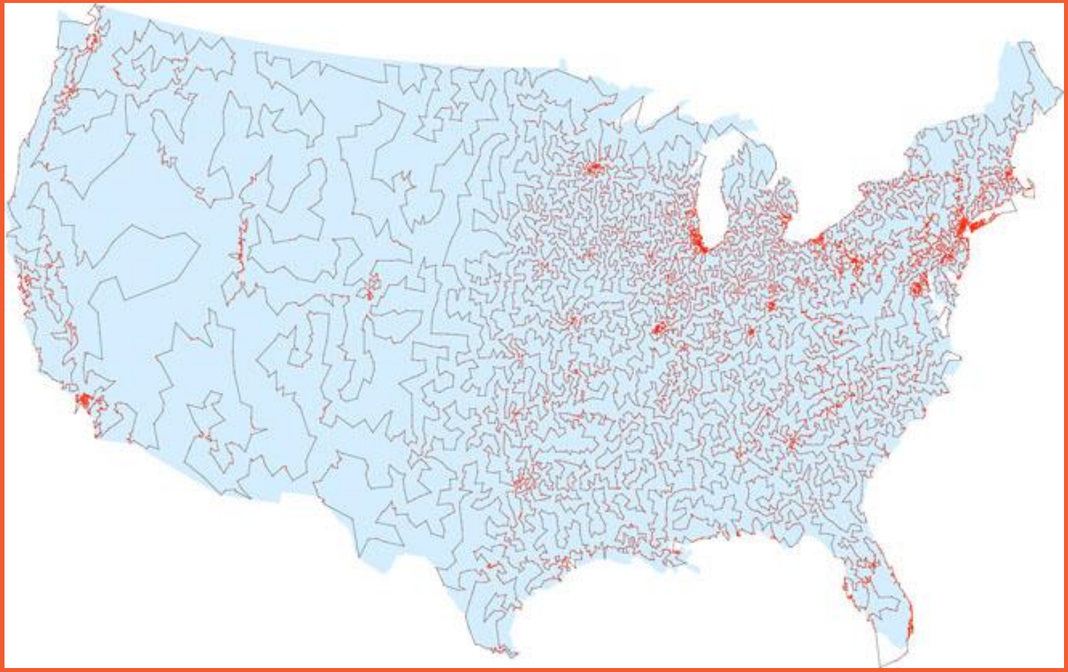
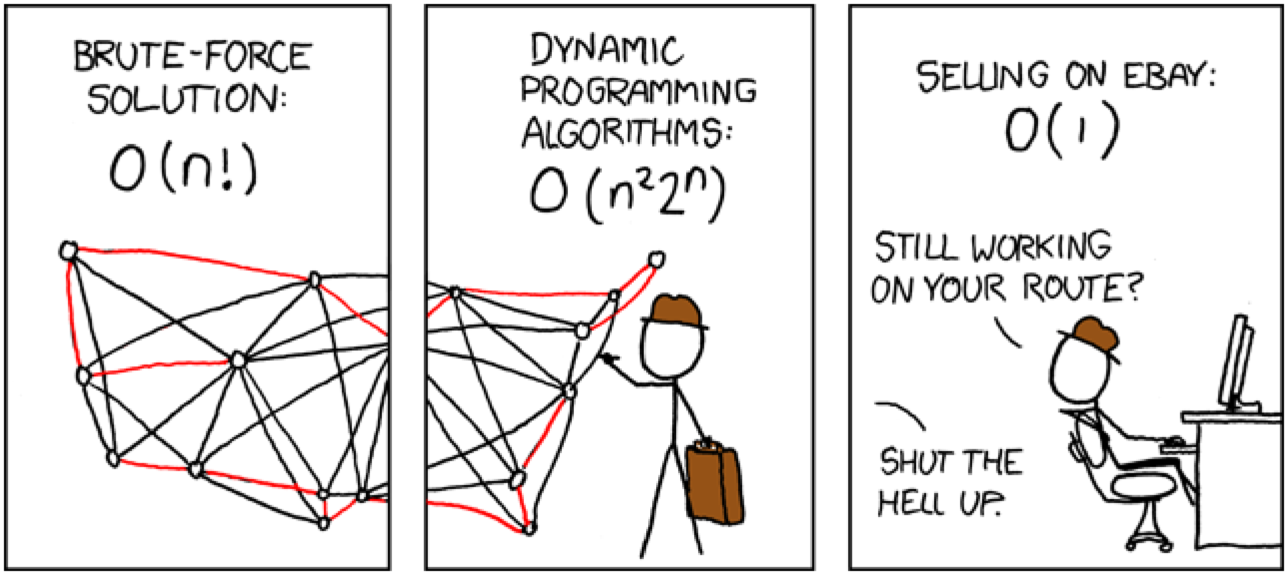
Lecture 7: Dynamic Programming II (Adv.)

Exponential time algorithms



THE UNIVERSITY OF SYDNEY

TSP in exponential time



Exact TSP

Input: An undirected graph $G=(V,E)$ where each edge $(u,v) \in E$ has a positive weight $d(u,v)$.

Aim: Find a bijection $\pi: \{1, \ldots, n\} \to V$ s.t.

$$(*) \quad \sum_{i=1}^{n} d(\pi(i),(\pi(i+1)) + d(\pi(n),\pi(1)) \text{ is minimized.}$$

Note: TSP is a permutation problem; find a permutation π of n vertices that minimizes (*).

Brute force algorithm

Algorithm: Test all possible permutations π of n vertices that minimizes (*).

Running time: There are n! possible permutations, each permutation takes O(n) time to test $\Rightarrow$ O(n! n)

n! -	$20! \approx 10^{18}$	$30! \approx 10^{32}$	$40! \approx 10^{47}$
2^n -	$2^{20} \approx 10^6$	$2^{30} \approx 10^9$	$2^{40} \approx 10^{12}$

TSP

- No polynomial time algorithm is believed to exists.

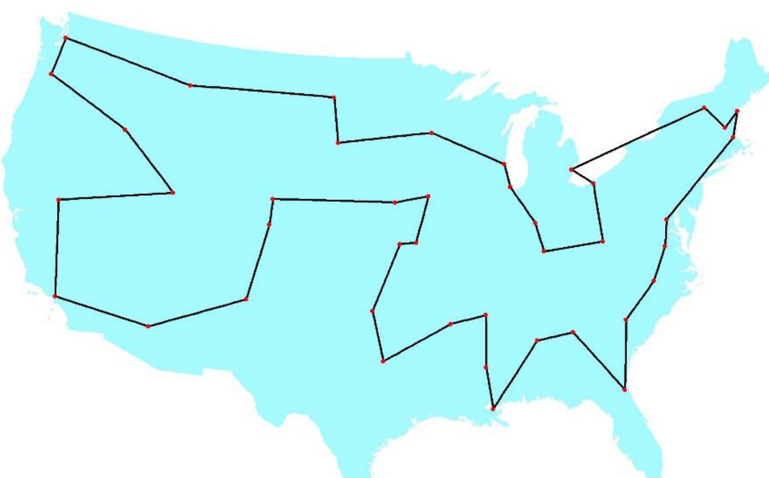
- The fastest known algorithm for TSP was discovered independently by Bellman 1962 and Held & Karp 1962 using dynamic programming.

- Many good heuristics: Lin-Kernigan, k-opt, genetic algorithms...

- Good algorithms for special cases, e.g., $(1+\varepsilon)$-approximation algorithm for Euclidean TSP.

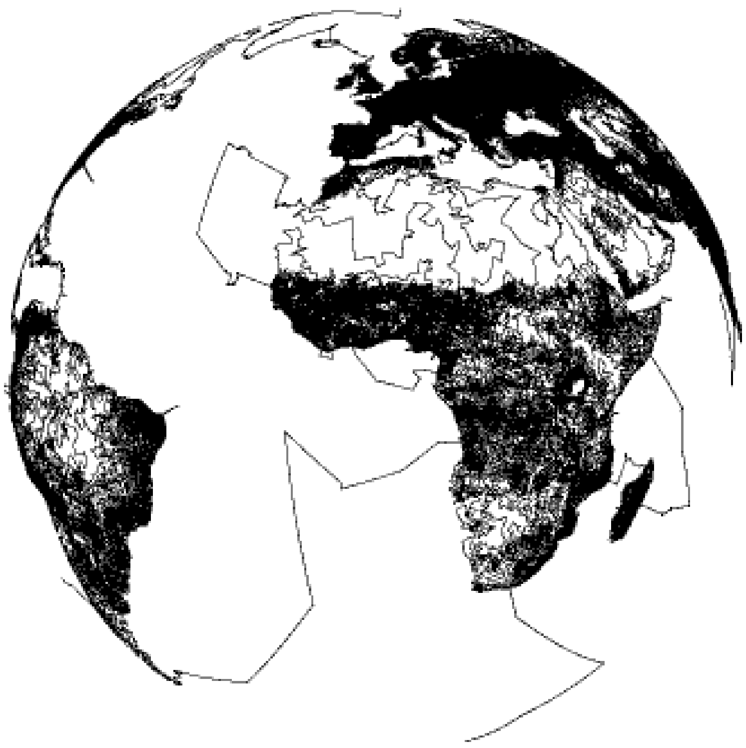
Instances

- 1954 n=49 [Dantzig, Fulkerson, Johnson]
- 1962 n=33
- 1977 n=120
- 1987 n=532
- 1987 n=666
- 1987 n=2392
- 1994 n=7397
- 1998 n=13509
- 2001 n=15112
- 2004 n=24978
- 2006 n=85900



World tour - 1,904,711 cities

Current best tour within 0.0474%

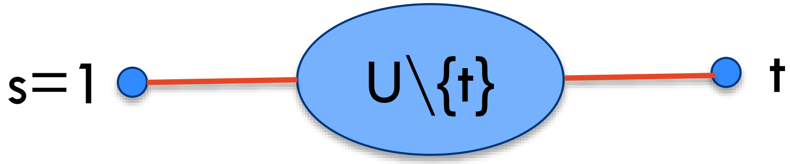


Solving TSP by Dynamic Programming

- Regard a tour to be a simple path that starts and end at vertex 1.

- Every tour consists of an edge $(1,k)$ for some $k \in V - \{1\}$ and a path from k to vertex 1. The path from vertex k to vertex 1 goes through each vertex in $V - \{1,k\}$ exactly once.

- Let $OPT[U,t]$ be the length of a shortest path starting at vertex 1, going through all vertices in U and terminating at vertex t.



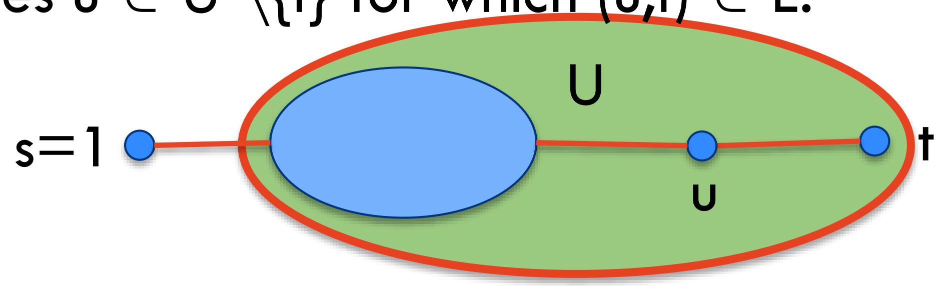
Solving TSP by Dynamic Programming

– $OPT[U,t]$ = length of a shortest path starting at vertex *1*, going through all vertices in *U* and terminating at vertex t.

– OPT[V,1] is the length of an optimal salesperson tour.

– $|U|=1 \;\Rightarrow\; OPT[U=\{t\},t] = d(s,t)$

– $|U|>1$: consider all vertices $u \in U \setminus\{t\}$ for which $(u,t) \in E$.



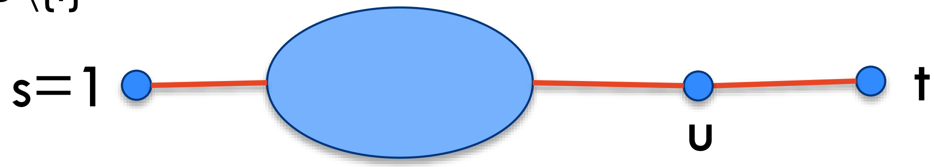
Observation: If a path containing (u,t) is optimal then the subpath on $U\setminus\{t\}$ ending in u must also be optimal.

Solving TSP by Dynamic Programming

– OPT[U,t] = length of a shortest path starting at vertex 1, going through all vertices in U and terminating at vertex t.

– OPT[V,1] is the length of an optimal salesperson tour.

– $|U|=1 \Rightarrow \text{OPT}[U=\{t\},t] = d(s,t)$
– $|U|>1 \Rightarrow \text{OPT}[U,t] = \min_{u \in U \setminus \{t\}} \text{OPT}[U \setminus \{t\},u] + d(u,t)$



Compute all solutions to subproblems in order of increasing cardinality of U.

Solving TSP by Dynamic Programming

$$\mathrm{OPT}[U,t] = \min_{u \in U \setminus \{t\}} \mathrm{OPT}[U \setminus \{t\},u] + d(u,t)$$

Compute all solutions to subproblems in order of increasing cardinality of U.

The cost of each subproblem can be evaluated in O(n) time.

Solving TSP by Dynamic Programming

$$OPT[U,t] = \min_{u \in U \setminus \{t\}} OPT[U \setminus \{t\},u] + d(u,t)$$

Compute all solutions to subproblems in order of increasing cardinality of U.

The cost of each subproblem can be evaluated in O(n) time.

Number of subproblems?

#sets $U = 2^n$

#vertices connected to $t < n$

Solving TSP by Dynamic Programming

$$OPT[U,t] = \min_{u \in U\setminus\{t\}} OPT[U\setminus\{t\},u] + d(u,t)$$

Compute all solutions to subproblems in order of increasing cardinality of U.

The cost of each subproblem can be evaluated in O(n) time.

Number of subproblems?

#sets U = 2^n

#vertices connected to t < n

Total time: $O(n^2\ 2^n)$

Longest path

Input: graph $G=(V,E)$ and an integer $k>0$.

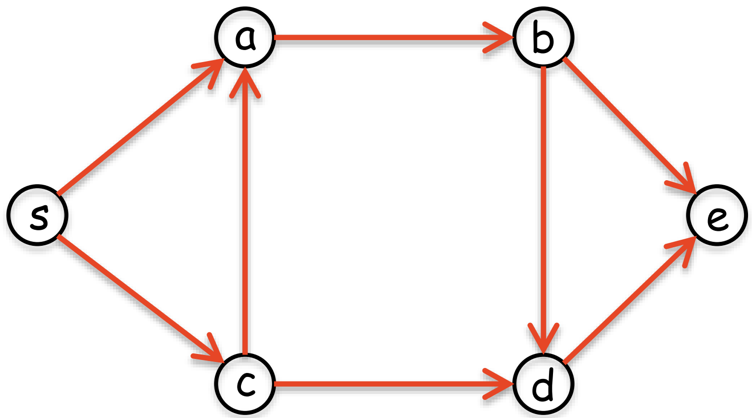
Task: Find a simple path in G on k vertices [unweighted edges].

k-path

Input: graph G=(V,E) and an integer k>0.

Task: Find a simple path in G on k vertices [unweighted edges].

Start with the special case when G is a directed acyclic graph.

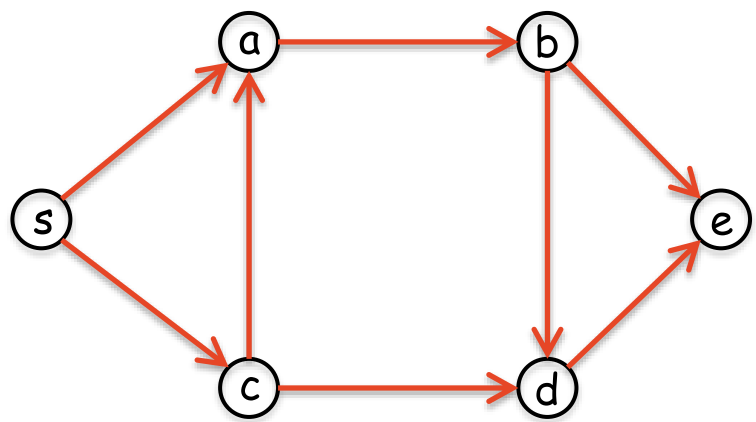


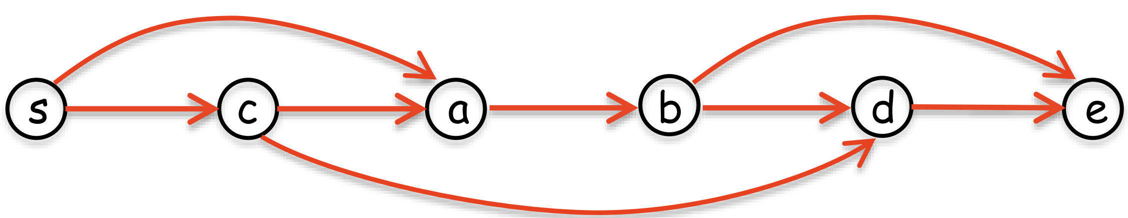
k-path

Input: graph G=(V,E) and an integer k>0.

Task: Find a simple path in G on k vertices [unweighted edges].

Start with the special case when G is a directed acyclic graph.



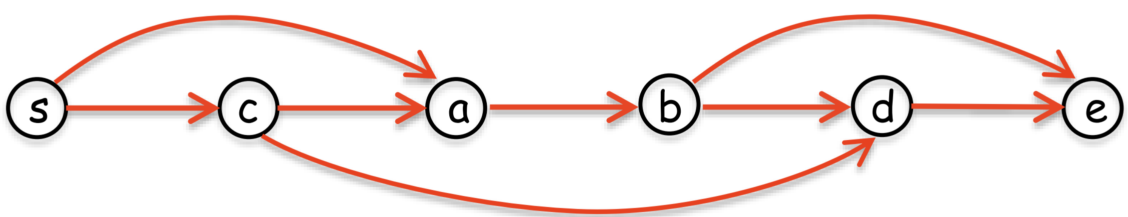


Topologically sorted

k-path

dist(v) = the longest distance from s to vertex v.

Example: $\text{dist}(d) = \max\{\text{dist}(b) + 1, \text{dist}(c) + 1\}$

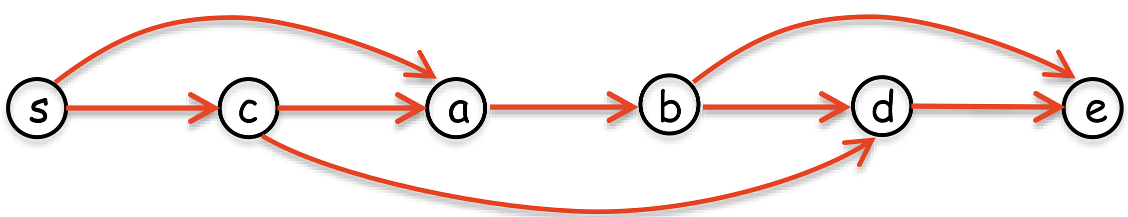


k-path

dist(v) = the longest distance from s to vertex v.

$$\text{dist}(s) = 0$$

$$\text{dist}(v) = \max_{(u,v)\in E} \{\text{dist}(u) + 1\}$$



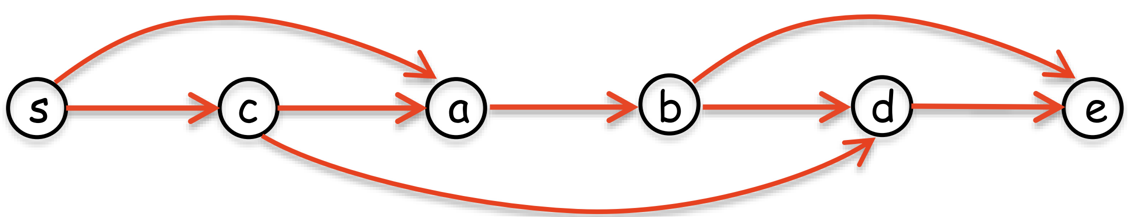
Running time: O(|V|+|E|)

k-path

dist(v) = the longest distance from s to vertex v.

$$\text{dist}(s) = 0$$

$$\text{dist}(v) = \max_{(u,v)\in E} \{\text{dist}(u) + 1\}$$

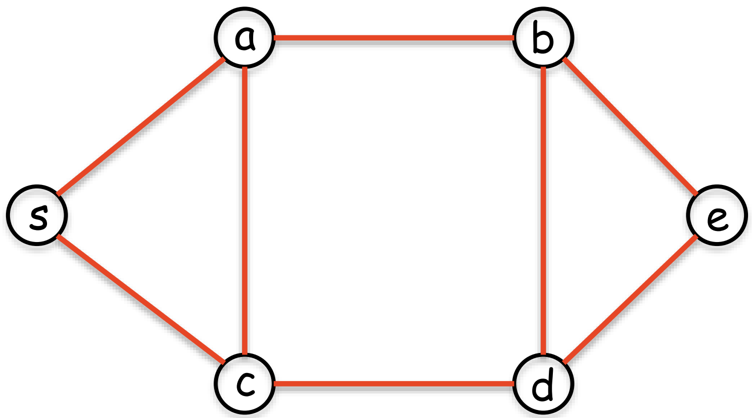


Theorem: The longest path in an acyclic directed graph (DAG) can be computed in $O(|V|+|E|)$ time.

k-path

Input: graph G=(V,E) and an integer k>0.

Task: Find a simple path in G on k vertices.

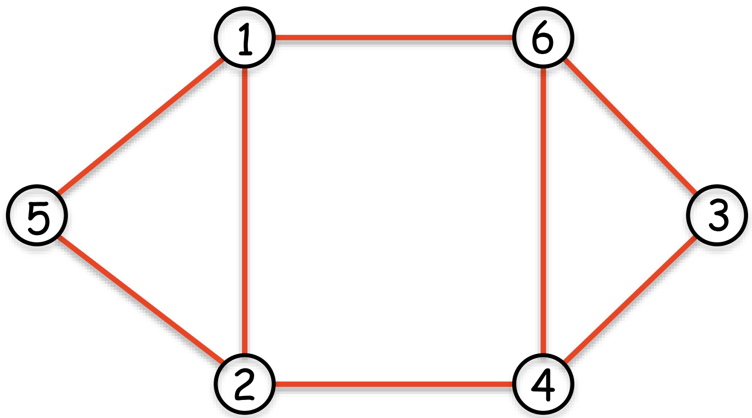


k-path

Input: graph $G=(V,E)$ and an integer $k>0$.

Task: Find a simple path in G on k vertices.

Let $\pi: V \rightarrow [1..n]$ be a random permutation of V.



k-path

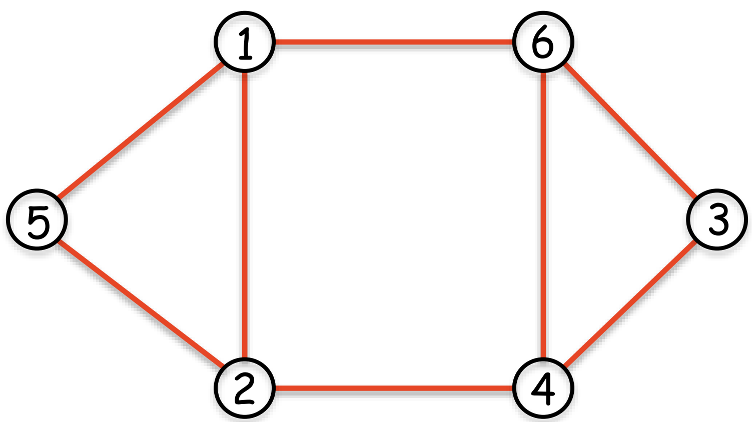
Input: graph G=(V,E) and an integer k>0.

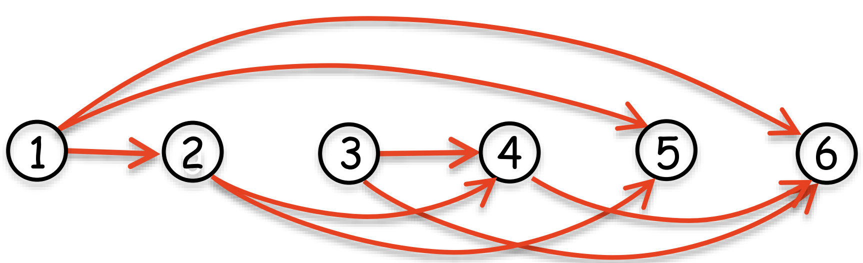
Task: Find a simple path in G on k vertices.

Let π: V $\rightarrow$ [1..n] be a random permutation of V.

Create a DAG G'=(V,E') with vertex set V and edge set E' where

(u,v)$\in$E' if and only if (u,v)$\in$E and π(u)< π(v).

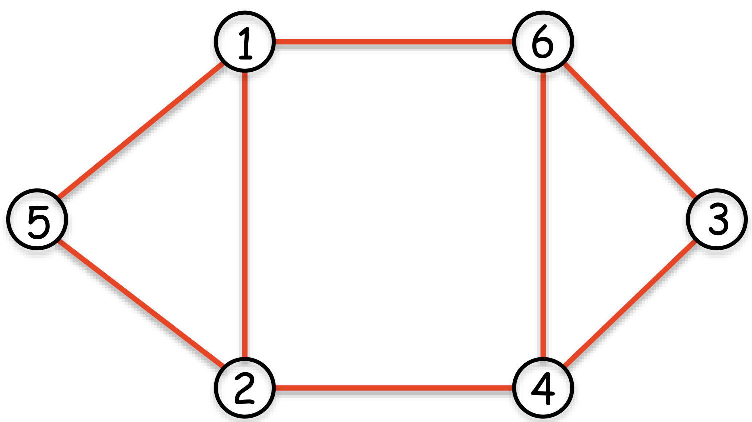


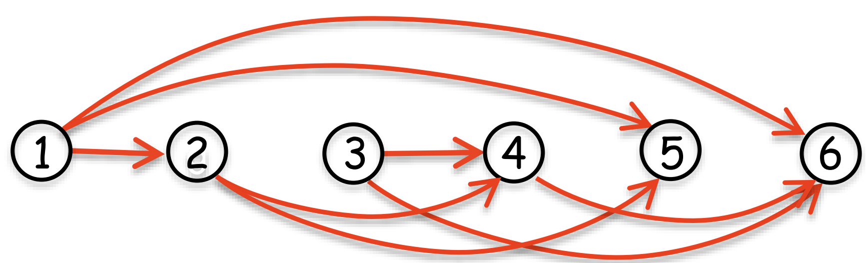


k-path

Create a DAG $G'=(V,E')$ with vertex set V and edge set E' where $(u,v)\in E'$ if and only if $(u,v)\in E$ and $\pi(u) < \pi(v)$.

Observation 1: If there exists a k-path P in G and π happens to order the vertices of P in an orderly manner then P will be detected as a k-path in G'.





k-path

Create a DAG $G'=(V,E')$ with vertex set V and edge set E' where

$(u,v) \in E'$ if and only if $(u,v) \in E$ and $\pi(u) < \pi(v)$.

Observation 1: If there exists a k-path P in G and π happens to order the vertices of P in an orderly manner then P will be detected as a k-path in G'.

Observation 2: If G does not have a k-path then G' (for any permutation) does not have one either.

k-path

Create a DAG $G'=(V,E')$ with vertex set V and edge set E' where

$(u,v)\in E'$ if and only if $(u,v)\in E$ and $\pi(u) < \pi(v)$.

Observation 1: If there exists a k-path P in G and π happens to order the vertices of P in an orderly manner then P will be detected as a k-path in G'.

Observation 2: If G does not have a k-path then G' (for any permutation) does not have one either.

Observation 3: The probability that π turns a k-path in G into a directed k-path in G' is $2/k!$ Why?

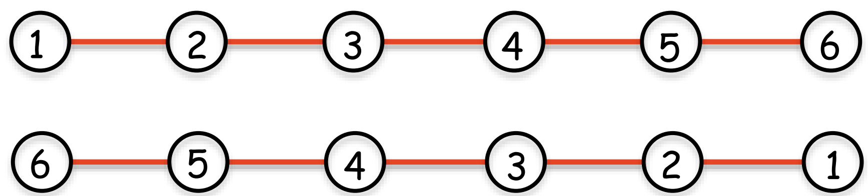
k-path

Observation 3: The probability that π turns a k-path in G into a directed k-path in G' is $2/k!$ Why?

Consider a k-path in G.

When will this path be represented as a k-path in G'?

Two possible cases:



How many permutations are there in total of the path? k!

k-path

Observation 3: The probability that π turns a k-path in G into a directed k-path in G' is 2/k! Why?

Algorithm:

Generate k! random permutations.

For each random permutation construct a DAG G'.

Test if G' has a k-path.

k-path

– If G does not have a k-path then the algorithm always fails.

– If G has a k-path then the probability that the algorithm fails after k! attempts is

$$\underbrace{(1-2/k!)\cdot(1-2/k!)\cdots(1-2/k!)}_{k!} = (1-2/k!)^{k!} < 1/e < ½.$$

– By increasing the number of iterations to c·k! one can improve the probability to $1-2^{-c}$.

Theorem: The k-path problem can be solved with probability $(1-1/2^c)$ in $O(c(|V|+|E|)k!)$ time.

Improved k-path

Technique: Colour coding [Alon et al. '94]

Algorithm

1. Colour the vertices of G with k colours uniformly at random. $C: V \to [1..k]$
2. Find a colourful k-path in G, if one exists. Otherwise, report that none has been found.

Improved k-path

Technique: Colour coding [Alon et al. '94]

Algorithm

1. Colour the vertices of G with k colours uniformly at random. $C{:}V \rightarrow [1..k]$
2. Find a colourful k-path in G, if one exists. Otherwise, report that none has been found.

Definition: A path is colourful if all vertices on the path have distinct colours.

Improved k-path

Technique: Colour coding [Alon et al. '94]

Algorithm

1. Colour the vertices of G with k colours uniformly at random. $C: V \rightarrow [1..k]$
2. Find a colourful k-path in G, if one exists. Otherwise, report that none has been found.

Step 1: What is the probability that a k-path P becomes colourful?

$$\frac{\#\text{colouring for which P is colourful}}{\#\text{possible colourings of P}} = \frac{k!}{k^k} \approx \frac{1}{e^k}$$

Improved k-path

Technique: Colour coding [Alon et al. '94]

Algorithm

1. Colour the vertices of G with k colours uniformly at random. $C: V \to [1..k]$
2. Find a colourful k-path in G, if one exists. Otherwise, report that none has been found.

Step 1: What is the probability that a k-path P becomes colourful?

$$\frac{\#\text{colouring for which P is colourful}}{\#\text{possible colourings of P}} = \frac{k!}{k^k} \approx \frac{1}{e^k}$$

$\Rightarrow$ The expected number of colourings required for a k-path to become colourful is e^k.

Improved k-path: colourful path

Step 2: Find a colourful k-path in G, if one exists. Otherwise, report that none has been found.

P[C,v] – indicator variable for every colour subset $C \subseteq \{1,\ldots,k\}$ and every vertex $v \in V$. Is there a colourful path consisting of the colours in C ending at v.

How many sets C can there be? 2^k

P[C,v] = 1 if there is a colourful path consisting of the colours in C and ending at v.

P[C,v] = 0 otherwise

P[C,v] =1 for C={red,blue,green,yellow}

Improved k-path: colourful path

Step 2: Find a colourful k-path in G, if one exists. Otherwise, report that none has been found.

$P[C,v] = 1$ if there is a colourful path consisting of the colours in C and ending at v.

$P[C,v] = 0$ otherwise

Iterate over i = the number of colours in C.

P[{blue},v]=1

P[{red},v]=0

[i=1]: $P[C,v] = 1$ iff $C=\{c(v)\}$

[i>1]: $P[C,v] = 1$ iff $c(v)\in C$ and $\exists(u,v)\in E$ s.t. $P[C\setminus\{c(v)\},u]=1$.

C={red,blue,green,yellow}

$P[C,v] =1$ since $P[C\setminus\{yellow\},u]=1$ and $(u,v)\in E$

Improved k-path: colourful path

Step 2: Find a colourful k-path in G, if one exists. Otherwise, report that none has been found.

Iterate over i = the number of colours in C.

[i=1]: $P[C,v] = 1$ iff $C=\{c(v)\}$

[i>1]: $P[C,v] = 1$ iff $c(v) \in C$ and $\exists (u,v) \in E$ s.t. $P[C \setminus \{c(v)\},u]=1$.

Theorem: A k-colourful path in a graph G with k colours can be found in $O(2^k \cdot (|V|+|E|))$.

Number of sets C

Improved k-path: summary

Algorithm

i=0

found = false

while not found and i< e^k **do**

1. Colour the vertices of G with k colours uniformly at random. $C: V \rightarrow [1..k]$
2. Find a colourful k-path in G, if one exists. Otherwise, report that none has been found.

end while

Theorem: The k-path problem can be solved with probability $(1-1/e^k)^{e^k} \approx 1/e$ in $O(2^k \cdot e^k \cdot (|V|+|E|))$ time.

Reading material

– TSP:

“Seminar on exact exponential algorithms – Dynamic programming” by Juho-Kustaa Kangas.

https://www.cs.helsinki.fi/u/jwkangas/seminars/report-eea.pdf

– Longest path:

“Exact Algorithms - Lecture 5: Randomized Methods and Color Coding” by Eunjung Kim.

http://www.lamsade.dauphine.fr/~mlampis/Resolution/lecture05.pdf