# 5CCS2FC2: Foundations of Computing II

# Optimisation and Approximation

## Week 9

**Dr Christopher Hampson**

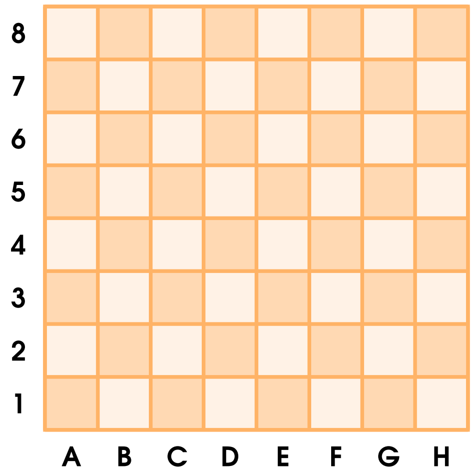
*Department of Informatics*

*King's College London*

## Warm-up : The Eight Queens

- Position **eight Queens** (8 x ♛) on the board so that no Queen is threatened.

  **(no two Queens can appear in the same row, column, or diagonal)**



## Objectives for Today

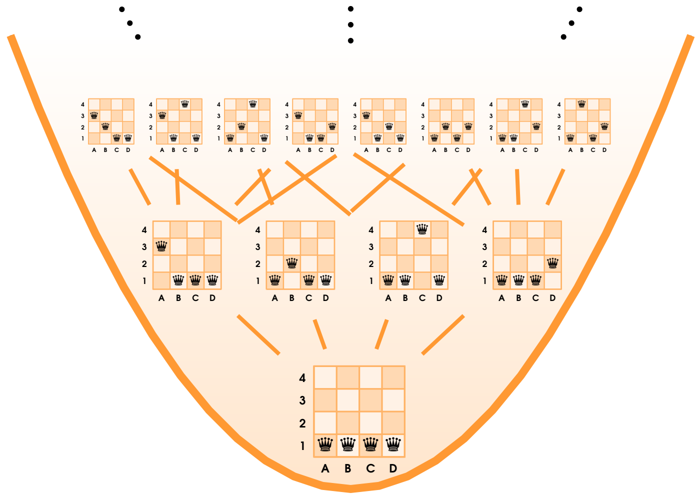
- To be able to explain **local search**, and where it might fail,
    - **Global** vs **Local** maxima / minima,
- To be able to explain what it means for a problem to be **approximable** or **unapproximable**
- To be able to apply the **2-OPT algorithm** for the TSP.

# Local Search

## Local Search

- **State Space** collection of all possible **solutions** and **non-solutions**

  **(*e.g.* all possible ways of placing eights / four queens on a chessboard)**



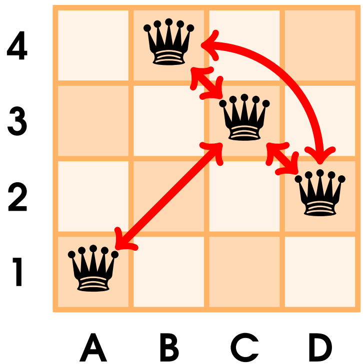

- **Successor Function** All the **'locally' accessible** states

  **(*e.g.* all configurations that differ by one vertical move)**

## Local Search

- **Heuristic Function** Assigns a **'score'** to each state in the state space

$$h : \text{search space} \rightarrow \text{possible score}$$




**Heuristic**
$h(x) = 4$

**(*e.g.* the number of pairs currently in conflict)**

## Local Search

### Hill-climbing Local Search

**Step 1)** Guess an initial configuration,

**Step 2)** Evaluate the heuristic function of the successor states,

**Step 3)** Move to a successor state with a better heuristic ‘score’.

**Step 4)** Repeat until no further improvement to the score are possible.

**(the Greedy SAT algorithm from last week employed Hill-climing)**

## Local Search

- **Potential Pitfalls:** The Hill-climbing search may get **'stuck'** in a local maximum/minumum!

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **8** | 3 | 3 | 3 | 3 | 2 | 3 | ♛ | 3 |
| **7** | 3 | 3 | 4 | 2 | ♛ | 4 | 2 | 4 |
| **6** | 2 | ♛ | 3 | 3 | 5 | 4 | 2 | 3 |
| **5** | 3 | 2 | 4 | ♛ | 4 | 4 | 3 | 2 |
| **4** | 3 | 3 | 4 | 3 | 4 | ♛ | 2 | 3 |
| **3** | 3 | 5 | 3 | 2 | 4 | 3 | 2 | ♛ |
| **2** | 4 | 3 | ♛ | 2 | 2 | 3 | 3 | 3 |
| **1** | ♛ | 3 | 3 | 2 | 2 | 3 | 2 | 3 |
| | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** |

**(there is only one conflict – (D5,G8) – but no local improvements!)**

## Local Search

- The happens with The Eight Queens about **86% of the time!**

  **(...so only successful 14% of the time!)**

- However each paths are typically **quite short**,

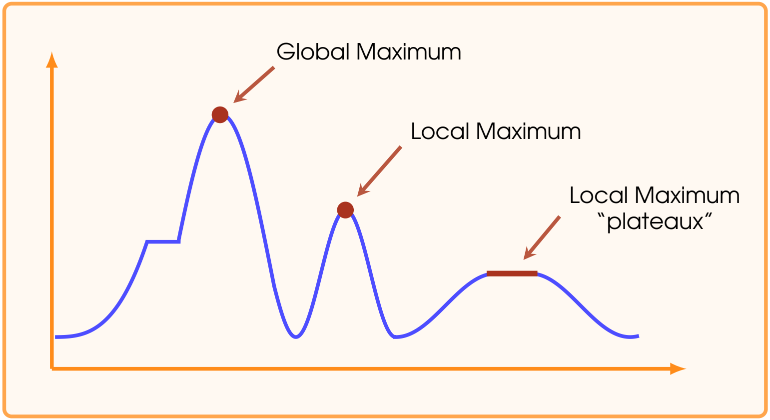
  **(takes about 3 moves on average to get stuck!)**

  - Can quickly **stop** and **re-search** from a random configuration,
  - I was able to get **577 successes** out of **4160 runs** in <20 seconds

# Global and Local Optima

- **Global and Local Maxima**
  - **Global Maximum** $h(x^*) \geq h(x)$ for all $x \in X$

    **($x^*$ is attains the greatest value *anywhere*)**
  - **Local Maximum** $h(x^*) \geq h(x)$ for all *'neighbouring'* $x \in X$




**(similarly, we may define global and local minima)**

# Optimisation Problems

## Decision vs Optimisation Problems

| **Decision Problems** | **VS** | **Optimisation Problems** |
|---|---|---|
| Decide whether a solution *exists* or not. | | Identify the *best* / *optimal* solution. |

- Some problems make sense only as **Decision Problems**
  - **Examples:** The SAT problem, the Hamiltonian cycle problem, the Clique problem, the Graph Isomorphism problem,

    **(typically problems where the answer is yes/no)**

- Others are more natual to consider as **Optimisation Problems**
  - **Examples:** The Travelling Salesman Problem, Vertex-cover problem, the Knapsack problem, *etc.*

## Decision vs Optimisation Problems

- We can **parameterize** any optimisation problem into a decision problem:

**Vertex Cover *Optimisation* Problem**

**Input)** A graph $(V, E)$,

**Output)** A vertex cover with the fewest nodes.

check if optimal solution is smaller than $k$

**Vertex Cover *Decision* Problem**

**Input)** A graph $(V, E)$ and parameter $k$,

**Output)** Is there a vertex cover with fewer than $k$ nodes?

decrease $k$ and repeat

# Optimisation by Local Search

- ## A Search Space for Vertex Cover

  - **State Space** All possible subsets of vertices,

    $$\boxed{\mathcal{P}(V) \;=\; \{U \;:\; U \subseteq V\}}$$

    (*i.e.*, the *powerset* of $V$)

  - **Successor Relation** All sets the differ by a single vertex

    $$\boxed{X \longleftrightarrow Y \quad \text{iff} \quad \big|(X - Y) \cup (Y - X)\big| = 1}$$

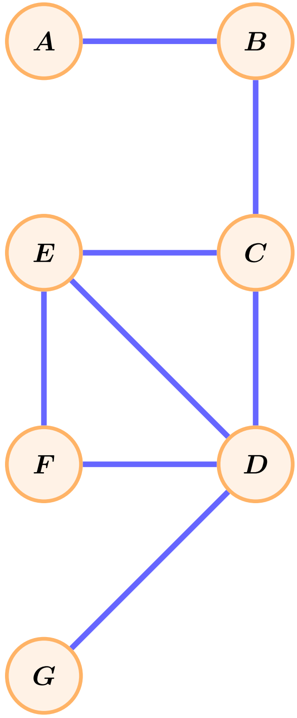
    (for all $X, Y \subseteq V$)

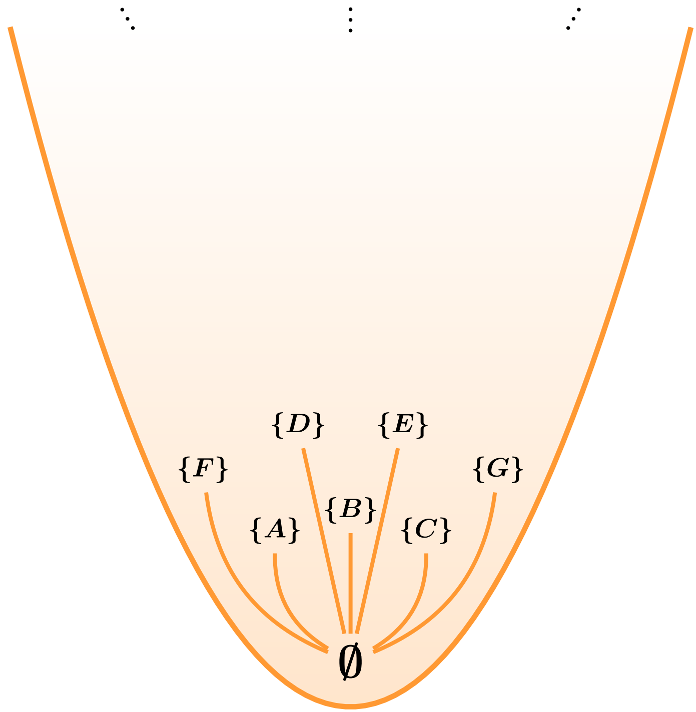
  - **Heuristic Function** The number of edges that are not covered.

    $$\boxed{h(X) \;=\; \big|\{(u, v) \in E \;:\; u \notin X \text{ and } v \notin X\}\big|}$$

    (for all $X \subseteq V$)

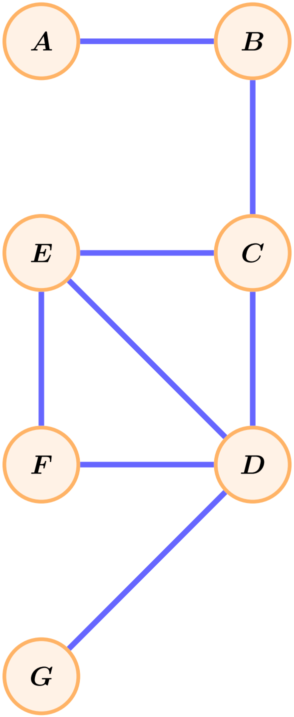
- **Example:**

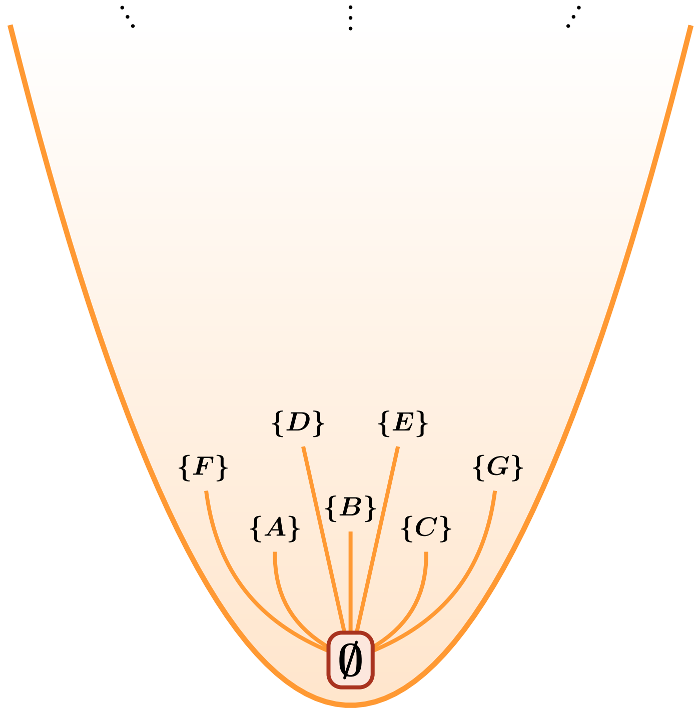


# Optimisation by Local Search

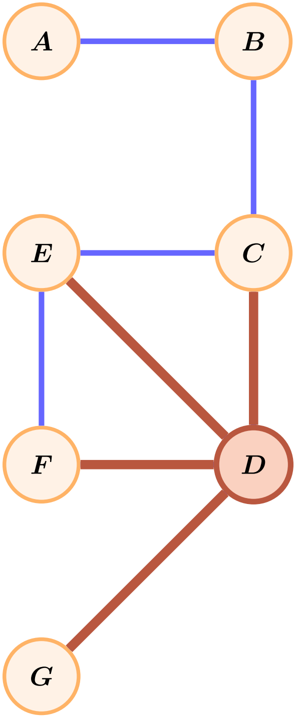
- **Example:**

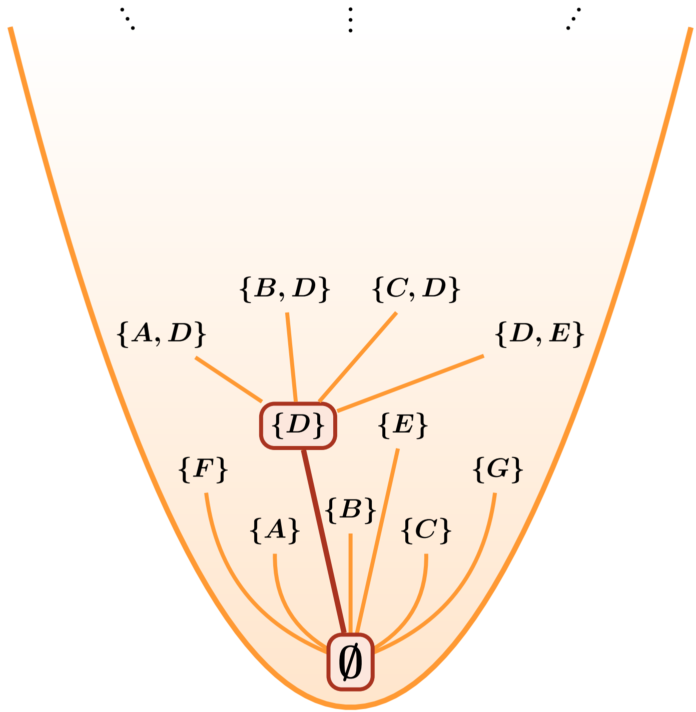


## Optimisation by Local Search

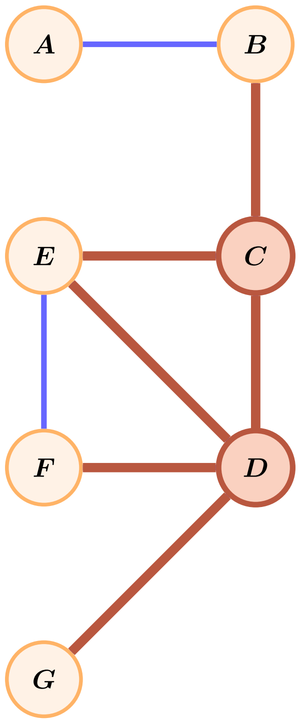
- **Example:**

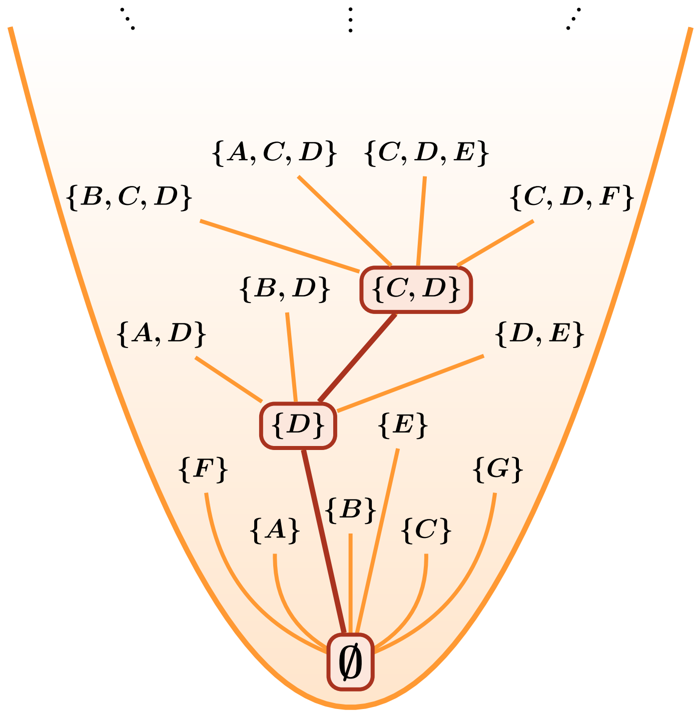


## Optimisation by Local Search

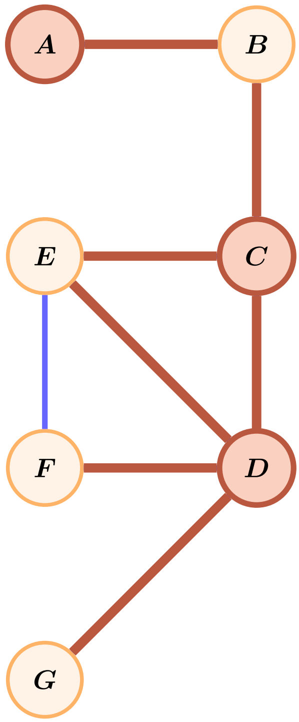
- **Example:**

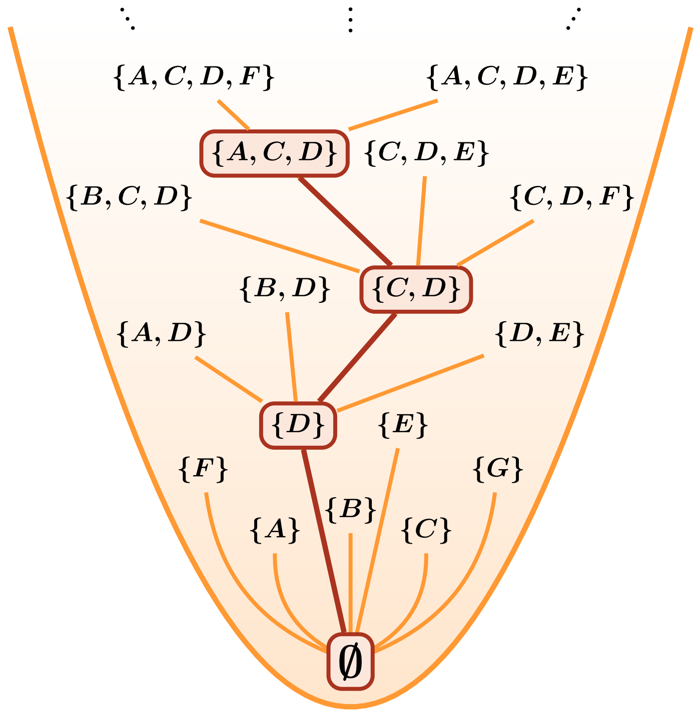


# Optimisation by Local Search

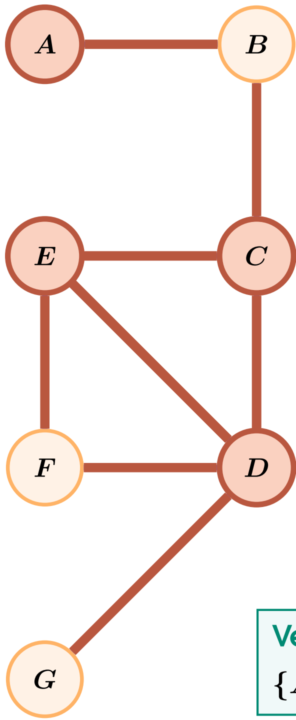
- **Example:**

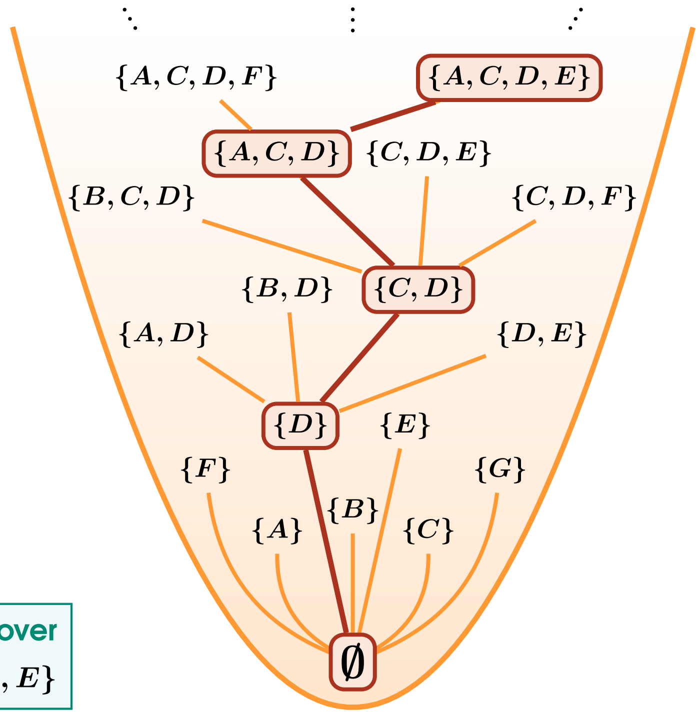


# Optimisation by Local Search

- **Example:**




**Vertex Cover**

$\{A, C, D, E\}$

# Approximate Algorithms

## Approximate Algorithms

- **The Approximation Ratio**

$$\boldsymbol{R} = \max\left(\frac{\boldsymbol{C}}{\boldsymbol{C^*}}, \frac{\boldsymbol{C^*}}{\boldsymbol{C}}\right)$$

**(if $C < C^*$ then $R = C^*/C$, otherwise $R = C/C^*$)**

  - $C$ is the cost of the ***approximate* solution**
  - $C^*$ is the cost of the ***optimal* solution**

- **$R$-approximable problems** There is an **approximate algorithm** such that **every instance** of the problem has an approximation ratio $\leq \boldsymbol{R}$.

**(the algorithhm never returns a solution worse that $R$ times the optimal)**

- **The Travelling Salesman (Optimisation) Problem (TSP):**

  **Input)** A complete weighted graph $(\boldsymbol{V}, \boldsymbol{d})$,

  **($d(x, y)$ is the distance between two points $x, y \in V$)**

  **Output)** The *shortest* Hamiltonian cycle, visiting all nodes.

---

**Theorem** TSP is NP-complete.

**But can find a *'good' approximation* for TSP?**

## The 2-OPT approximation algorithm for TSP

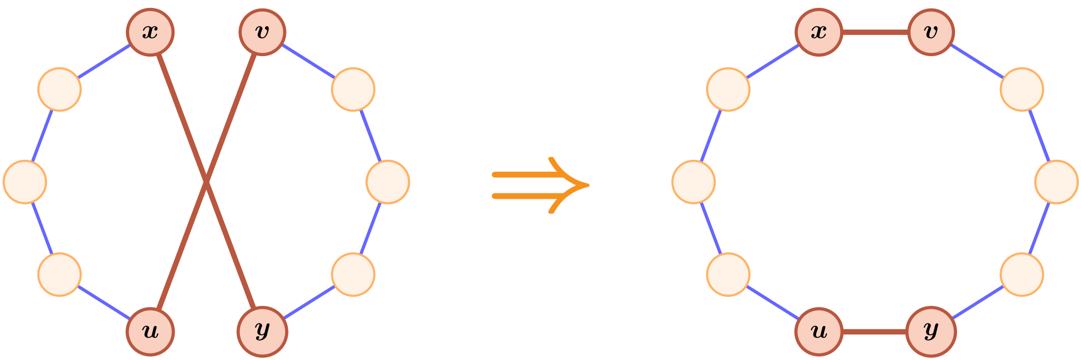
- **The 2-OPT swap move:**

**Step 1)** Given a cycle, choose two *non-adjacent* edges in the cycle:

$$\boxed{(x, y) \quad \text{and} \quad (u, v)}$$

**(where $x$, $y$, $u$, and $v$ are all distinct)**

**Step 2)** Compare the *weight* of $(x, y)$ and $(u, v)$ with the weight $(x, v)$ and $(u, y)$.



# The 2-OPT approximation algorithm for TSP

- **The 2-OPT swap move (cont.):**

  **Step 3)** Replace the two edges $(x, y)$ and $(u, v)$ with $(x, v)$ and $(u, y)$ whenever,

$$\underbrace{d(x,v) + d(u,y)}_{\text{after swap}} < \underbrace{d(x,y) + d(u,v)}_{\text{before swap}}$$

**The new path visits all nodes in a shorter distance!**

## The 2-OPT approximation algorithm for TSP

**Input:** A complete weighted graph $G = (V, d)$,

**Step 1)** Construct a **minimum spanning tree** of the complete weighted graph.

(using *Prim's Algorithm*, for example)

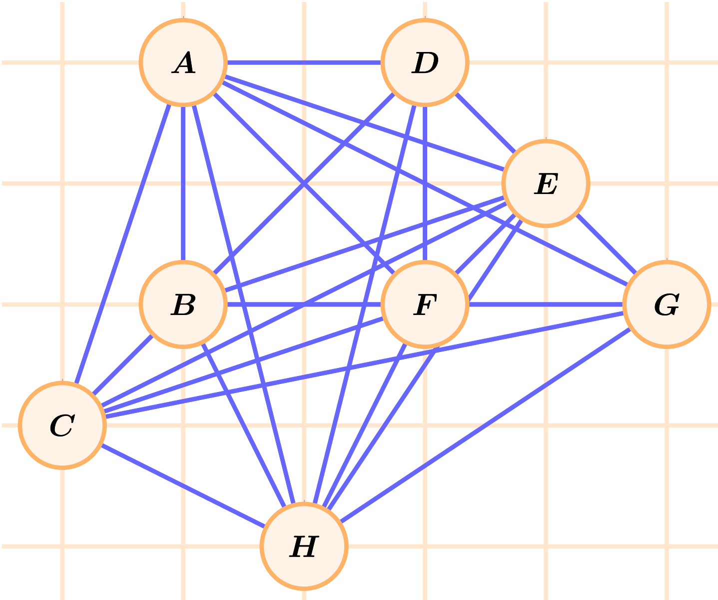
**Step 2)** Use the **preorder traversal** of the spanning tree as an initial cycle.

**Step 3)** Apply the **2-OPT swap move** for each pair of non-adjacent edges.

**Step 4)** **Repeat** until no more optimisations are possible.

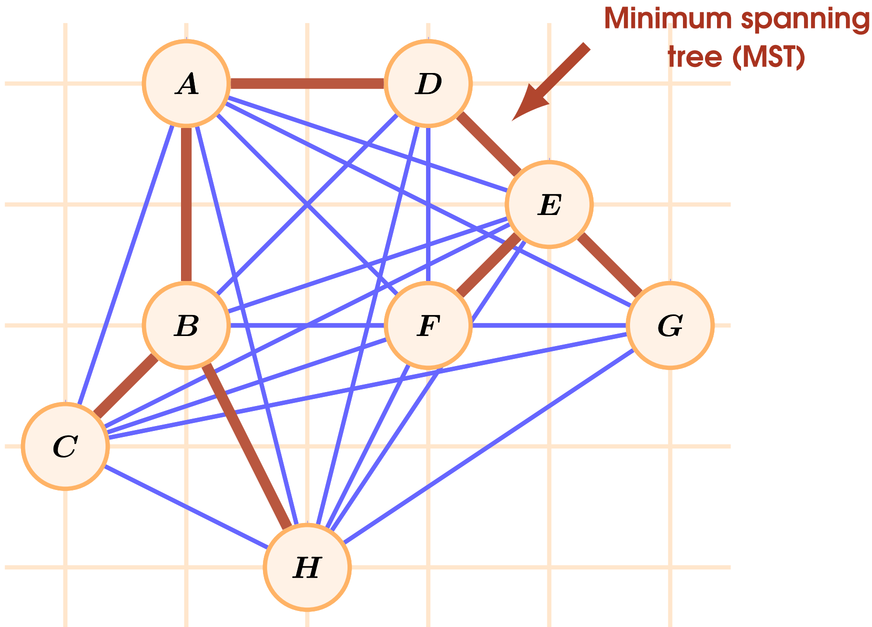
## The 2-OPT approximation algorithm for TSP

- **Example:**



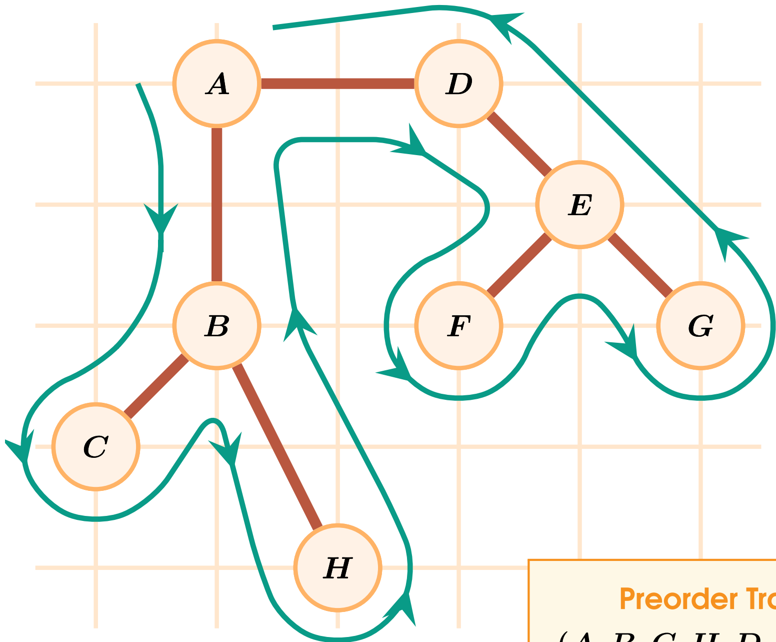
# The 2-OPT approximation algorithm for TSP

- **Example:**



# The 2-OPT approximation algorithm for TSP

- **Example:**

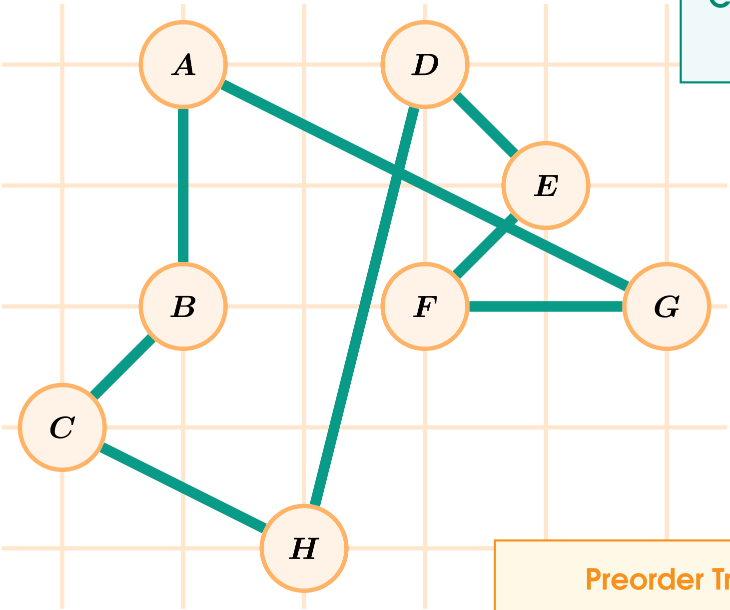



**Preorder Traversal**

$(A, B, C, H, D, E, F, G, A)$

## The 2-OPT approximation algorithm for TSP

- **Example:**

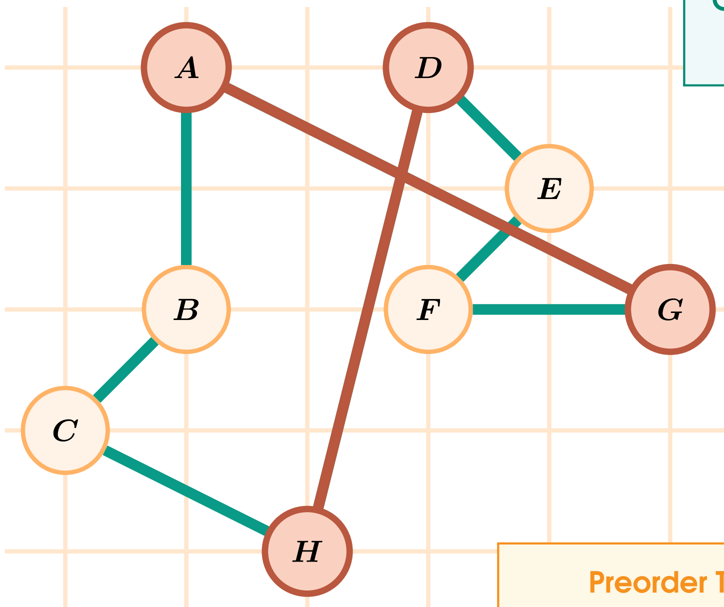



Cycle Length
$\approx 19.06$

Preorder Traversal
$(A, B, C, H, D, E, F, G, A)$

# The 2-OPT approximation algorithm for TSP

- **Example:**

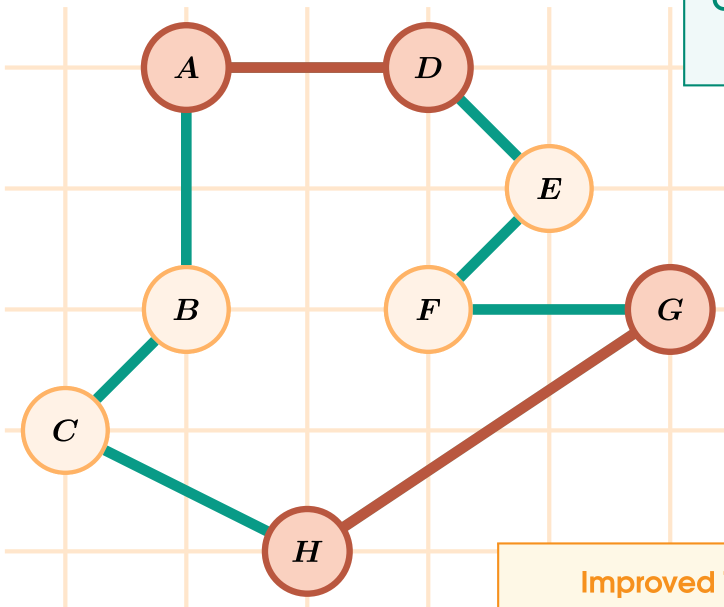



Cycle Length
$\approx 19.06$

Preorder Traversal
$(A, B, C, H, D, E, F, G, A)$

# The 2-OPT approximation algorithm for TSP

- **Example:**

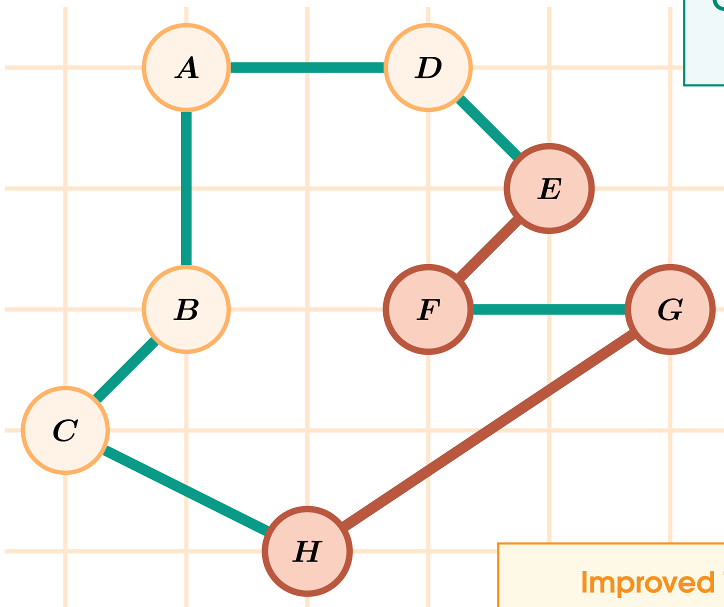



**Cycle Length**
$\approx 16.08$

**Improved Traversal**
$(A, B, C, H, G, F, E, D, A)$

## The 2-OPT approximation algorithm for TSP

- **Example:**

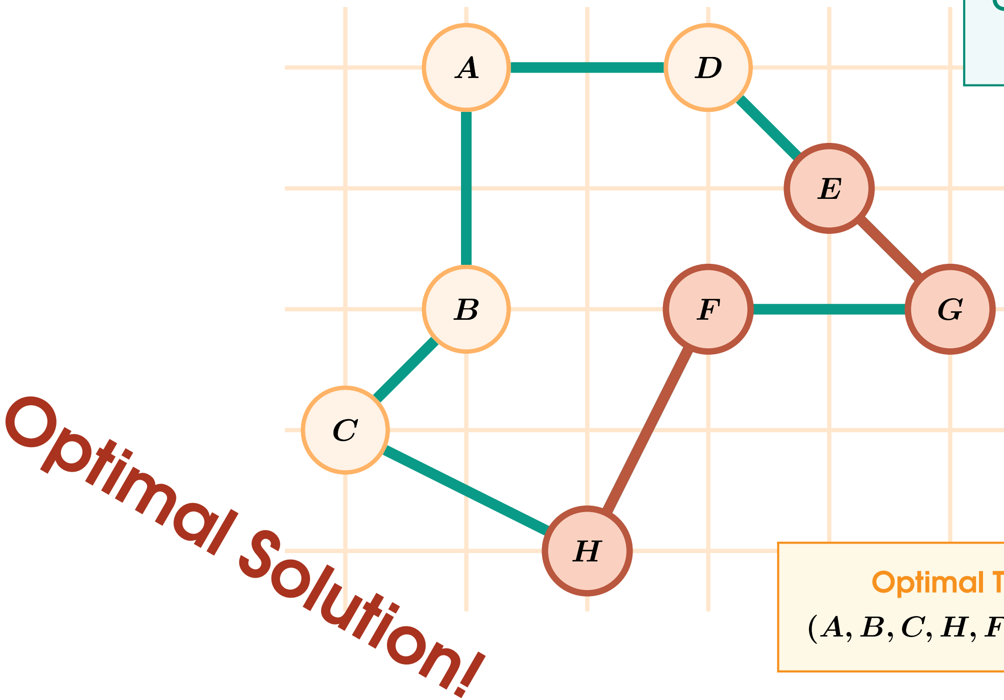



**Cycle Length**
$\approx 16.08$

**Improved Traversal**
$(A, B, C, H, G, F, E, D, A)$

## The 2-OPT approximation algorithm for TSP

- **Example:**




**Cycle Length**
$\approx 14.71$

**Optimal Traversal**
$(A, B, C, H, F, G, E, D, A)$

## The 2-OPT approximation algorithm for TSP

- In this instance our **approximate algorithm** was able to find the optimal path!

$$\text{length\_of ( optimal path )} \approx 14.71$$

- In general, for other instances, we may may get stuck in a **local optimum**.

- But how close to the **global optimum** does our approximate algorithm get, when we get stuck?

**What is the Approximation Ratio in the *worst* case?**

## The 2-OPT approximation algorithm for TSP

**Theorem** The cycle found is no worse than **twice** the optimal length!

**Proof:** **Step 1)** Let $H$ and $H^*$ denote the following cycles

$$\begin{aligned} H &= \text{Local optimal found by 2-OPT algorithm} \\ H^* &= \text{Global optimal Hamiltonian cycle,} \end{aligned}$$

**(we want to show that length_of $(H) \leq$ 2 $\times$ length_of $(H^*)$)**

**Step 2)** Remove one edge from $H^*$ to create a path $T$,

$$\text{length\_of}\,(T) \;\leq\; \text{length\_of}\,(H^*)$$

**(note that $T$ is a *(trivial) spanning tree*!)**

## The 2-OPT approximation algorithm for TSP

**Step 3)** Let $T_0$ denote the following **minimum spanning tree** computed as the first step of the 2-OPT algorithm,

$$\boxed{T_0 \quad = \quad \text{Minimum spanning tree}}$$

It then follows that

$$\boxed{\text{length\_of}\,(T_0) \;\leq\; \text{length\_of}\,(T)}$$

**(since a *minimum* spanning tree is shorter than another spanning tree!)**

**Step 4)** The length of the ***pre-order traversal*** is at most twice the length of the spanning tree,

$$\boxed{\text{length\_of}\,(W) \;\leq\; 2 \times \text{length\_of}\,(T_0)}$$

**(we walk the length of each edge *twice*—first on the left, then on the right)**

## The 2-OPT approximation algorithm for TSP

**Step 5)** Every application of the 2-OPT swap decreases the length of the cycle, so

$$\text{length\_of}\,(H) \;\leq\; \text{length\_of}\,(W)$$

where $H$ is the local minimum return by the 2-OPT algorithm.

**Step 6)** Therefore

$$\begin{aligned}
\text{length_of}\,(H) &\leq \text{length_of}\,(W) \\
&\leq 2 \times \text{length_of}\,(T_0) \\
&\leq 2 \times \text{length_of}\,(T) \\
&\leq 2 \times \text{length_of}\,(H^*)
\end{aligned}$$

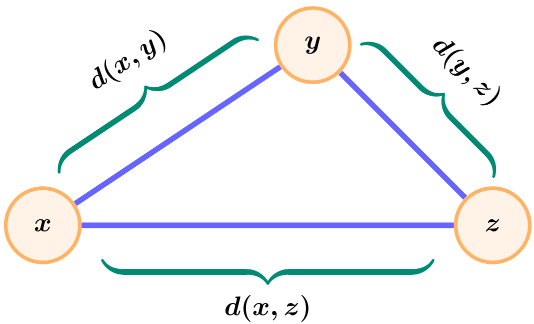
**Q.E.D**

## Unapproximable Instances of TSP

- **Caution!** In the above example, we assumed that our weighted graph was **'reasonable'**.
- We assumed that it satisfied the **triangle inequality**

$$\boxed{d(x,y) + d(y,z) \;\geq\; d(x,z)}$$

**(for all nodes $x, y, z \in V$)**




- If we drop this requirement then there is **no approximate algorithm** for TSP.

# Unapproximable Instances of TSP

**Theorem** If $d$ does not satisfy the **triangle inequality** then TSP is **unapproximable**.

**Proof:**

**Step 1)** Suppose that there is a **polynomial time** approximate algorithm for TSP whose approximation ratio is $R$.

**(so all approxiate solutions are no worse that $R$ times the optimal)**

**Step 2)** Consider a instance of the **Hamiltonian Cycle** problem $\boxed{G = (V, E)}$ and let $n = |V|$ denote the number of vertices in $G$.

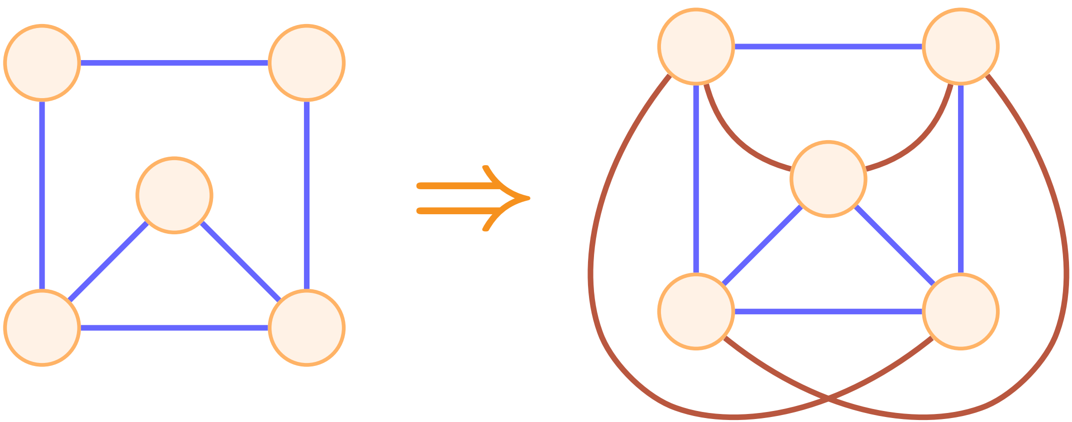
**We will convert the graph $G$ into an instance of TSP and use the approximate algorithm to find a solution for the Hamiltonian Cycle problem**

## Unapproximable Instances of TSP

**Step 3)** Construct a complete weighted graph $(V, d)$ by setting

$$d(x,y) = \begin{cases} 1 & \text{if } (x,y) \in E \\ nR+1 & \text{if } (x,y) \notin E \end{cases}$$

(the edges from the original graph are short)



## Unapproximable Instances of TSP

**Step 4)** Let $H$ be the cycle returned by the approximate algorithm.

We will show that

| | | |
|---|---|---|
| length_of $(H) \leq nR$ | $\Longleftrightarrow$ | $G$ has a Hamiltonian cycle |

**Left-to-Right)** Suppose that $H$ is a *'short'* TSP cycle

$$\boxed{\text{length\_of}\,(H) \leq nR}$$

Then $H$ must use only *'short edges'*!

**('short edges' = length 1)**

**Therefore $H$ is a Hamiltonian cycle in $G$!**

## Unapproximable Instances of TSP

**Right-to-Left)** Suppose that $G$ has a Hamiltonian cycle.

Then there is a TSP path $H^*$ that uses only 'short' edges.

$$\text{length\_of}\,(H^*) \;=\; \underbrace{1 + 1 + \cdots + 1}_{n \text{ nodes}} \;=\; n$$

**($H^*$ must be *optimal* as no shorter paths can visit all $n$ nodes)**

Since our proposed approximate algorithm has an **approximation ratio** of $R$, we must have that

$$\text{length\_of}\,(H) \;\leq\; R \times \text{length\_of}\,(H^*) \;=\; nR$$

**Therefore $H$ is 'short' TSP cycle!**

**Step 5)** This equivalance is a key to **efficiently deciding** whether $G$ has a Hamiltonian cycle.

**A Polynomial Time Algorithm for TSP (?)**

- Convert $G$ into a TSP graph,
- Use polynomial-time approimate algorithm to find $H$,
- If length_of $(H) \leq nR$ then return **TRUE**,
- Else return **FALSE**

**Conclusion)** Either we have just proved that **P** $=$ **NP** ... or ...

**There is no polynomial-time approximate algorithm for TSP.**

**Q.E.D**

## Summary of TSP

- **Summary**
  - If your graph is **'reasonable'** then TSP is **2-approximable**,

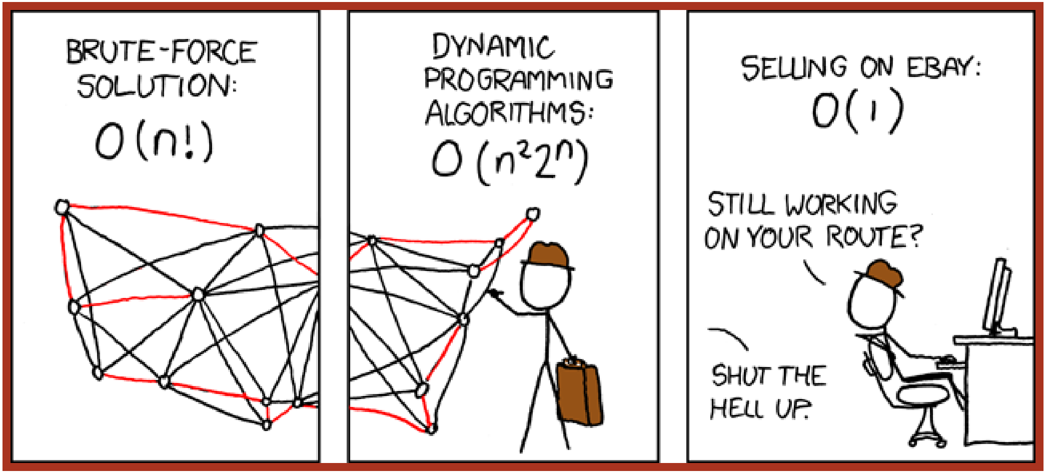
    **('reasonable' = satisfies the triangle inequality)**

  - Otherwise the graph is **unapproximable**.

**Routing to minimise *distance* is approximable...**

**... Routing to minimse *duration* is not!**

# Summary of TSP




https://xkcd.com/399/

# End of Slides!

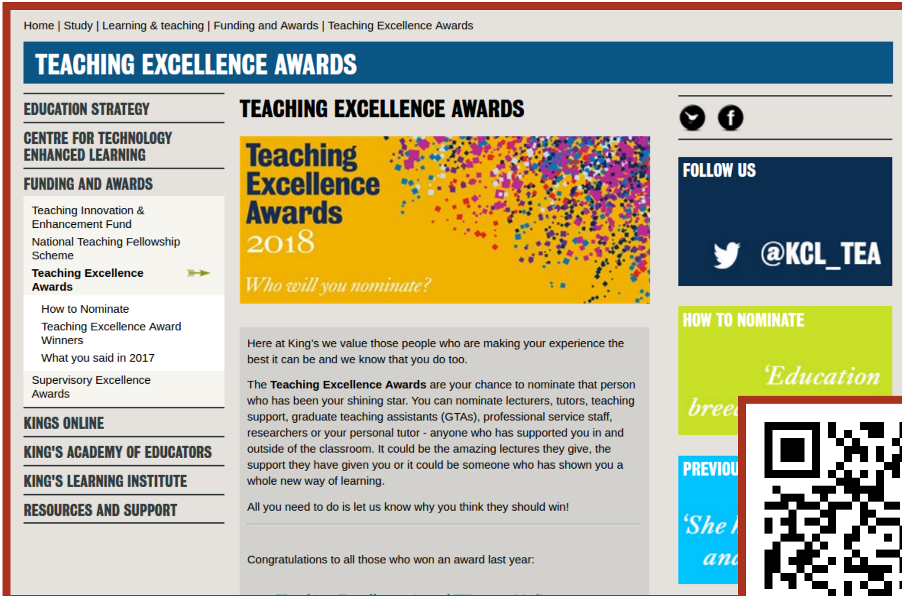
## Feedback

- **Let me know how you found today's lecture?**

https://goo.gl/forms/wHsWHGU9qWskEm8x2

# Teaching Excellence Awards 2018

- **Teaching Excellence Awards 2018**




https://goo.gl/QURKsH