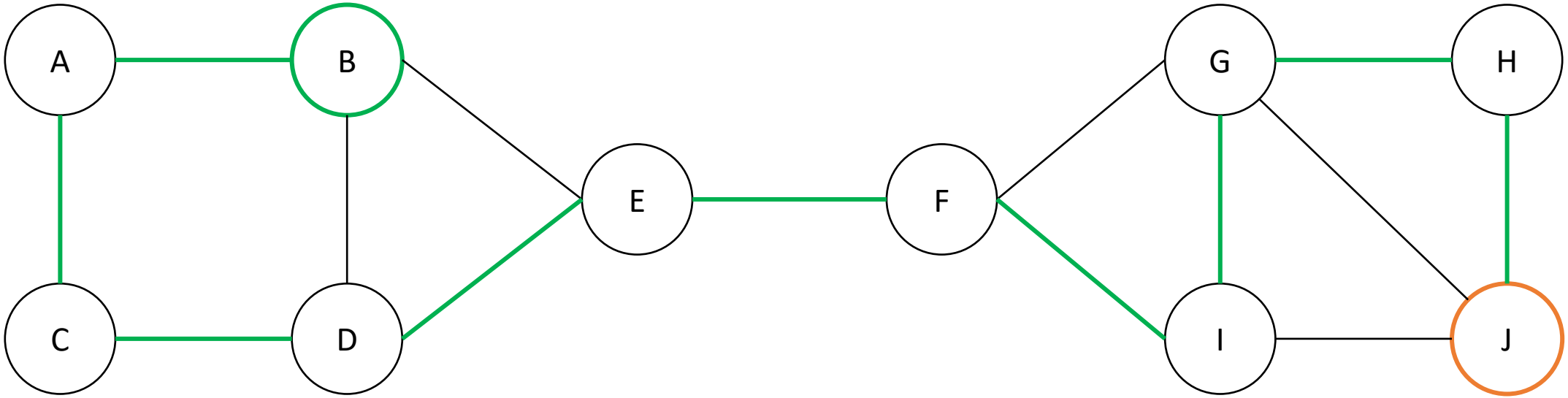
Let *G* be an undirected graph. A Hamilton path in *G* is a sequence of adjacent vertices and distinct edges in which every vertex of the graph appears exactly once.

Relatedly, a Hamilton cycle in G inherits the same requirements, except that the first and last vertices are the same.

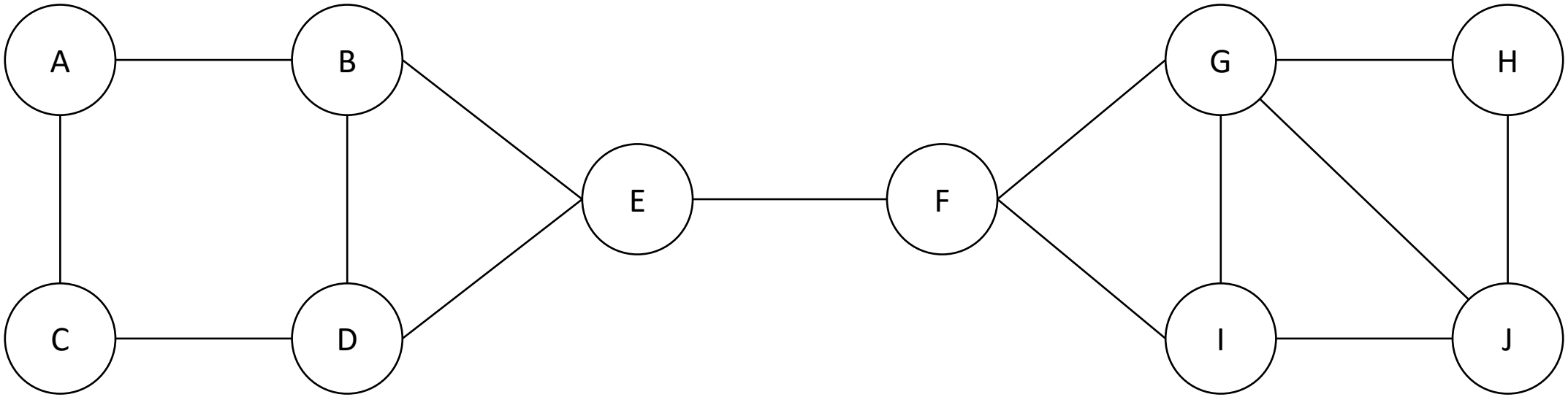
Hamilton Path



Hamilton Path

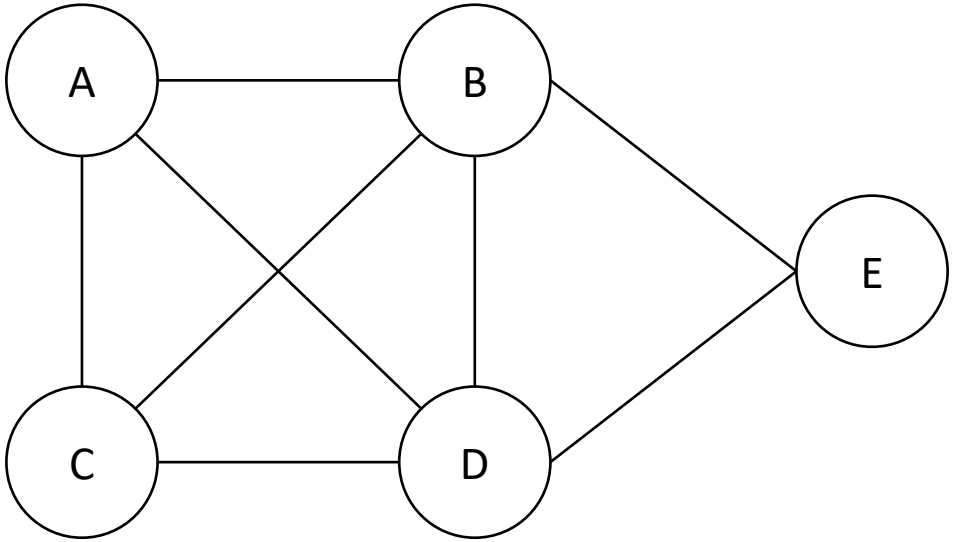


Hamilton Path

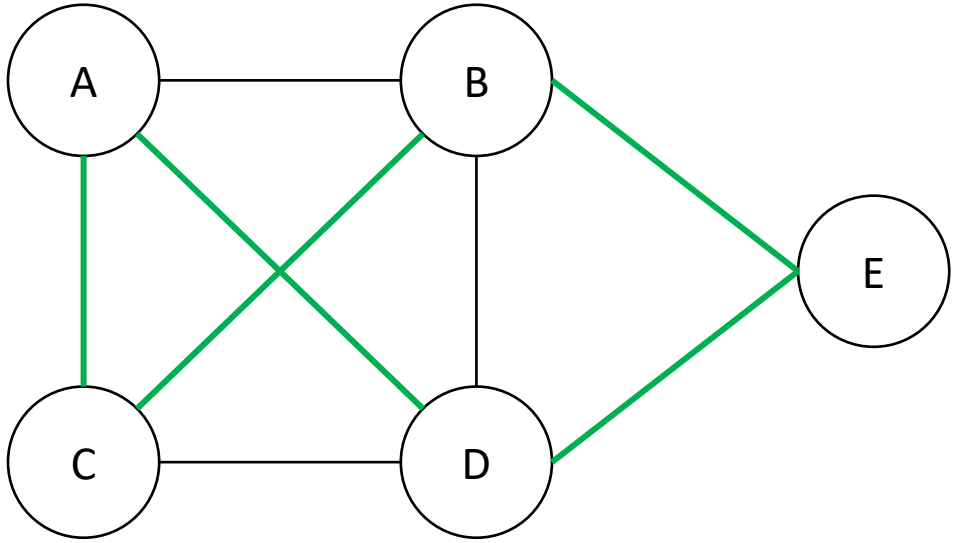


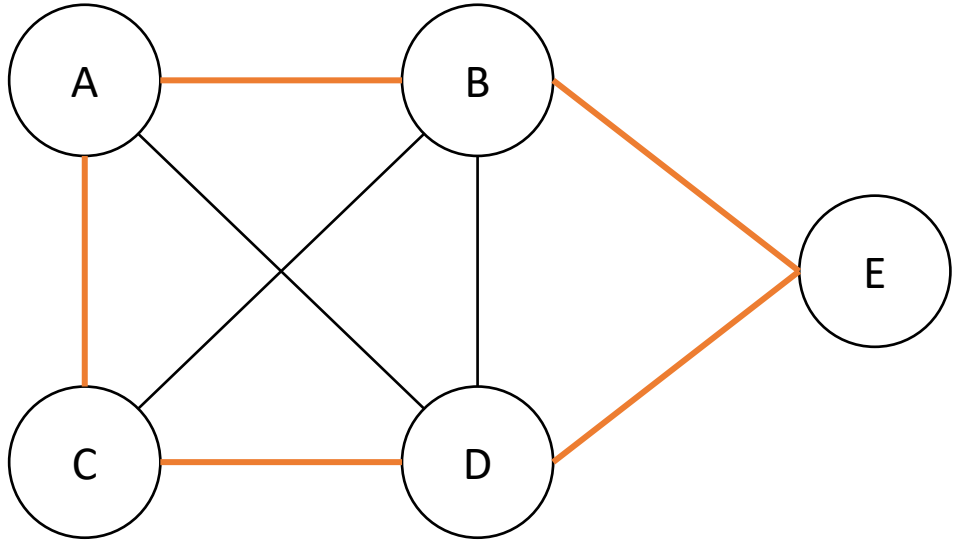
Come up with another Hamilton Path

Hamilton Cycle



Hamilton Cycle





Solving the Hamilton Cycle Problem

The simplest algorithm to solve the HCP is a brute force algorithm:

1. list every permutation of the *n* vertices in *G*
2. for each permutation, check whether *G* has edges connecting the neighbouring vertices.

Solving the Hamilton Cycle Problem

The simplest algorithm to solve the HCP is a brute force algorithm:

1. list every permutation of the *n* vertices in *G*
2. for each permutation, check whether *G* has edges connecting the neighbouring vertices.

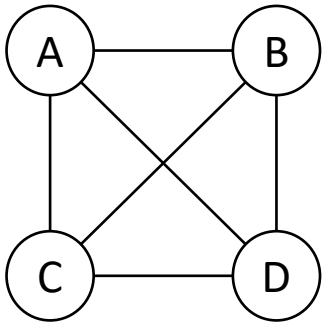
Things to consider

How many permutations of vertices are there?

What is the complexity of the check procedure in step 2?

What is the complexity of this algorithm?

Hamilton cycle problem



Lets try the brute force algorithm on a simpler graph:

We have complete graph K, with *n* edges. Here our graph is K_4.

Lets say we choose A. We can now choose from *n*-1 edges. That is, 3 edges. After that, we can choose from only *n*-2 edges, then *n*-3.

E.g.,: starting at A, we could have:
{A, B, C, D}, {A, B, D, C}, {A, C, B, D}, {A, C, D, B}, {A, D, B, C}, {A, D, C, B}.

So, just starting from A we have 6 possible permutations. That is, (n-1)!. Since we have 4 starting positions, that means we have n!

Today’s outline

1. Hard problems
2. Hamilton cycle problem
3. Problem classes
4. NP-Complete
5. Travelling Salesman problem

The classes P and NP

P is the class of all problems solvable by algorithms that are $\mathrm{O}(\mathrm{n}^{\mathrm{k}})$. That is, problems that are *polynomial*.

NP is the class of problems for which a proposed *solution* is verifiable in polynomial time.

You might think that NP stands for “not polynomial”, but it doesn't. It actually stands for *non-deterministic polynomial*.

The classes P and NP

What does *non-deterministic* mean? A non-deterministic algorithm is one that gives different outputs on different runs, when given the same inputs.

This is in contrast to a deterministic algorithm that always gives the same output on the same input. In COSC242, we have only looked at deterministic algorithms.

These algorithms can be viewed as random decision algorithms. They attempt to “guess” a solution by making random choices.

Why? Because we don’t have a known deterministic solution.

Sudoku – An example of NP

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Rules

All rows and columns must use numbers 1-9. Each 3x3 grid must also only use numbers 1-9

Verify a solution

Easy! We can do this in polynomial time. We just check each row and column, and ensure that each only contains each number only once.

Sudoku – An example of NP

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Generate a solution?

1. Guess a solution.
2. Check if it satisfies the rules.

There is no deterministic algorithm for solving soduku. This problem NP-complete.

Today’s outline

1. Hard problems
2. Hamilton cycle problem
3. Problem classes
4. NP-Complete
5. Travelling Salesman problem

P and NP

We know:

- $P \subseteq NP$
- $HCP \in NP$

P and NP

We know:

- $P \subseteq NP$
- $HCP \in NP$

Easy to see. If a problem is solvable in polynomial time, then its solutions are also verifiable in polynomial time, just by simply solving the problem.

What about?

- is $HCP \in P$?
- is $P \subset NP$?
- Is $P = NP$?

HCP and P

How could we show that $HCP \in P$?

Just provide an algorithm that solves HCP in polynomial time.

No-one has been able to do that yet, so perhaps $HCP \notin P$

How could we show that $HCP \notin P$?

HCP and P

How could we show that $HCP \in P$?

Just provide an algorithm that solves HCP in polynomial time.

No-one has been able to do that yet, so perhaps $HCP \notin P$

How could we show that $HCP \notin P$?

We would have to show that no such algorithm exists that could run in polynomial time. This is much harder to prove, and no-one has been able to prove this either.

In fact, it is one of the Millenium Prize problems worth US$1,000,000 if you can solve it.

NP-Complete problems

HCP is one of several problems that have been shown to be NP-complete.

A problem X is NP-complete if $X \in NP$ and every other problem in NP can be efficiently converted to an example of problem X, so that an algorithm solving X can easily be modified to solve every other problem in NP.

Now imagine if we could find an algorithm of polynomial complexity to solve HCP.

NP-Complete problems

Then every problem in NP can be solved by this algorithm (after modifying it), and so every problem in NP would have a polynomial time solution. Thus it would follow that $P = NP$.

But all problems in NP, like HCP, have stubbornly resisted all efforts to find polynomial time solutions.

By now, everyone strongly suspects that HCP and the other NP - complete problems are too hard to be solved by algorithms that are of less than exponential complexity.

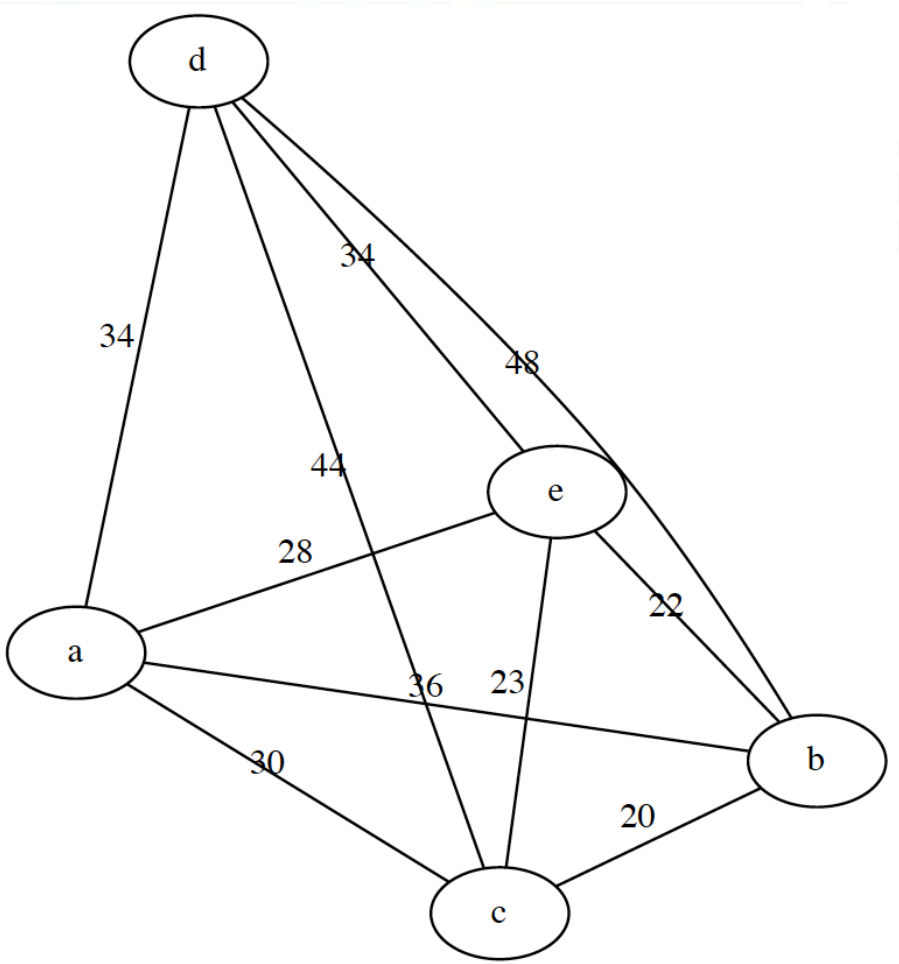
Today’s outline

1. Hard problems
2. Hamilton cycle problem
3. Problem classes
4. NP-Complete
5. Travelling Salesman problem

Travelling Salesman Problem (TSP)

Given a set of cities, a salesman must travel to each city once, starting and ending at the same city. Which route is the shortest?

E.g: starting at a, what is the shortest tour?



Brute Force

Brute force uses essentially the same algorithm as HCP:

1. Generate all possible tours (Hamilton cycles)

2. Return the solution with shortest distance.

Complexity? It's $O(n!)$

Dynamic programming

We can do better by using a dynamic programming approach. Let *G=(V,E)*, and consider a start node *s*. Also, let's define the minimum path between two nodes starting at *a* and finishing at *b* and including all nodes in the set *N* as $m_{tour}(G, a, b, N)$

The minimum tour of *G,* starting at *a* and finishing at *b*, consists of the following recurrence:

$$m_{tour}(G, a, b, N) = \min_{n \in N - a}([d(a, n) + m_{tour}(G, a, b, N - a)]$$

If we start and finish at *s*, we need to solve $m_{tour}(G, a, b, V - s)$. A naive recursive implementation will give us the |V|! solution. If we memoise though, we can do a bit better.

Dynamic programming

How big does our memo array need to be? Here's the recurrence again:

$$m_{tour}(G, a, b, N) = \min_{n \in N - a}([d(a, n) + m_{tour}(G, a, b, N - a)]$$

The only things that change in the recurrence are the start node and the set of nodes to tour through. At the first level of recursion, we look at all subsets of size |V| - 1, then at the second level all subsets of size |V| - 2 etc.

The memo array needs to be big enough to t all possible subsets of the nodes in our graph, and that is $2^{|v|}$.

The complexity of the dynamic programming solution is actually $|V|^2 2^{|v|}$. Well, that's better than factorial, but it's still pretty bad. No algorithm faster than $O(2^{|v|})$ is known to exist.

Suggested reading

Section 34 discusses the problem of NP-completeness and talks about the classes P and NP. The textbook delves into the topic in rather more detail than we do here, so if you're interested, then that's a good place to look.

There are several excellent books on the topic. Two such books are:

- Computers and Intractability, M.R. Garey & D.S. Johnson, 1979 [1].
- The nature of computation, C. Moore & S. Mertens, 2011 [2].

Image attributions

This Photo by Tim Stellmach is licensed under CC0

This Photo by Tim Stellmach is licensed under CC BY-SA 3.0

Disclaimer: Images and attribution text provided by PowerPoint search. The author has no connection with, nor endorses, the attributed parties and/or websites listed above.