

620-362 Applied Operations Research

Heuristics

Department of Mathematics and Statistics
The University of Melbourne

This presentation has been made in accordance with the provisions of Part VB of the copyright act for the teaching purposes of the University.

*For use of students of the University of Melbourne enrolled in the subject 620-362.
Copyright©2008 by Heng-Soon Gan*

Some contents of this presentation are adapted from year 2005 course notes for 620-362 Applied Operations Research, Department of Mathematics and Statistics, The University of Melbourne (compiled by Prof Natasha Boland and Dr Renata Sotirov)

Heuristic Types

1. Building

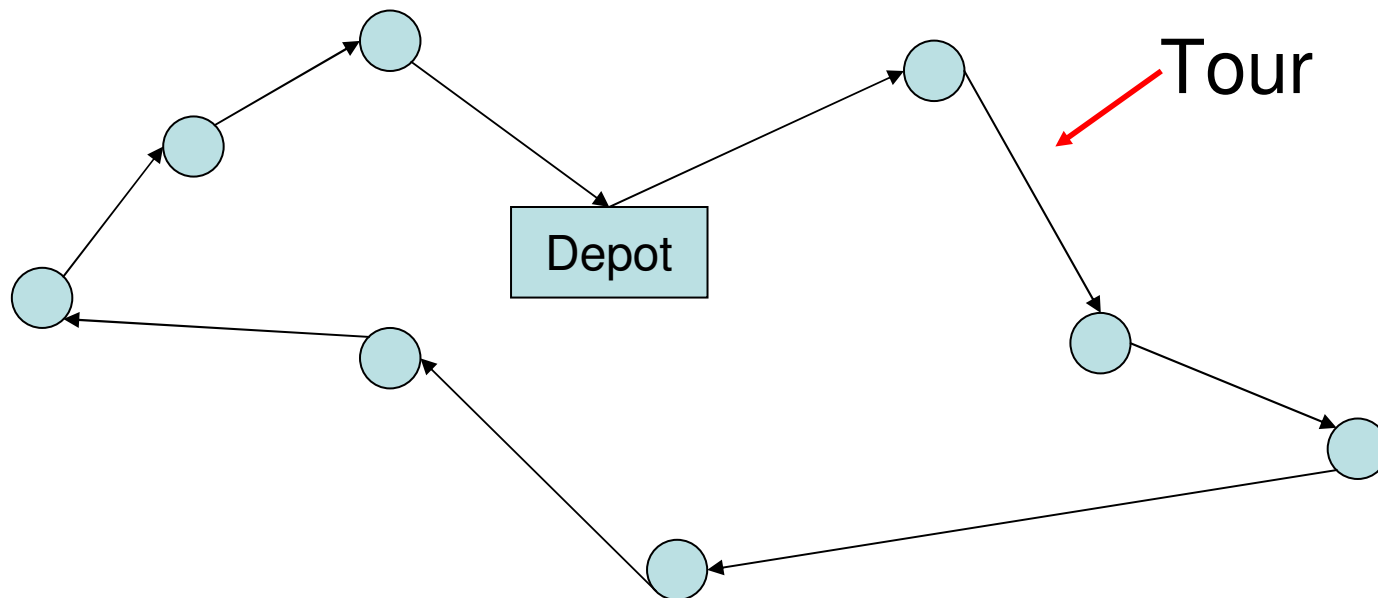
2. Changing

- Exchange heuristics
- Metaheuristics
 - Simulated annealing
 - Tabu search
 - Genetic algorithms

Building Heuristics: TSP

TSP: Travelling Salesperson Problem

What order should salesperson visit customers in (a **tour**) so as to minimize travel costs?

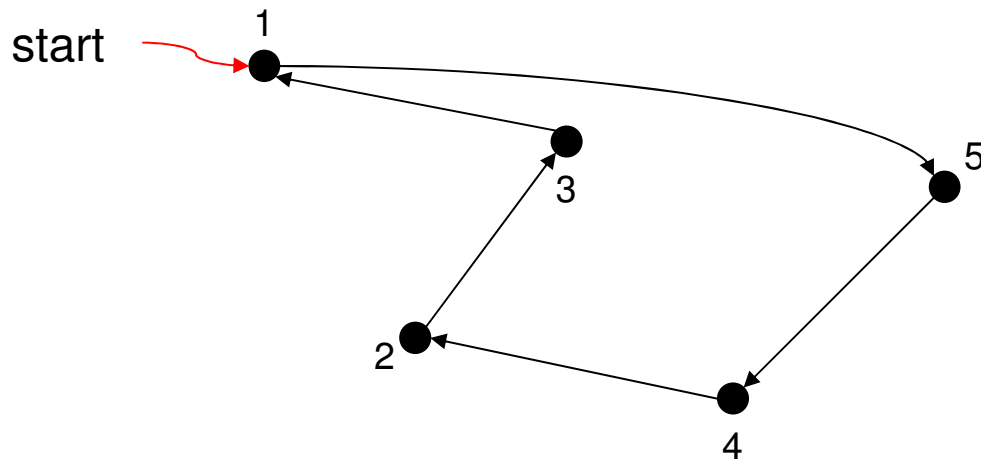


Building Heuristics: TSP

1. Nearest neighbour heuristic

Choose any starting node.

Given a path so far, add on nodes not in path, but at least cost (closest) to path at end of path.



Cost of tour

$$= 2.1 + 2 + 2 + 7.7 + 7.4$$

$$= 21.2$$

	2	3	4	5
1	2.9	7.4	7.1	2.1
2		7.7	2	9.6
3			6.7	3.9
4				2

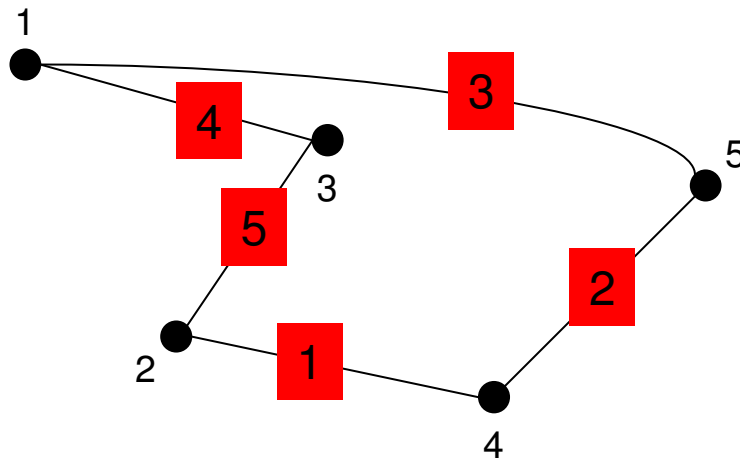
Building Heuristics: TSP

2. Greedy

Choose the cheapest edge and put it in the tour.

Add the cheapest edge not already in the tour, which if added, would not create a **subtour**, to the tour.

Also the edge to be added can't create a degree 3 node!



k k^{th} edge chosen

Cost of tour

$$= 2.1 + 2 + 2 + 7.7 + 7.4$$

$$= 21.2$$

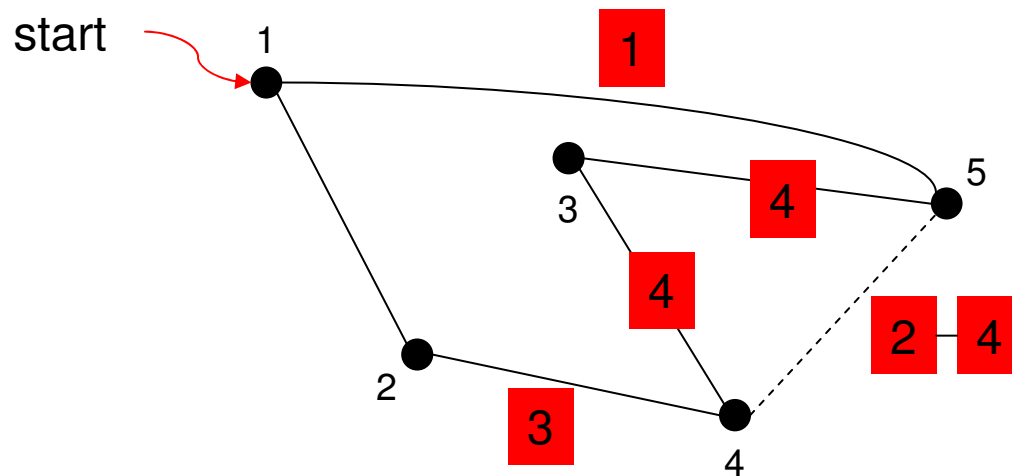
	2	3	4	5
1	2.9	7.4	7.1	2.1
2		7.7	2	9.6
3			6.7	3.9
4				2

Building Heuristics: TSP

3. Nearest Insertion

Given subtour T , look for node i closest to any node in T , say i closest to j in T , where $l-j-k$ is in T .

If i is closer to l than to k , insert i in T between l and j ; otherwise insert i in T between j and k .



k edge created/removed on the k^{th} iteration

Cost of tour

$$= 2.1 + 3.9 + 6.7 + 2 + 2.9$$

$$= 17.6$$

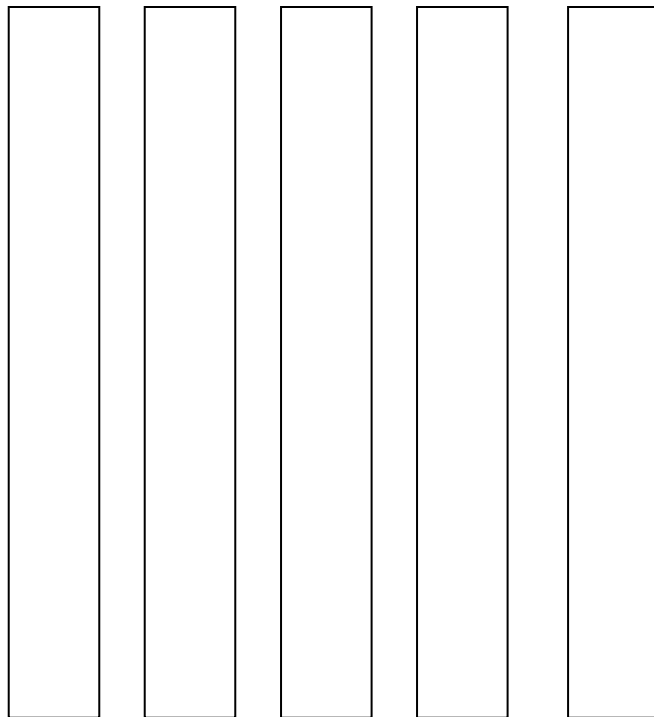
	2	3	4	5
1	2.9	7.4	7.1	2.1
2		7.7	2	9.6
3			6.7	3.9
4				2

Building Heuristics: Cutting Stock

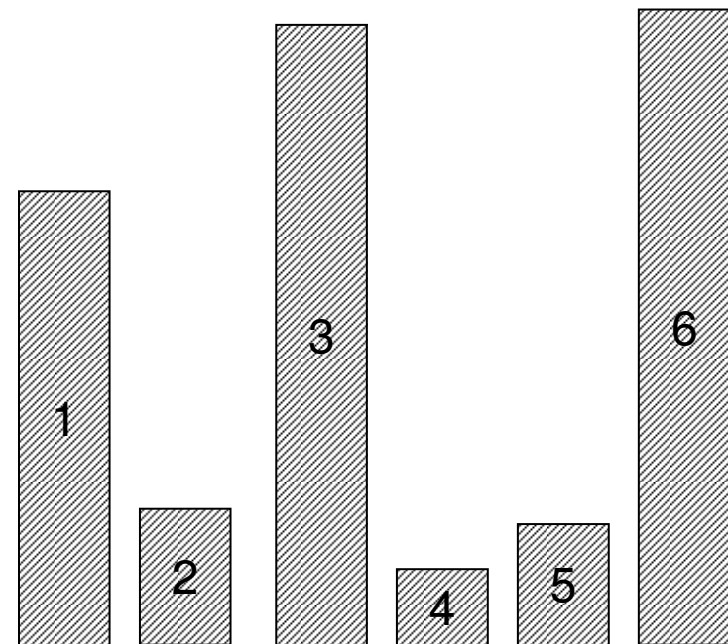
Cutting Stock Problem

Minimise the amount of stock used to meet demand

Stock



Demands

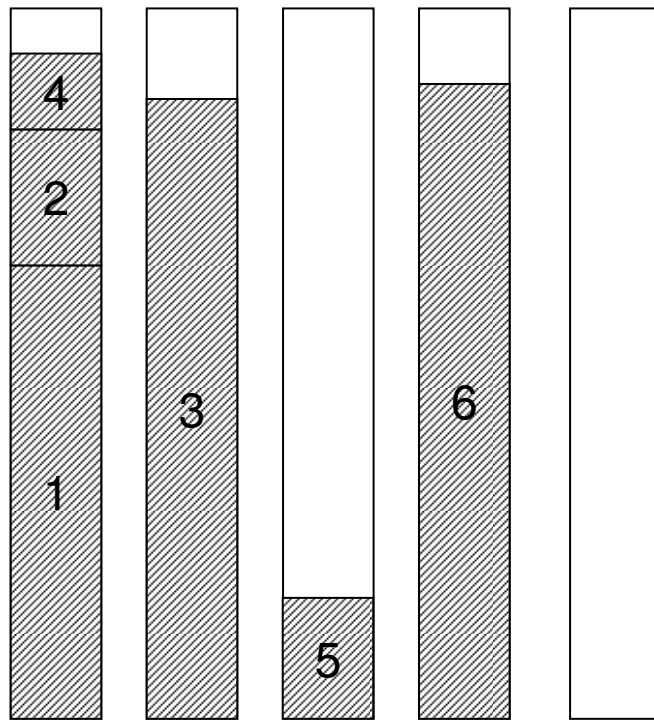


Building Heuristics: Cutting Stock

1. First-fit

Assign pieces, in order given, to first stock piece in which it fits.

Stock



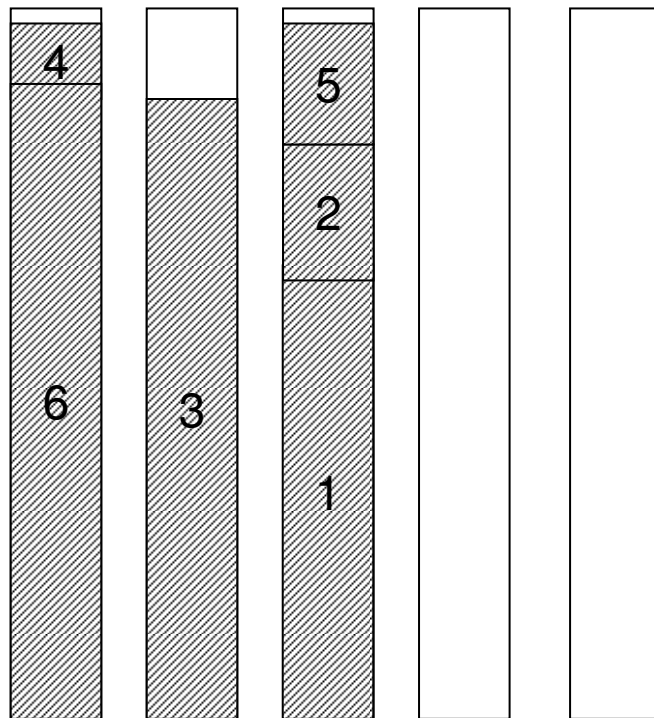
Demands

Building Heuristics: Cutting Stock

2. First-fit Decreasing

Assign pieces, in non-increasing order, to first stock piece in which it fits.

Stock



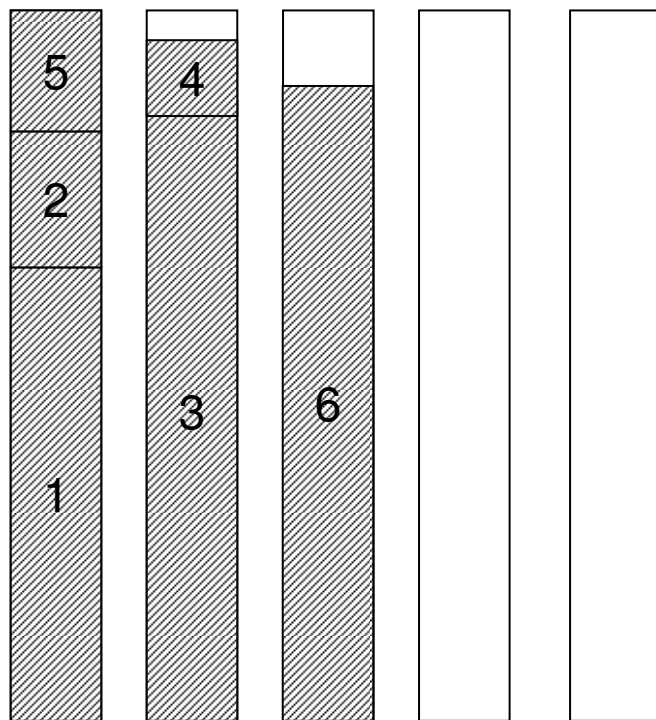
Demands

Building Heuristics: Cutting Stock

3. Best-fit (tightest fit)

Assign pieces, in order given, to stock piece in which it fits best, i.e. tightest – with least room to spare.

Stock



Demands

Local search heuristics

Combinatorial optimization problem:

$$\min f(x)$$

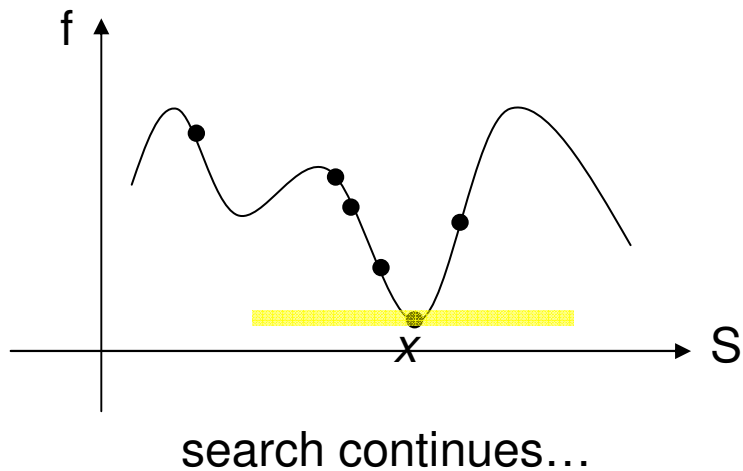
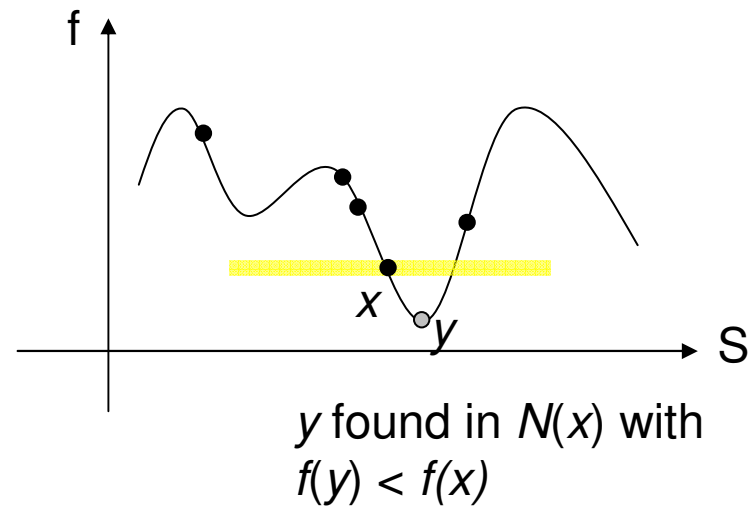
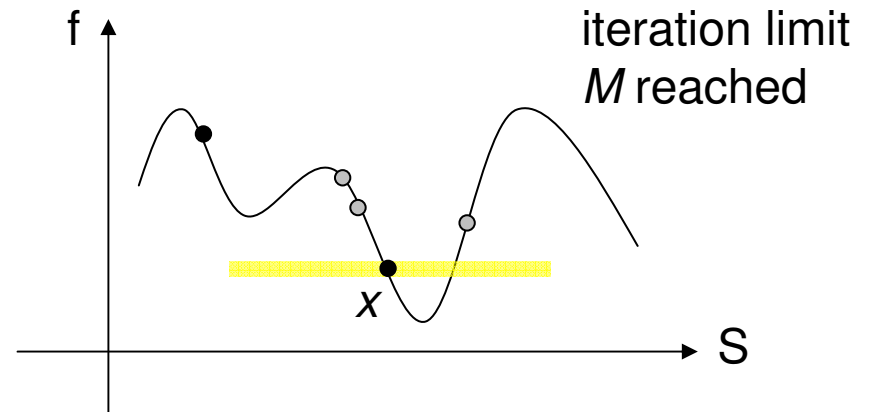
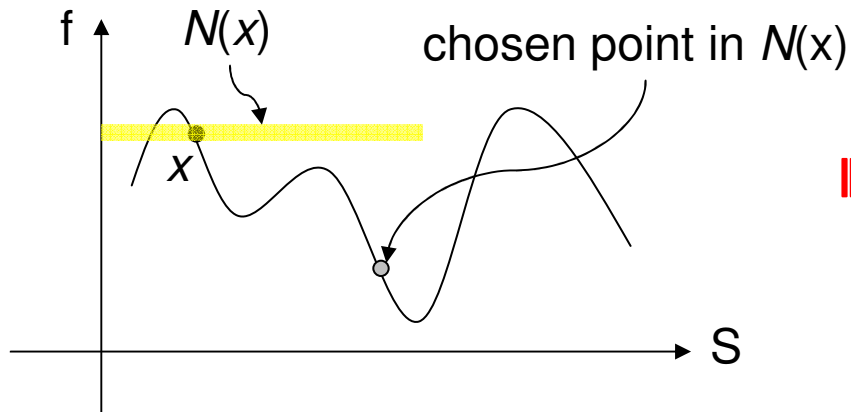
$$s.t. \quad x \in S$$

A neighbourhood of a point $x \in S$ is a set of “nearby” points in S , denoted by $N(x) \subseteq S$.

Iterative Improvement Algorithm

1. Choose initial $x \in S$. Set $i := 0$
2. Select $y \in N(x)$ at random.
3. If $f(y) < f(x)$ then
 set $x := y$, $i := 0$
otherwise $i := i + 1$.
4. If $i < M$ then go to Step 2, otherwise go to Step 5.
5. If for all $y \in N(x)$, $f(y) \geq f(x)$ then STOP: x is a local optimum.
 Else choose $y \in N(x)$ with $f(y) < f(x)$, go to Step 3.

Iterative Improvement Algorithm



Iterative “Steepest Descent” Algorithm

1. Choose initial $x \in S$.
2. For every $y \in N(x)$ calculate $f(y)$.
Set $x' := \operatorname{argmin} \{f(y) \mid y \in N(x)\}$
3. If $f(x') < f(x)$ then $x := x'$. Go to Step 2.
Otherwise STOP: x is locally optimum.

Optional. Random restart: if not had enough, go to Step 1.

Defining Neighbourhoods

The choice of neighbourhood is critical to the effectiveness of any local search method.

We will look at some examples for TSP.

Defining Neighbourhood: TSP

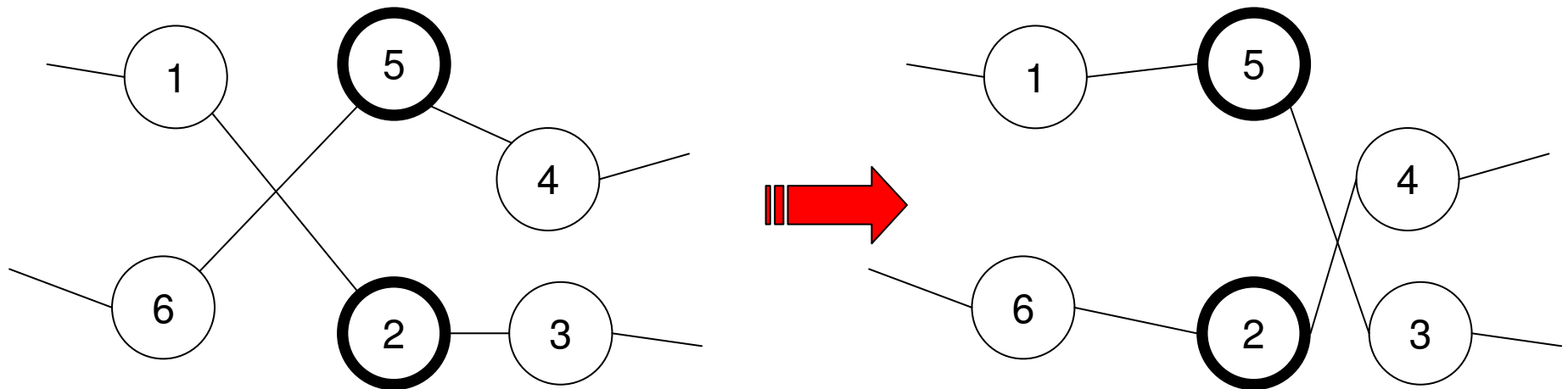
1. 2-SWAP OR 2-EXCHANGE NEIGHBOURHOOD

Exchange the positions of 2 cities in tour.

Neighbourhood of a tour is set of tours that can be obtained as a result of a 2-SWAP applied to the tour.

How many tours are there in this neighbourhood?

How many edges (arcs) are affected by a 2-SWAP?



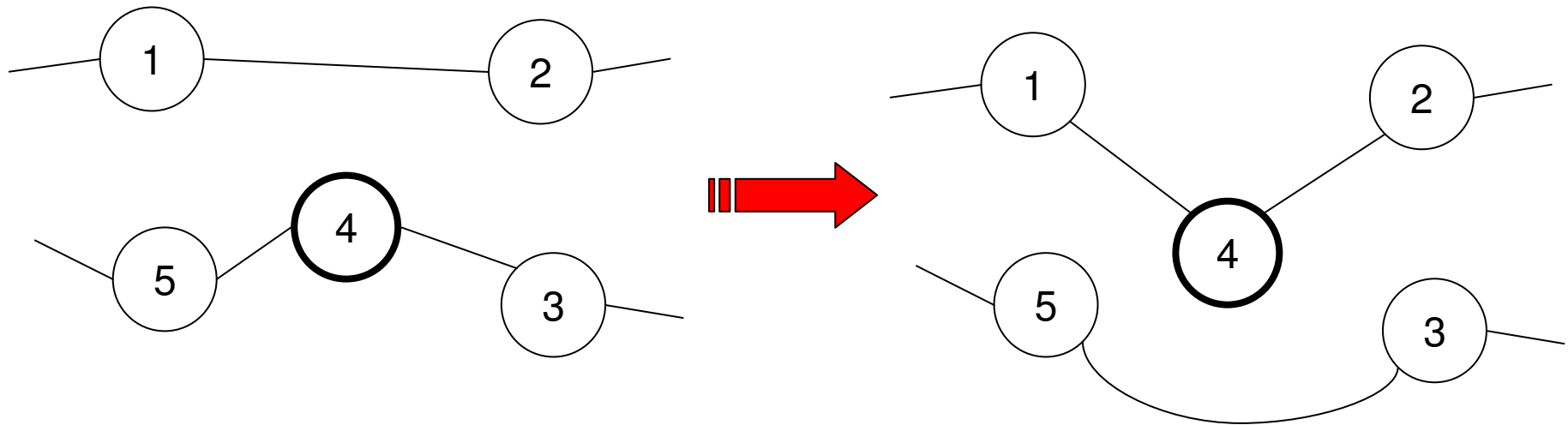
Defining Neighbourhood: TSP

2. 1-INSERTION NEIGHBOURHOOD

Select a city and move it into a new position in the tour.

How many edges are affected?

How big is the neighbourhood?



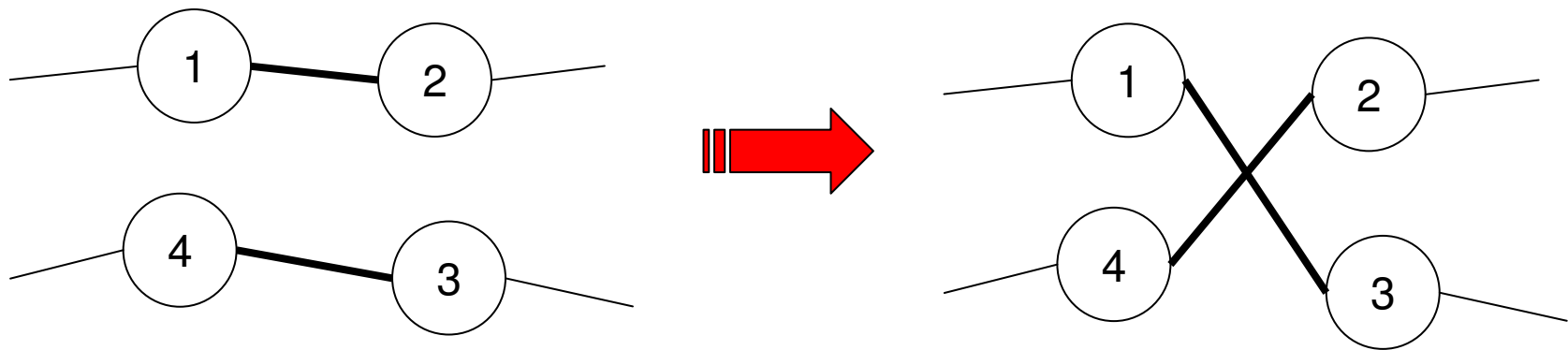
Defining Neighbourhood: TSP

3. 2-OPT NEIGHBOURHOOD

Exchange two edges in tour for two edges not in tour.

How many edges affected?

Size of neighbourhood?

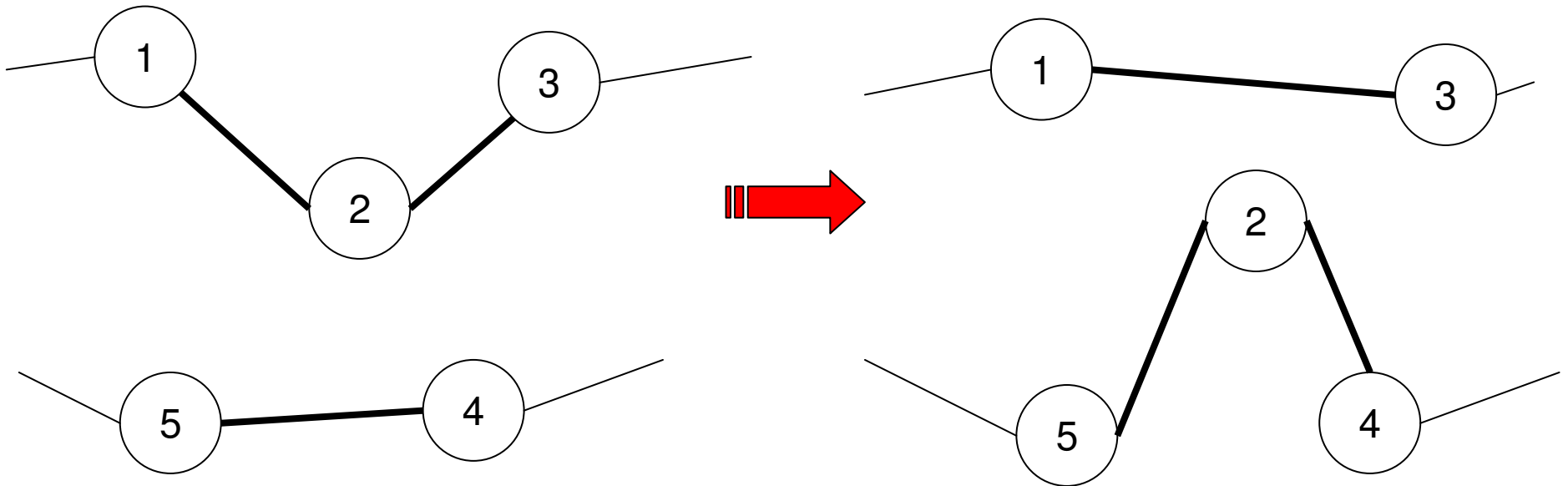


Defining Neighbourhood: TSP

4. 3-OPT NEIGHBOURHOOD

Exchange 3 edges in tour for 3 edges not in tour.

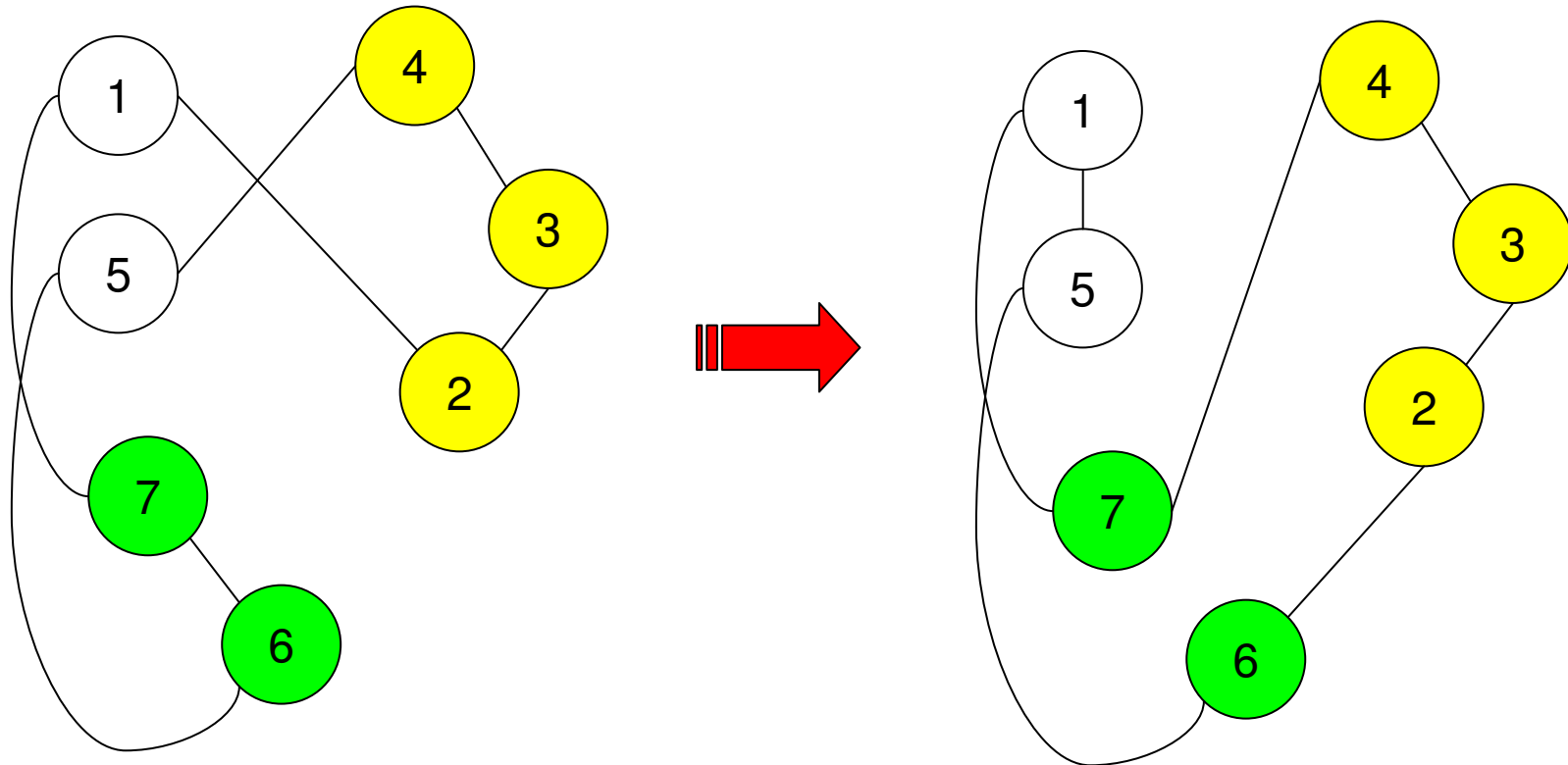
Note: 1-insertion is a special case of 3-Opt



Defining Neighbourhood: TSP

5. OR-OPT NEIGHBOURHOOD

Special case of 3-OPT in which strings of 3 adjacent cities in the tour are inserted between two other cities.



Defining Neighbourhood: TSP

6. LIN-KERNIGHAN (Variable r -OPT)

Decides dynamically at each iteration how big to make r , and then exchanges r edges in tour for r edges not in tour.

Note: The r -OPT “algorithm” for $r = 2,3$ and other neighbourhood algorithms usually refer to a “steepest descent” algorithm applied with the corresponding neighbourhood definition.

(Tour Improvement Heuristics)

Other algorithms e.g. simulated annealing, tabu search, can be based on any combination of these neighbourhood definitions.

Simulated Annealing

Given a sequence $\{T_0, T_1, \dots, T_F\}$, called the **cooling schedule**, such that $T_0 > T_1 > \dots > T_F$:

1. Set $t := T_0$, $i = 0$. Choose initial $x \in S$. $z := f(x)$, $x' := x$
2. If $t < T_F$ then STOP: x' is best.
Otherwise select $y \in N(x)$ at random.
3. If $f(y) < f(x)$, or $r \leq e^{-(f(x)-f(y))/t}$ where $r \in [0, 1]$ is random, then
 $x := y$
if $f(x) < z$ then $z := f(x)$, $x' := x$
4. $i := i + 1$, $t := T_i$
Go to Step 2.

Simulated Annealing: Cooling schedule

- **Initial temperature, T_0 :**

Select so that initially all uphill moves are accepted. e.g. $T_0 =$ max possible function variation in a neighbourhood

Example: For 2-OPT, choose $T_0 =$ length of 2 longest edges – length of 2 shortest edges

- **Geometric cooling:**

$T_{i+1} := \sigma T_i$ for some $\sigma \in (0, 1)$.

σ close to 1 implies slow cooling.

- **Final Temperature, T_F :**

$T_F = \varepsilon / \log(N)$ where $\varepsilon > 0$ small and $N = \max_{x \in S} |N(x)|$

e.g. for 2-OPT choose $T_F = \varepsilon / \log(1/2n(n-3))$

Good Neighbourhoods

1. Not too large, or too time-consuming to enumerate (definitely polynomial size).
2. Between any $x, y \in S$ there is a sequence $\{z_0=x, z_1, z_2, \dots, z_k = y\}$ with $z_i \in S$ for all $i=0, \dots, k$ such that $z_i \in N(z_{i-1})$ for all $i = 1, \dots, k$. (Connectivity Property)
3. The variance of objective function values across neighbourhoods should be small.
4. Having calculated $f(x)$, calculating $f(x')$ for $x' \in N(x)$ is very quick.

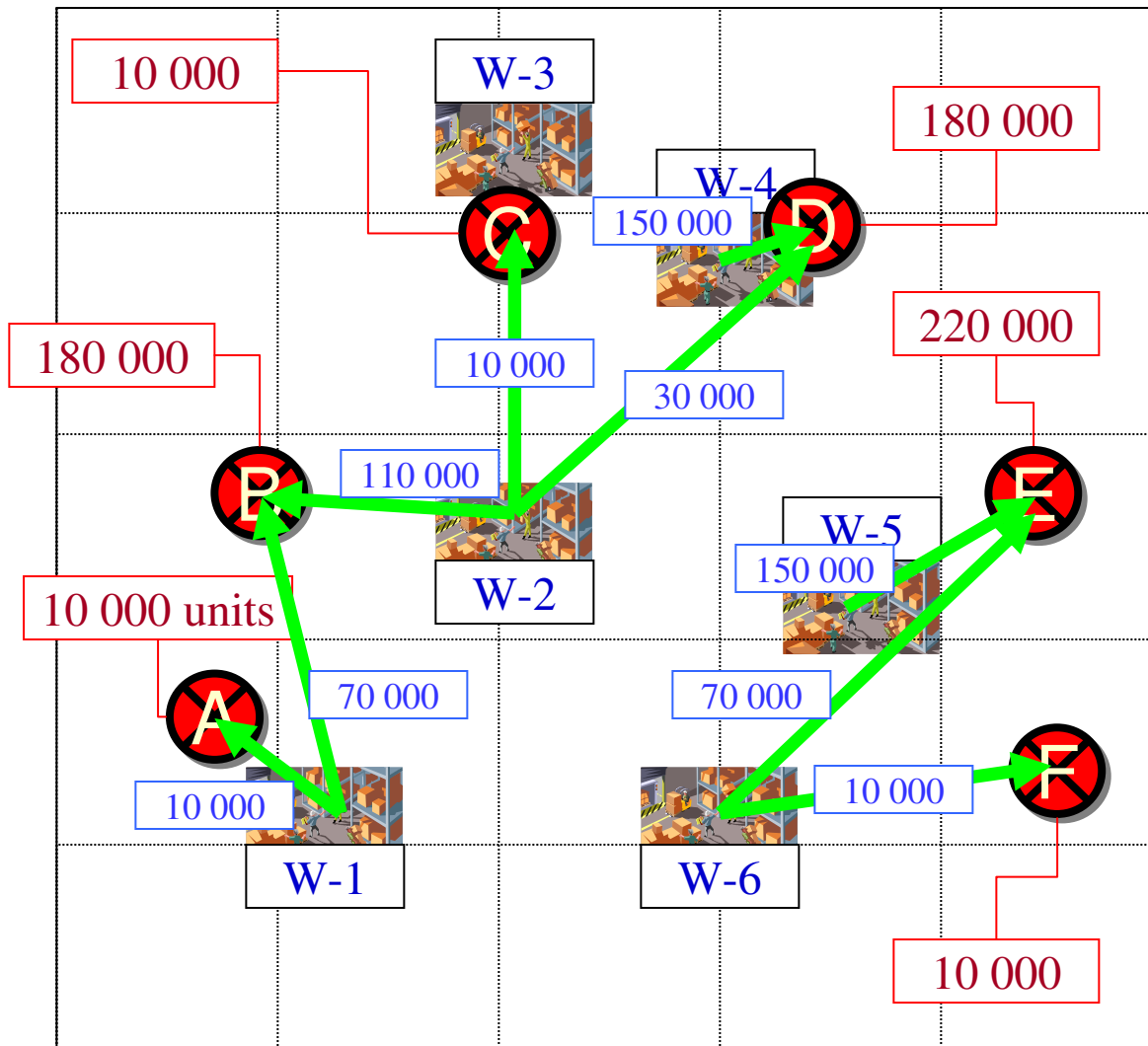
Scheduling on parallel machines



Possible neighbourhood:

- move job from one machine to another
- swap jobs between machines
- swap position of 2 jobs on a single machine

Facility Location



Possible neighbourhood

- Open a new facility
- Close a facility
- Open a new facility and close an old one

Check: Is calculating objective value easy?

Multiple supplier variant:

solve LP

Uncapacitated single supplier: greedy

Capacitated single supplier: binary LP

Add to neighbourhood:

- move client from one facility to another
- swap clients between facilities

Other methods...

- Do your own research/readings on:
 - Simulated annealing (other variants)
 - Tabu search
 - Genetic algorithm
 - Ant colony optimisation, Particle swarm optimisation
 - Constraint programming