Turning Points in the Information Sciences

Lecture 5: *What can be computed efficiently?*

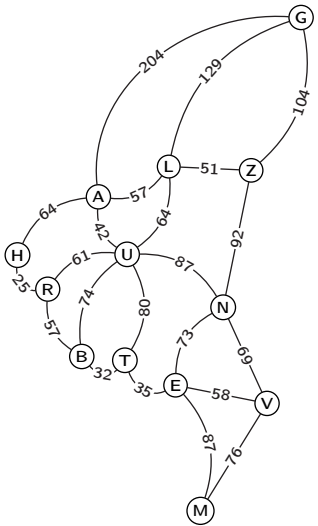
Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Plan for Today

We are still busy understanding the theoretical foundations of computing:

- What can be computed?
  - Turing Machines: general-purpose definition of computation
  - Halting Problem: some things cannot be computed at all
- What can be computed efficiently?
  - Algorithms and Computational Complexity
  - Applications to Cryptography
- What can be computed on a quantum computer?

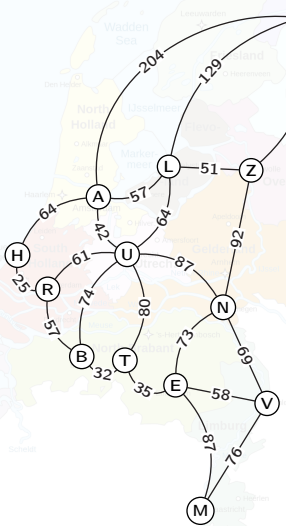
Graph Theory



A weighted graph consists of nodes, with edges between some of them. Each of the edges has a weight.

Exercise: *Have you seen this graph before?*

Graph Theory



A weighted graph consists of nodes, with edges between some of them. Each of the edges has a weight.

Exercise: *Have you seen this graph before?*

Graph Theory

A weighted graph consists of nodes, with edges between some of them. Each of the edges has a weight.

Exercise: *Have you seen this graph before?*

Graph Theory

A weighted graph consists of nodes, with edges between some of them. Each of the edges has a weight.

Possible interpretation:

- nodes = cities
- edges = roads
- weights = distances

The Travelling Salesperson Problem (TSP)

"Is there a tour of length at most K that visits each city exactly once?"

Exercise: *Can we solve this problem for any graph?*

Solution: Brute Force

Yes, we can solve any instance of the TSP, of any problem size, in finite time. *Nice!*

One possible approach is the so-called Brute-Force Algorithm:

(1) Write down every possible way or ordering the cities
(2) Calculate for each such ordering the length of the corresponding tour
(3) Check whether at least one tour is sufficiently short

Exercise: *How many tours do we need to check for maps with 10 cities?*

`wooclap.com` → `UVAINF`

Combinatorial Explosion

For 10 cities, fixing the origin, we need to check this many tours:

$$9! \;=\; 9 \times \cdots \times 3 \times 2 \times 1 \;=\; 362{,}880$$

For 20 cities, this number goes up to $19! = 121{,}645{,}100{,}408{,}832{,}000$. *Too much!*
Even if you can check 100,000 tours per millisecond this might take around 38 years.

(the expression $n!$ is pronounced "n *factorial*")

Efficient Algorithm?

Runtime depends on the size of the problem (such as the number of cities on the map). We want algorithms that run in polynomial time $f(n)$ relative to the problem size n.

polynomial
good!

$f(n) = n^2$	n	$f(n) = 2^n$
100	10	1024
400	20	1048576
900	30	1073741824
1600	40	1099511627776
2500	50	1125899906842624
3600	60	1152921504606846976

exponential
bad!

Fact: For close to 100 years now, countless mathematicians and computer scientists have tried to design a poly-time algorithm for TSP. *Nobody managed.*

The Shortest Path Problem (SPP)

"Is there a path from A to Z of length at most K?"

SPP looks very similar to TSP

Brute Force also works for SPP

But can we do better?

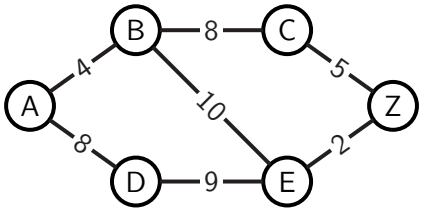
Dijkstra's Algorithm

Suppose we are looking for a path from A to Z. Here's a poly-time algorithm for this.

Visit nodes in order of distance from the source A ("expanding spheres"):

- Maintain a list of visited nodes (initially just A). Other nodes are unvisited.
- For each visited node, keep track of: shortest path from A + its length.
- In each round, visit one new node, picking the one with the shortest path.
- Stop once you visit Z.

A	0	
B	∞	?
C	∞	?
D	∞	?
E	∞	?
Z	∞	?



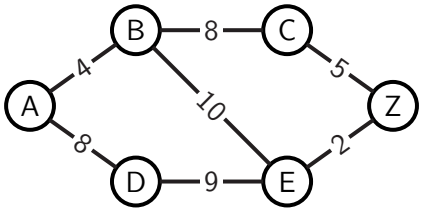
Dijkstra's Algorithm

Suppose we are looking for a path from A to Z. Here's a poly-time algorithm for this.

Visit nodes in order of distance from the source A ("expanding spheres"):

- Maintain a list of visited nodes (initially just A). Other nodes are unvisited.
- For each visited node, keep track of: shortest path from A + its length.
- In each round, visit one new node, picking the one with the shortest path.
- Stop once you visit Z.

A	0	
B	4	A–B
C	∞	?
D	∞	?
E	∞	?
Z	∞	?



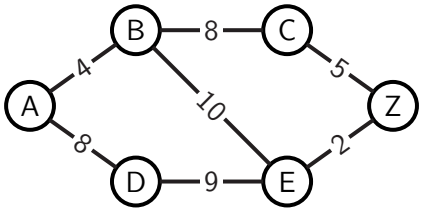
Dijkstra's Algorithm

Suppose we are looking for a path from A to Z. Here's a poly-time algorithm for this.

Visit nodes in order of distance from the source A ("expanding spheres"):

- Maintain a list of visited nodes (initially just A). Other nodes are unvisited.
- For each visited node, keep track of: shortest path from A + its length.
- In each round, visit one new node, picking the one with the shortest path.
- Stop once you visit Z.

A	0	
B	4	A–B
C	∞	?
D	8	A–D
E	∞	?
Z	∞	?



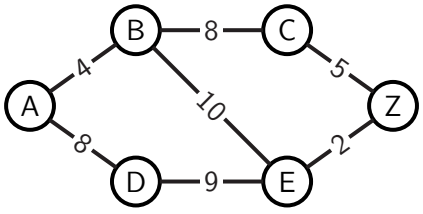
Dijkstra's Algorithm

Suppose we are looking for a path from A to Z. Here's a poly-time algorithm for this.

Visit nodes in order of distance from the source A ("expanding spheres"):

- Maintain a list of visited nodes (initially just A). Other nodes are unvisited.
- For each visited node, keep track of: shortest path from A + its length.
- In each round, visit one new node, picking the one with the shortest path.
- Stop once you visit Z.

A	0	
B	4	A–B
C	12	A–B–C
D	8	A–D
E	∞	?
Z	∞	?



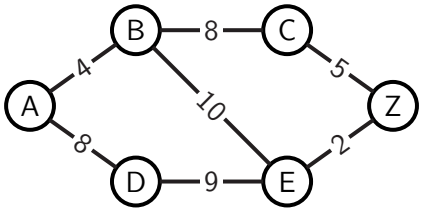
Dijkstra's Algorithm

Suppose we are looking for a path from A to Z. Here's a poly-time algorithm for this.

Visit nodes in order of distance from the source A ("expanding spheres"):

- Maintain a list of visited nodes (initially just A). Other nodes are unvisited.
- For each visited node, keep track of: shortest path from A + its length.
- In each round, visit one new node, picking the one with the shortest path.
- Stop once you visit Z.

A	0	
B	4	A–B
C	12	A–B–C
D	8	A–D
E	14	A–B–E
Z	∞	?



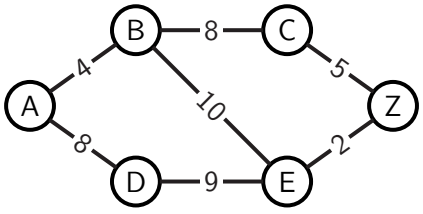
Dijkstra's Algorithm

Suppose we are looking for a path from A to Z. Here's a poly-time algorithm for this.

Visit nodes in order of distance from the source A ("expanding spheres"):

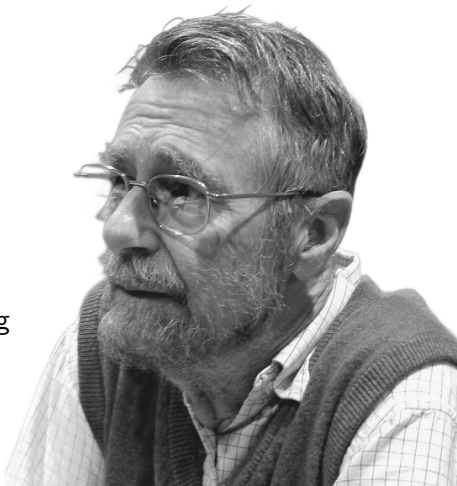
- Maintain a list of visited nodes (initially just A). Other nodes are unvisited.
- For each visited node, keep track of: shortest path from A + its length.
- In each round, visit one new node, picking the one with the shortest path.
- Stop once you visit Z.

A	0	
B	4	A–B
C	12	A–B–C
D	8	A–D
E	14	A–B–E
Z	16	A–B–E–Z



Edsger W. Dijkstra

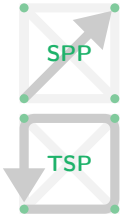
- Born 1930 in Rotterdam (died 2002)
- Studied Mathematics and Physics at Leiden
- First job at *Mathematisch Centrum* (now CWI)
- Shortest Path paper published in 1956
- Work on first ALGOL 60 compiler around 1960
- Go-To Considered Harmful paper published in 1968
- Turing Award in 1972 for structured programming
- Advocate of simplicity and elegance in programming



WOOCLAP.COM
CODE: **UVAINF**

SHORTEST **BIKE TOUR** VISITING ALL 57,912 NATIONAL MONUMENTS IN THE NETHERLANDS

Turning Point: **P** *vs* **NP**



So SPP *seems* easier than TSP.

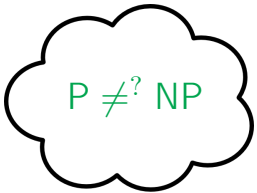
We say: SPP is in **P** (meaning: it can be solved in polynomial time).

We don't know whether TSP is in **P** as well (most likely it is not).

Observe that just verifying correctness of a claimed solution is easier than finding one.

We say: TSP is in **NP** (meaning: claimed solutions can be verified in polynomial time).

The **N** is for "nondeterministic" (as we have to guess a solution before we can verify it).



NP-Completeness

Problem A is said to be no harder than problem B, if we can turn any algorithm for solving B into an algorithm for solving A, by doing at most poly-time extra work.

The hardest problems in **NP** are said to be **NP**-complete.

TSP is **NP**-complete. Hundreds more problems are known to be **NP**-complete as well.

Should you manage to find an efficient algorithm for one **NP**-complete problem, you'll automatically get one for all other problems in **NP** as well. *Fame and fortune await.*

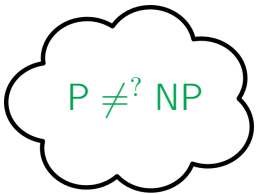
"I can't find an efficient algorithm, but neither can any of these famous people."

Recap

No known efficient solution for TSP (Travelling Salesperson Problem)

But Dijkstra's algorithm efficiently solves SPP (Shortest Path Problem)

SPP is in **P**, while TSP is **NP**-complete



Turning Point: Realisation of the significance of **P**-*vs*-**NP** (around 1971)

What next?

Christian Schaffner on cryptography
(using computational complexity to protect your secrets)

Picture Credits

Netherlands map from Wikipedia/Alphaton

Picture of 48-city TSP solution taken from Lance Fortnow's "The Golden Ticket"

Photo of Edsger Dijkstra by Hamilton Richards, E.W. Dijkstra Archive

Picture of NL57912 bike tour taken from TSP resource page at University of Waterloo

NP-completeness cartoon by Stefan Szeider (based on classic by Garey and Johnson)

Clipart from `pngwing.com`