Optimisation problems

- **Many decision problems are decision versions of *optimisation problems***
  - **problems that involve maximising or minimising a value over a set of feasible solutions**
- **Formally, an *optimisation problem* has:**
  - **A set of instances I**
  - **A function SOL that associates with any instance the set of feasible solutions**
  - **A measure function m that assigns a non-negative integer $m(x,y)$ to any feasible solution y for a given instance x**
  - **a $GOAL$, that is, either max or min**
- **Given $x \in I$, let $m^*(x) = GOAL\{m(x,y) : y \in SOL(x)\}$ denote the *optimal measure***
- **Given $x \in I$, the objective is to find, an *optimal solution*, i.e. a feasible solution $y^* \in SOL(x)$ such that $m(x,y^*)=m^*(x)$**

Example optimisation problem

Maximum Satisfiability (MAX-SAT):

- **Instance:** Boolean formula B in CNF (as for SAT)
- **Feasible solutions:** All truth assignments for B
- **Measure:** Number of clauses of B that are satisfied
- **Goal:** max

Objective is to find a truth assignment for B which satisfies the largest number of clauses of B simultaneously

Example:

$$(x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3)$$

Setting $f(x_1) = F$ and $f(x_2) = f(x_3) = T$ satisfies two clauses

Setting $g(x_1) = g(x_2) = g(x_3) = T$ satisfies three clauses

Decision version of an optimisation problem

- Given an optimisation problem Π, the *decision version* of Π, denoted by Π_d, is defined as follows:
  - Instance: any instance $x \in I$; integer k (the target)
  - Question: is there a feasible solution $y \in SOL(x)$ such that $m(x,y) \leq k$ (if GOAL=min) or $m(x,y) \geq k$ (if GOAL=max)?
- Usually Π is solvable in polynomial time if and only if Π_d is
- Example decision version – MAX-SAT-D:
  - Instance: Boolean formula in CNF with m clauses and target integer k
  - Question: Is there a truth assignment that satisfies k or more clauses simultaneously?
- Restriction: MAX-SAT-D with $k=m$ is SAT so MAX-SAT-D is NP-complete

The Classes NPO and PO

- **NPO:** Optimisation problems whose decision versions are in NP
  - Examples: MAX-SAT, Minimum Vertex Cover, Maximum Clique, Minimum Graph Colouring, Maximum Cut, . . .
- **PO:** those NPO problems solvable in polynomial time
  - Example: Maximum Matching
- We say that a problem Π in NPO is *NP-hard* if its decision version Π_d is NP-complete
  - Example: MAX-SAT
- **Theorem 1:** P = NP if and only if PO = NPO
- **Theorem 2:** If P $\neq$ NP and Π is NP-hard then $\Pi \notin$ PO

Approximation algorithms

Let Π be a problem in NPO

- An *exact* or *optimising algorithm* finds an optimal solution, given any instance x of Π

If Π is NP-hard, we consider *approximation algorithms* for Π

- An *approximation algorithm* A for Π returns, in polynomial time, A(x), where $A(x) \in SOL(x)$, for any instance x of Π
- A has a *performance guarantee* c ($c \geq 1$) if
  - GOAL=min and $m(x,A(x)) \leq c \times m^*(x)$ for all instances x of Π, *or*
  - GOAL=max and $m(x,A(x)) \geq 1/c \times m^*(x)$ for all instances x of Π
- Refer to A as a *c-approximation algorithm*
  - The closer c is to 1, the better the feasible solution A guarantees to deliver

Examples of approximation algorithms

Example 1

- Approximation algorithm for Minimum Vertex Cover with performance guarantee 2
  - see tutorial exercises

Example 2

- Approximation algorithm for MAX-SAT with performance guarantee 2

- Returns a truth assignment satisfying at least half the number of clauses satisfied by an optimal truth assignment

- In fact, returns a truth assignment that satisfies at least half the number of clauses in the given formula

Approximation algorithm for MAX-SAT

```
/** Input: Boolean formula b in CNF
  * Output: Truth assignment f */
for ( variable x : b )
   f(x) = true;
while (b has at least one clause)
{  let u be a variable in b;
   p = number of clauses in b which contain u;
   q = number of clauses in b which contain ū;
   if ( p ≥ q ) {
      f(u) = true;
      remove clauses containing u from b;
      delete occurrences of ū from b;
   }
   else
   {  f(u) = false;
      remove clauses containing ū from b;
      delete occurrences of u from b;
   }
   delete empty clauses from b;
}
return f;
```

Claim: this algorithm returns a truth assignment which satisfies at least half of the **m** clauses

Proof: By induction on the number of variables **n** in any CNF formula **B**

Base step n=1: **B** is u_1 or $\bar{u}_1$ or $(u_1 \vee \bar{u}_1)$ or $u_1 \wedge \bar{u}_1$ or $u_1 \wedge (u_1 \vee \bar{u}_1)$ or $\bar{u}_1 \wedge (u_1 \vee \bar{u}_1)$ – in all cases, algorithm satisfies at least one out of one or two clauses

Inductive step: assume true for all CNF formulae with **n-1** variables.

Let **u** be the first variable to which a value has been assigned. Assume $p \geq q$ (argument similar for $p < q$).

Let **B′** be CNF formula resulting from deletions of clauses and literals. Assume **B′** contains **r** clauses.

By the induction hypothesis, algorithm returns a truth assignment **f** satisfying $\geq r/2$ clauses of **B′**.

Let **f(u)=true**; then **f** satisfies at least

$$p + r/2 \geq p + (m-p-q)/2 \geq m/2$$

clauses of **B**. □

- **Theorem: MAX-SAT** has a **2**-approximation algorithm
- A more sophisticated algorithm achieves a performance guarantee of 1.2551 (Avidor et al, 2005)

The Class APX

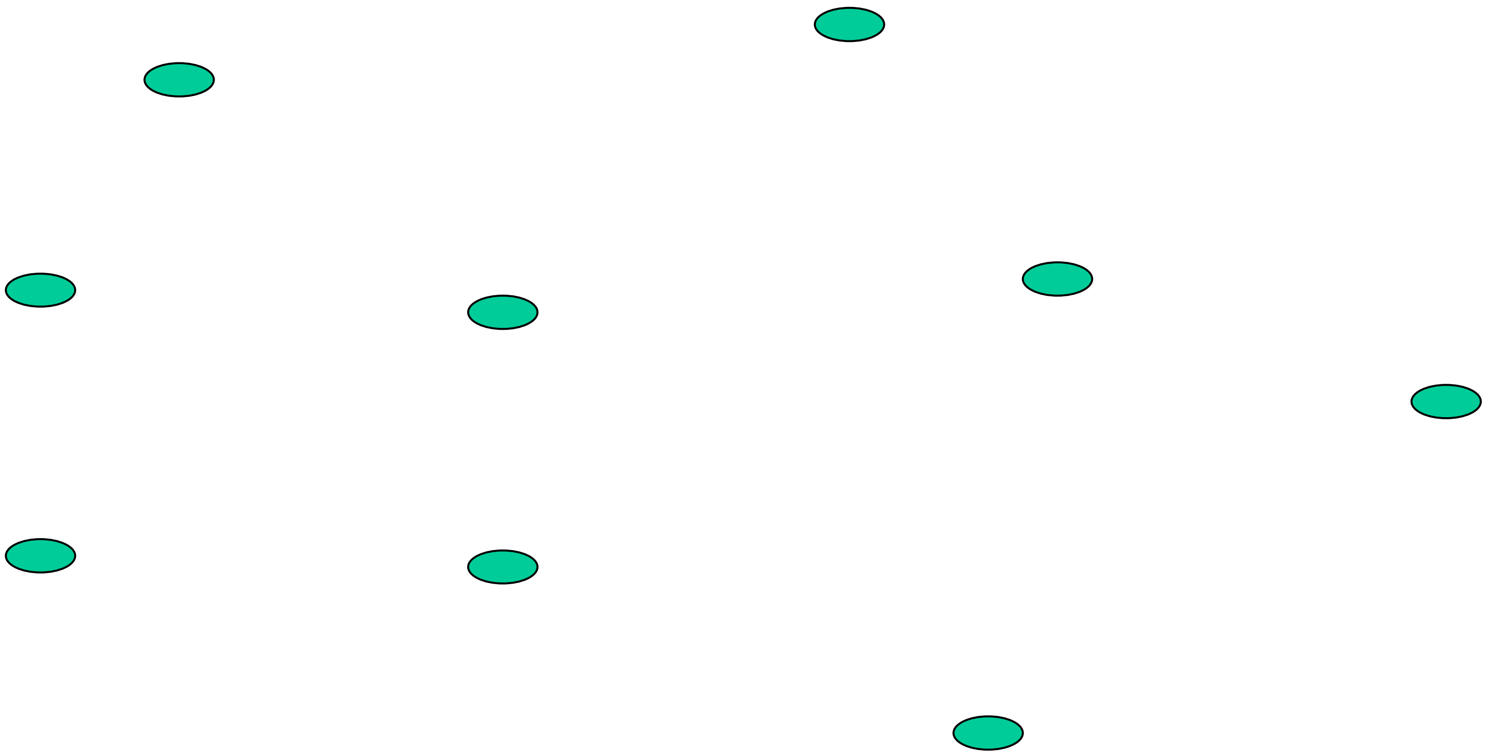
- **NPO** problems Π that admit a **c**-approximation algorithm, for some constant $c \geq 1$
- Problem Π is said to be ***c-approximable*** or ***approximable within (a factor of) c***
- **Examples:**
  - Minimum Vertex Cover (c=2)
  - TSP under triangle inequality (c=3/2) – see below
  - MAX-SAT (c=1.2551)
- We have $PO \subseteq APX \subseteq NPO$

Example 3: the *Travelling Salesman Problem* (TSP)

- Approximation algorithms based on *spanning trees*
- Assume *triangle inequality* is satisfied
  - $d(x, z) \leq d(x, y) + d(y, z) \;\; \forall\, x, y, z$
  - true in most applications
- Claim: For a TSP instance I let OPT(I) be the length of the shortest tour, and T(I) the cost of a minimum weight spanning tree; then T(I) < OPT(I)
  - because, discarding one edge from shortest tour gives a spanning tree
- Algorithm M (*"twice-around the tree"*):

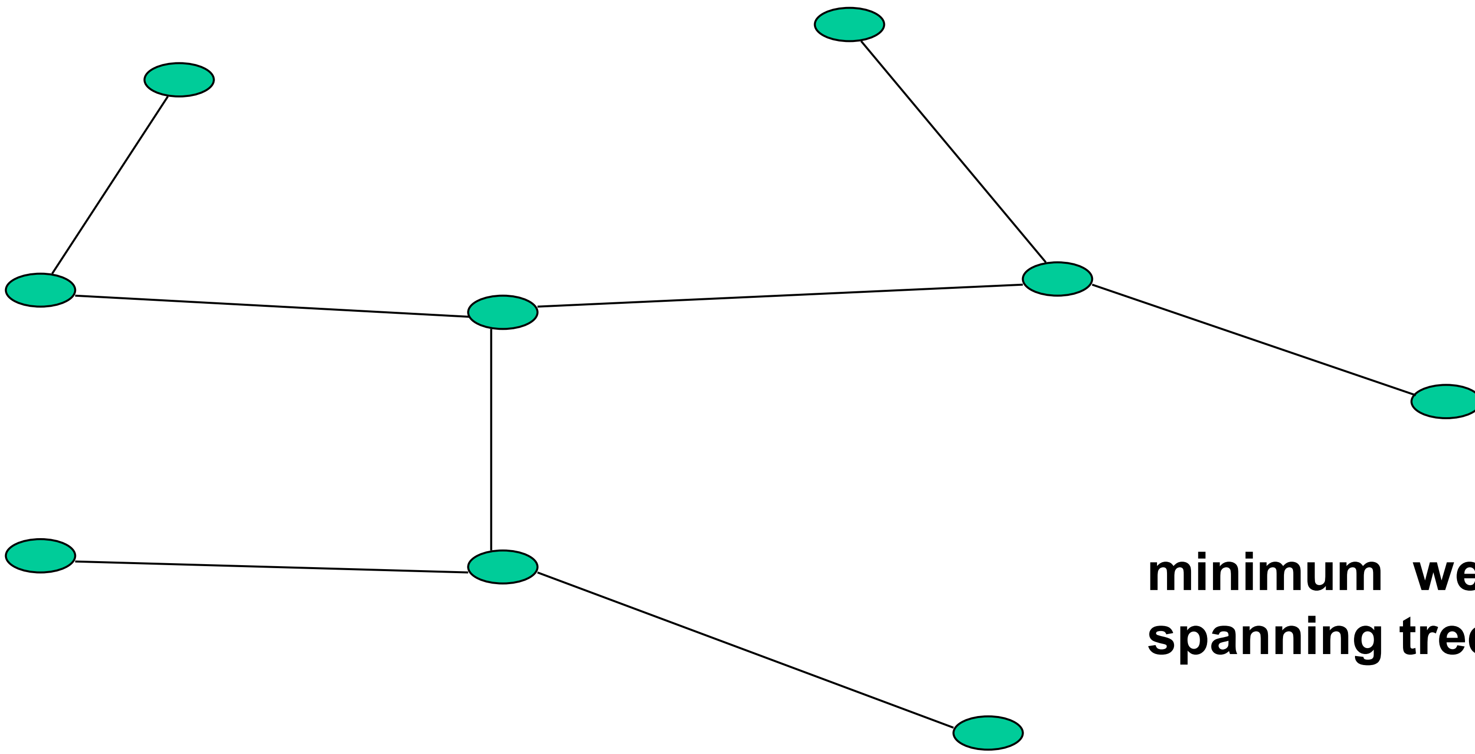
```
find a minimum weight spanning tree T;
carry out a depth-first traversal of T, following each
                              edge once in each direction;
take shortcuts to avoid visiting cities more than once;
return the tour so constructed;
```

Illustration of Algorithm M



Here, edge weight = Euclidean distance between the vertices

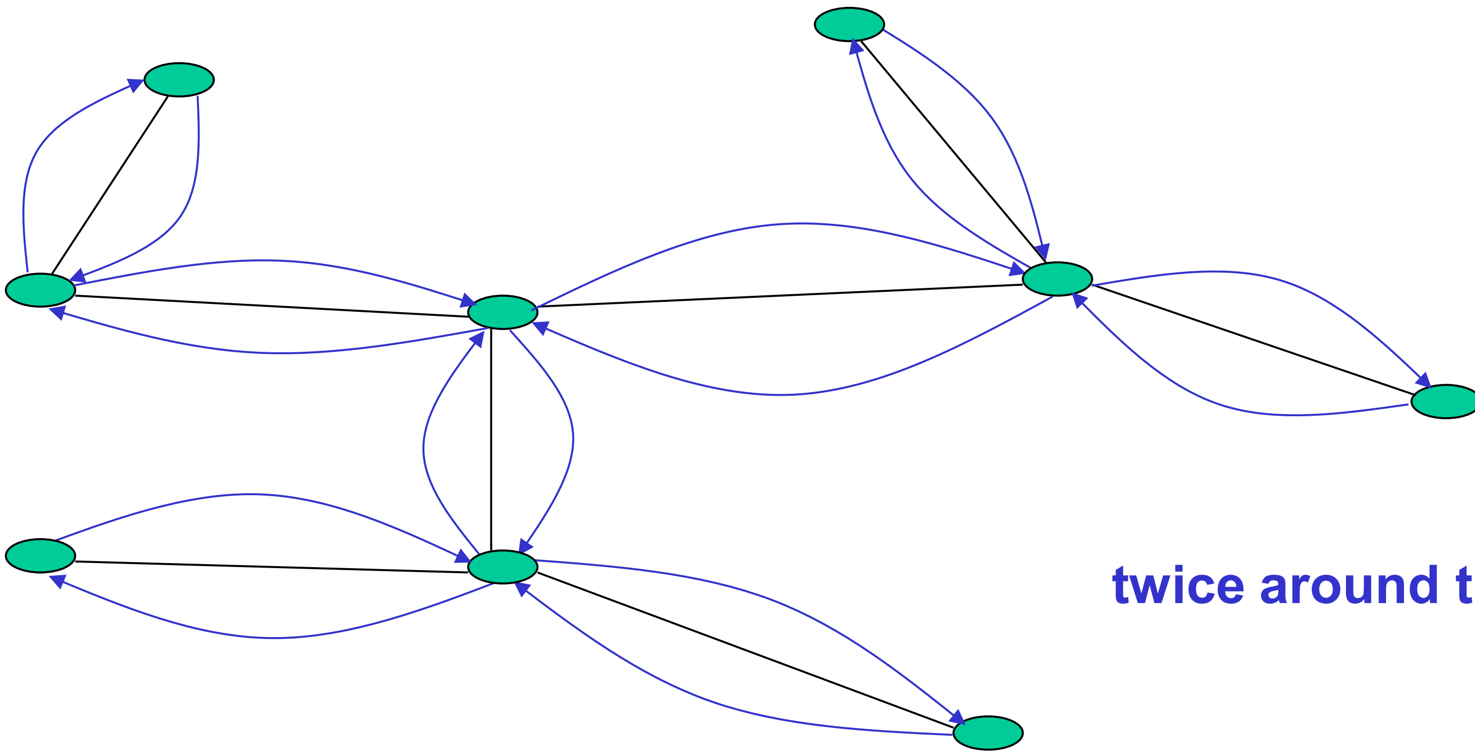
Illustration of Algorithm M



minimum weight
spanning tree

Here, edge weight = Euclidean distance between the vertices

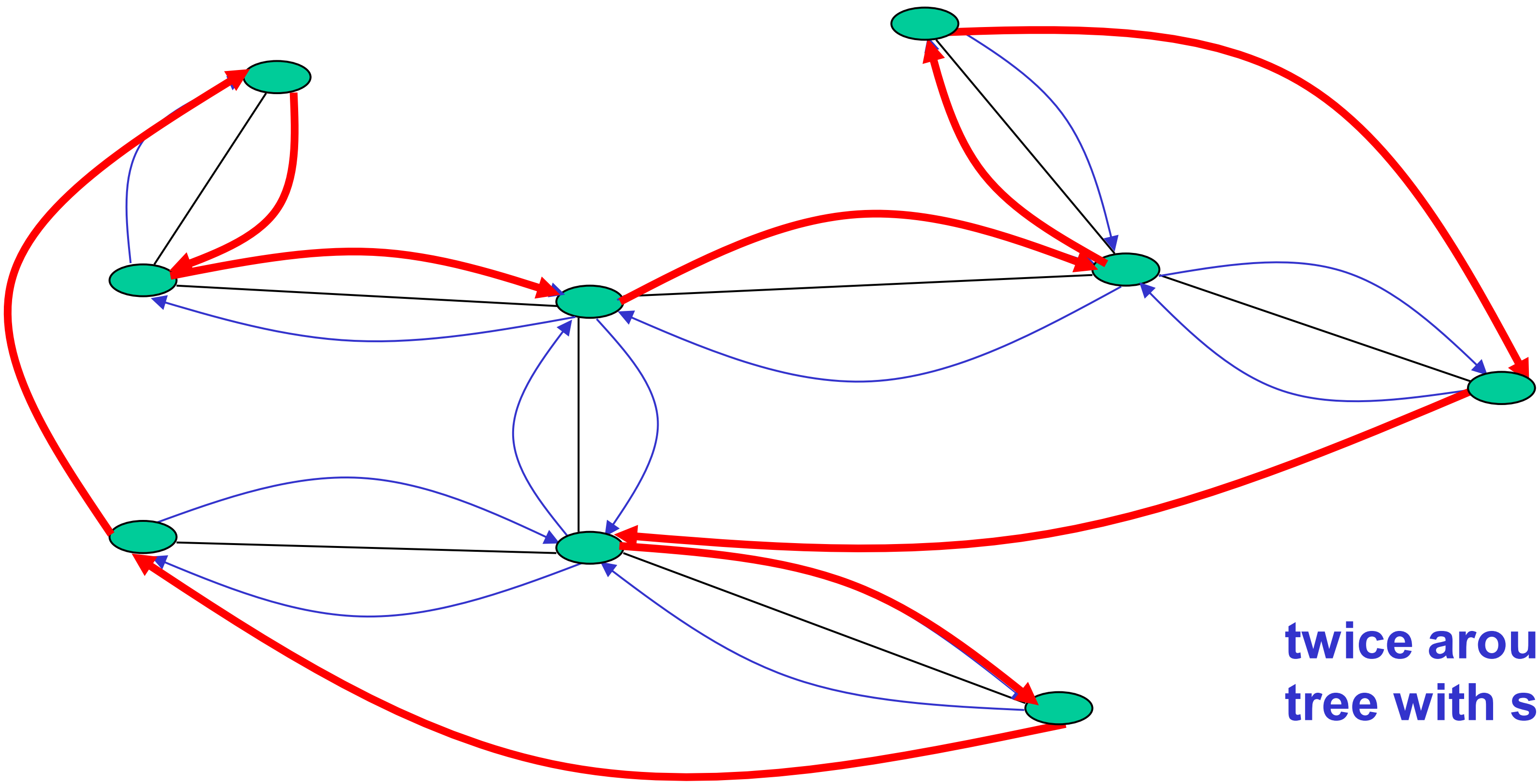
Illustration of Algorithm M



twice around the tree

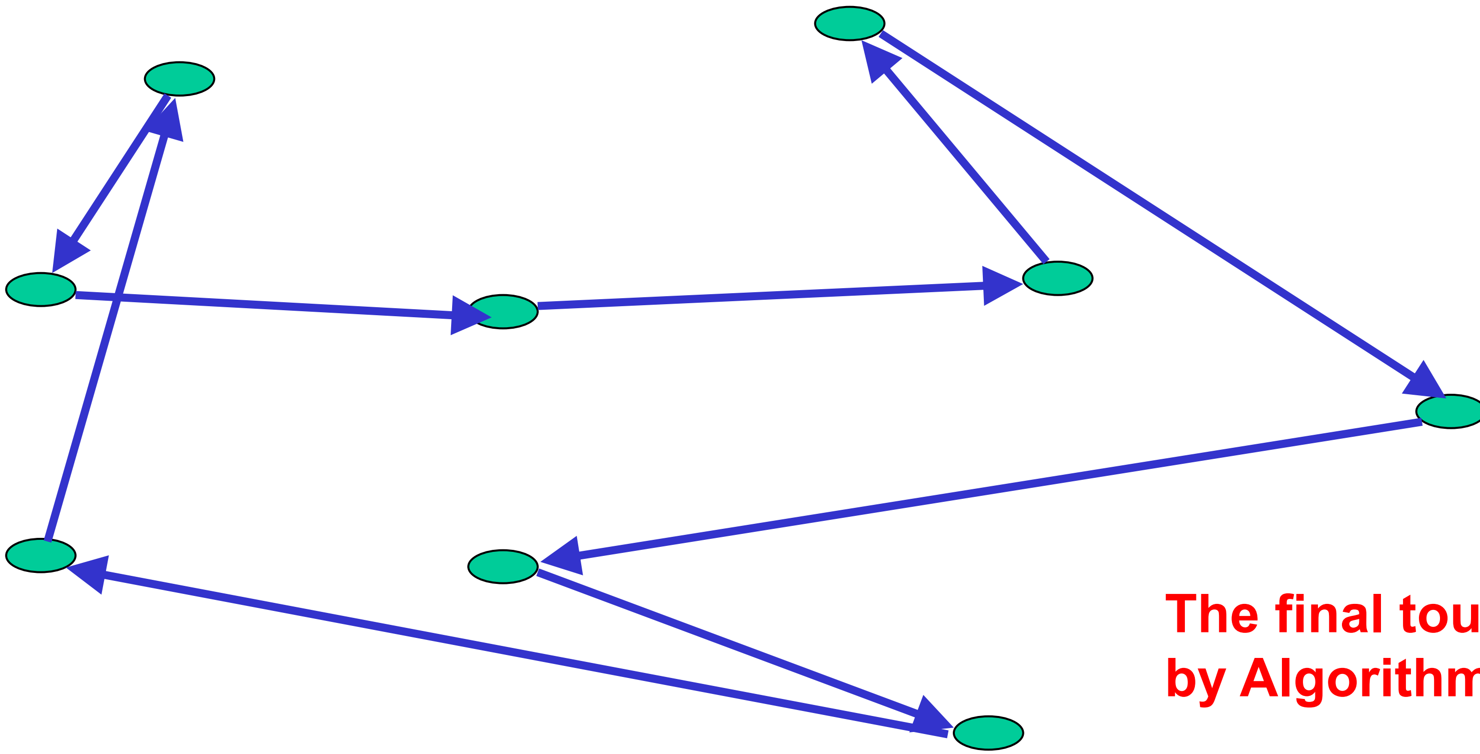
distance travelled = $2 \times T(I)$

Illustration of Algorithm M



twice around the tree with shortcuts

Illustration of Algorithm M

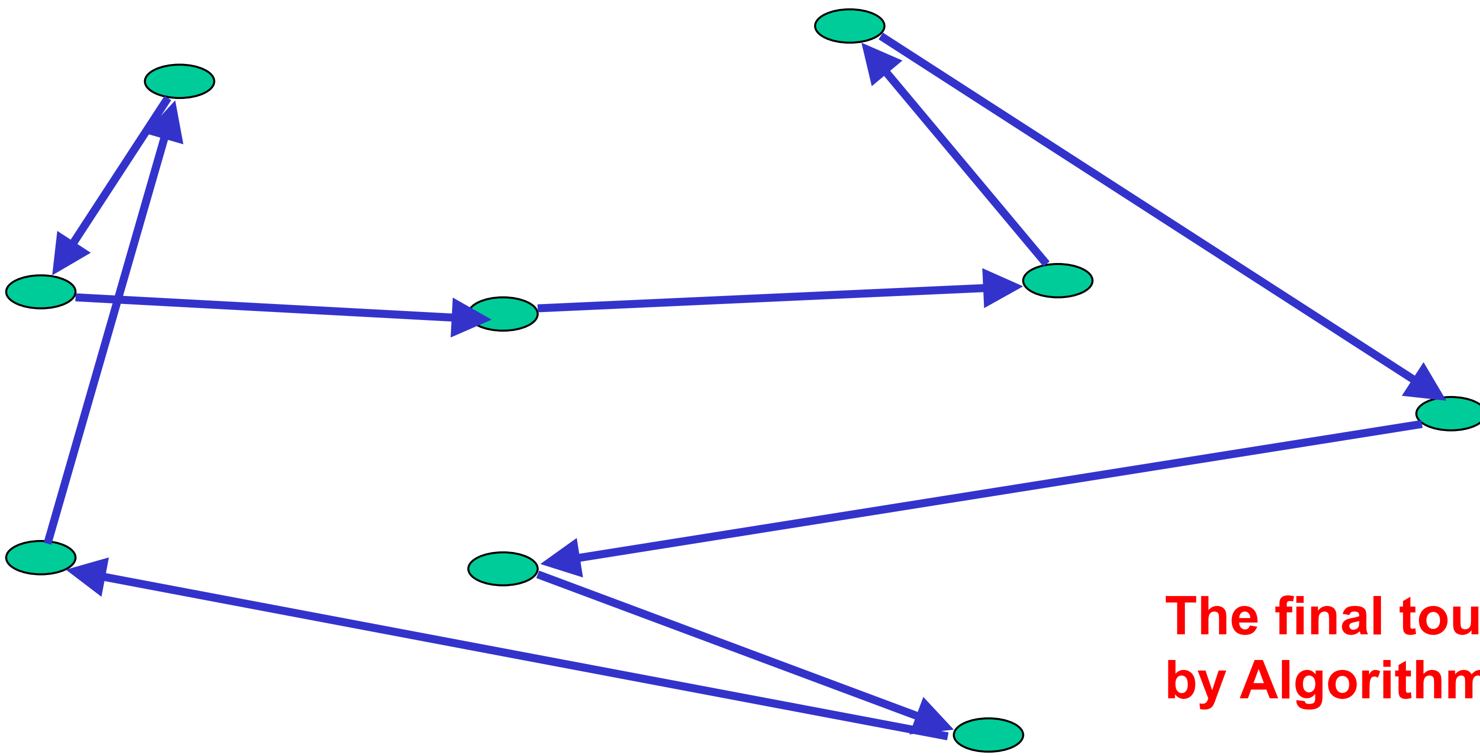


The final tour given by Algorithm M

- distance travelled = M(I) $\leq$ 2 $\times$ T(I) (triangle inequality)
  < 2 $\times$ OPT(I) (from earlier)

- Hence performance guarantee of 2

Illustration of Algorithm M



The final tour given by Algorithm M

- More complex algorithm: Algorithm C (*Christofides' algorithm*)
- Involves computing minimum weight spanning tree, minimum weight matching and Eulerian Cycle
- Gives a performance guarantee of 3/2