University of Nottingham
UK | CHINA | MALAYSIA

COM2001/2011 Artificial Intelligence Methods

Lecture 1

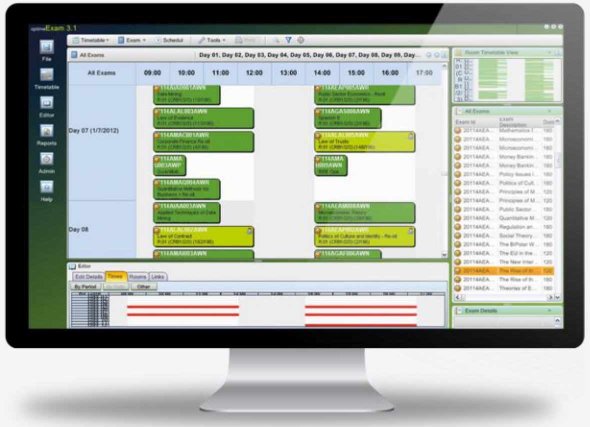
Prof Ender Özcan
Computer Science, Office: C86
Ender.Ozcan@nottingham.ac.uk
www.nottingham.ac.uk/~pszeo/

What Do the Following Things Have in Common?

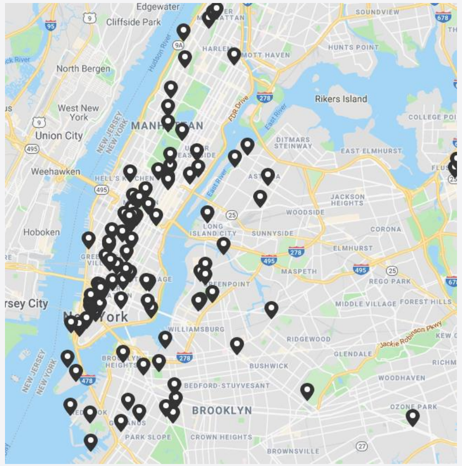
Timetables

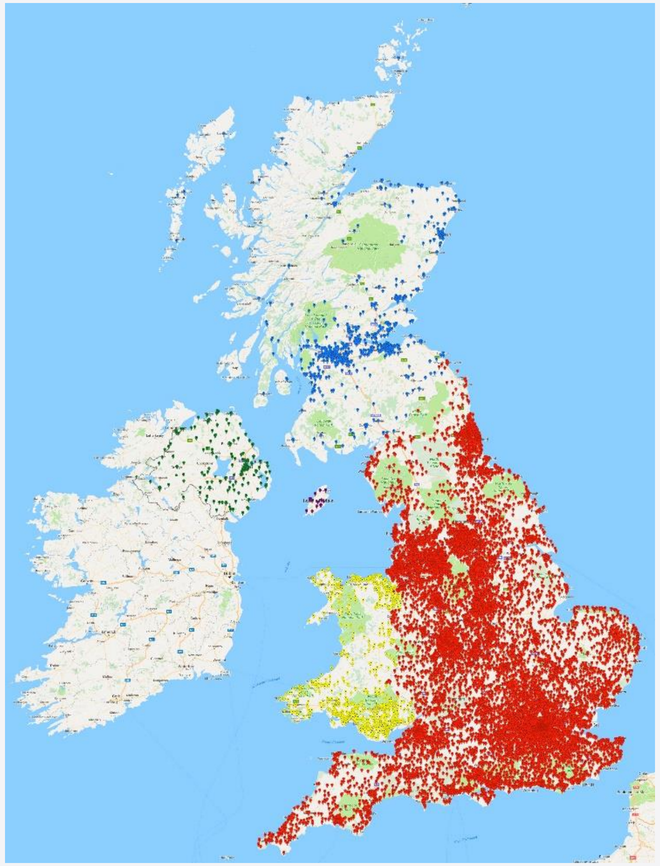
Nurse Rosters

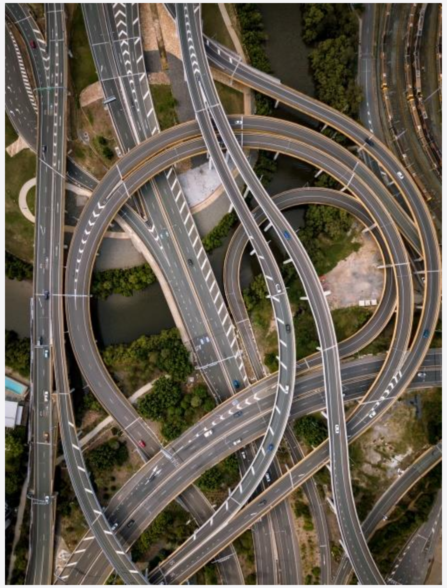
Exam Timetables



Routing


Edgewater
Cliffside Park
North Bergen
West New York
Union City
Weehawken
Hoboken
MANHATTAN
Rikers Island
HUNTS POINT
MOTT HAVEN
DITMARS STEINWAY
EAST ELMHURST
JACKSON HEIGHTS
WOODSIDE
CORONA
SUNNYSIDE
ELMHURST
MASPETH
REGO PARK
MIDDLE VILLAGE
FOREST HILLS
GREENPOINT
WILLIAMSBURG
RIDGEWOOD
GLENDALE
BUSHWICK
WOODHAVEN
BEDFORD-STUYVESANT
BROOKLYN
OZONE PARK
PARK SLOPE
CROWN HEIGHTS
BROWNSVILLE
COLLEGE POINT
SOUNDVIEW

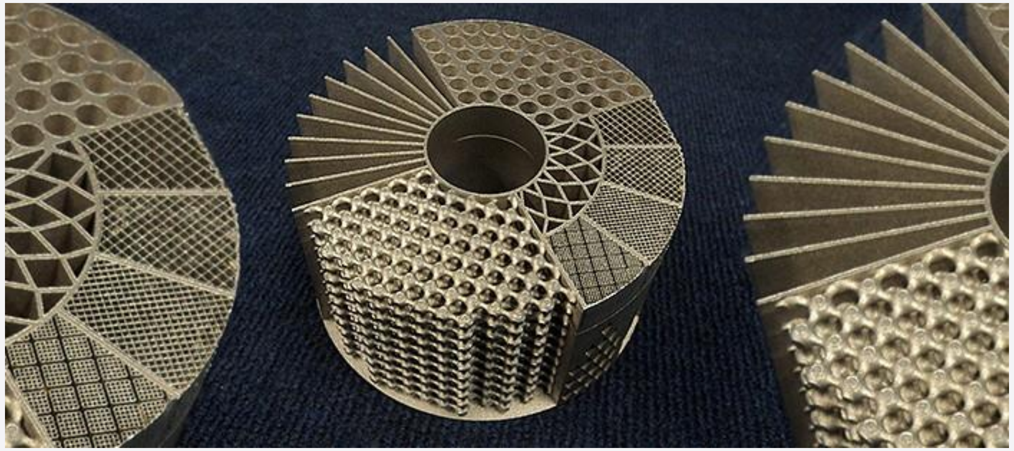




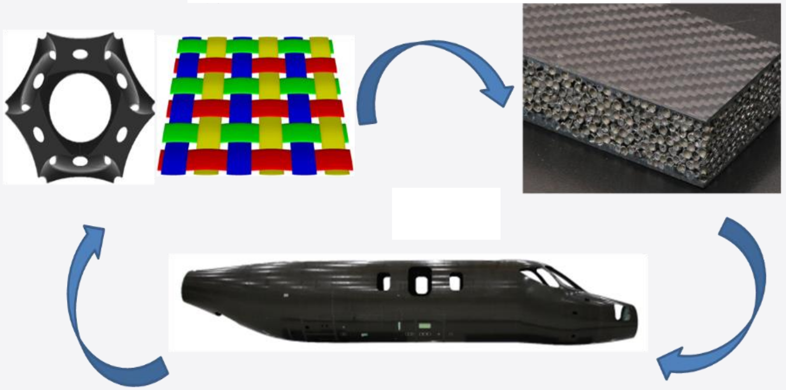
Supply Chains

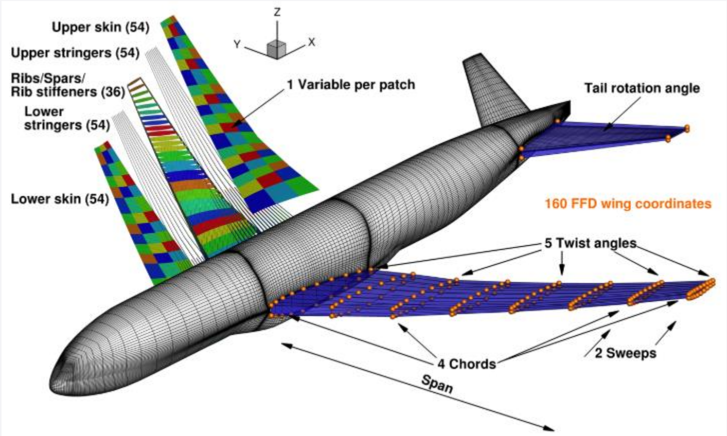
Wind-farms

Additive Manufacturing (3D Printing)



Engineering Design





Modern Heuristic Optimisation/ Search Techniques Can

- do automated timetabling
- decrease carbon emissions and other costs of routing
- reduce operational cost of supply chains
- increase energy production via drawing wind-farm layouts
- improve packing and scheduling workflow for additive manufacturing
- optimise multifunctional structures combining composite and porous layers

Course Details
COMP2011 (10 cr)

- Module web page can be reached from:
  https://moodle.nottingham.ac.uk/course/view.php?id=140203
- Module Activities:
  - 2 hour lectures per week
  - Asynchronous Lecture Engagement Activities:
    - Formative quizzes for self-assessment (1 week to respond)
    - Discussion forums for collaborative problem solving/peer support and more
- Assessment
  - Examination [100%]

Recordings

Course Details
COMP2001 (20 cr)

- Module web page can be reached from:
  https://moodle.nottingham.ac.uk/course/view.php?id=140203
- Module Activities:
  - 2 hour lectures & 2 hour labs per week
  - Asynchronous Lecture Engagement Activities:
    - Formative quizzes for self-assessment
    - Discussion forums for collaborative problem solving/peer support and more

echo 360

Recordings

- Assessment
  - Examination [50%] + Coursework [50%]
    [20%] in-lab exam (week 7) + [30%] project with demo

Summary of Content (provisional)

- The main aim of this module is to provide a sound understanding of a wide range of fundamental concepts and techniques of Artificial Intelligence (used for intelligent decision support), focusing on
  - modern heuristic search/optimisation techniques, including
    - Metaheuristics, such as Iterated Local Search, Simulated Annealing, Tabu Search and Evolutionary Algorithms and hyper-heuristics
  - and at the introductory level;
    - Fuzzy sets, planning for robotics, symbolic AI

Main Educational Aims

- Students will have a sound understanding of the selected modern (heuristic) search techniques in Artificial Intelligence, and some selected AI topics
- Students will understand the methods and techniques that are available as an aid in automated decision making/optimisation.
- Students will be acquainted with a number of applications and will understand how software tools are designed to solve them.

Resources

- Metaheuristics: From Design to Implementation, El-Ghazali Talbi, DOI: 10.1002/9780470496916, John Wiley, ISBN: 9780470278581 [PDF from ResearchGate] (this version is publicly available now)
- Search methodologies: introductory tutorials in optimization and decision support techniques - Edmund Burke, Graham Kendall c2014 [copy found over the internet in PDF]
- Stochastic local search: foundations and applications - Holger H. Hoos, Thomas Stützle 2005 [Public access to an old version]
- Automated scheduling and planning: from theory to practice - A. Şima Etaner-Uyar, Ender Özcan, Neil Urquhart 2013
- Scheduling: theory, algorithms, and systems - Michael Pinedo 2016 [5th Edition in PDF]
- NOT REQUIRED FOR THIS MODULE
- Lecture notes and links to the relevant papers will be provided

Examinable Material

- Lecture (and Lab*) Notes
  - What’s written on the slides
- Lecture (and Lab*) Content
  - What’s said
  - What’s written on the whiteboard
- Lecture (and Lab*) Exercises
  - Questions: Asked during a lecture
  - Answers: Said or written during a lecture

- *For the COMP2001 students

Provisional Topics

- Introduction: Heuristic Search/Optimisation, Search Paradigms
- Components of Search Methodologies & Hill Climbing
- Metaheuristics, Single point based search:
  - Iterated Local Search,
  - Tabu Search
- Move Acceptance in Metaheuristics and Parameter Setting Issues:
  - Late Acceptance,
  - Great Deluge,
  - Simulated Annealing
- Evolutionary Algorithms:
  - Genetic Algorithms,
  - Memetic Algorithms,
  - Multimeme Memetic Algorithms
- Hyper-heuristics:
  - classification,
  - cross-domain search,
  - HyFlex/Chesc
- Selection hyper-heuristics
- Generation hyper-heuristics
- Fuzzy sets, Planning,
- Symbolic AI/*Advanced topics*
- *Revision*

Provisional Computing Sessions

w0 Preparation for the following sessions

w1 Comparison of Hill Climbing Heuristics

w2 Iterated Local Search

w3 Simulated Annealing and Cooling Schedules

w4 Local Search in Memetic Algorithms

w5 A Multimeme Memetic Algorithm

w6 In-lab exam/HyFlex training

w7 A Selection Hyper-heuristic for MAX-SAT

w8 A Selection Hyper-heuristic for Cross-domain Search

w9 An Adaptive Great Deluge Move Acceptance for Cross-domain Search

University of Nottingham
UK | CHINA | MALAYSIA

Computational Optimisation & Learning Lab

Part 1. Preliminaries

COM2001/2011
Artificial Intelligence Methods

Ender Özcan

Lecture 1

Definition – Decision Support

- This term is used often and in a variety of contexts related to decision making.
- Multidisciplinary:
  - Artificial Intelligence,
  - Operations Research,
  - Decision Theory,
  - Decision Analysis,
  - Statistics,...

Definition – Systems

- Degree of dependence of systems on the environment
  - Closed systems are totally independent
  - Open systems dependent on their environment
- Evaluations of systems
  - *System effectiveness*: the degree to which goals are achieved, i.e. result, output
  - *System efficiency*: a measure of the use of inputs (or resources) to achieve output, e.g., speed

Solving Problems by Searching

- Search for paths to goals
  - efficiently finding a set of actions that will move from a given initial state to a given goal
  - central to many AI problems (e.g., game playing, path finding)
  - typical algorithms are the depth first search, breadth first search, uniform cost search, A*, branch and bound
- Search for solutions (optimisation)
  - more general class than searching for paths to goals.
  - efficiently finding a solution to a problem in a large space of candidate solutions
  - subsumes the first type, since a path through a search tree can be encoded as a candidate solution

Solving a Single Objective Optimisation Problem

“Consider everything. Keep the good. Avoid evil whenever you notice it."

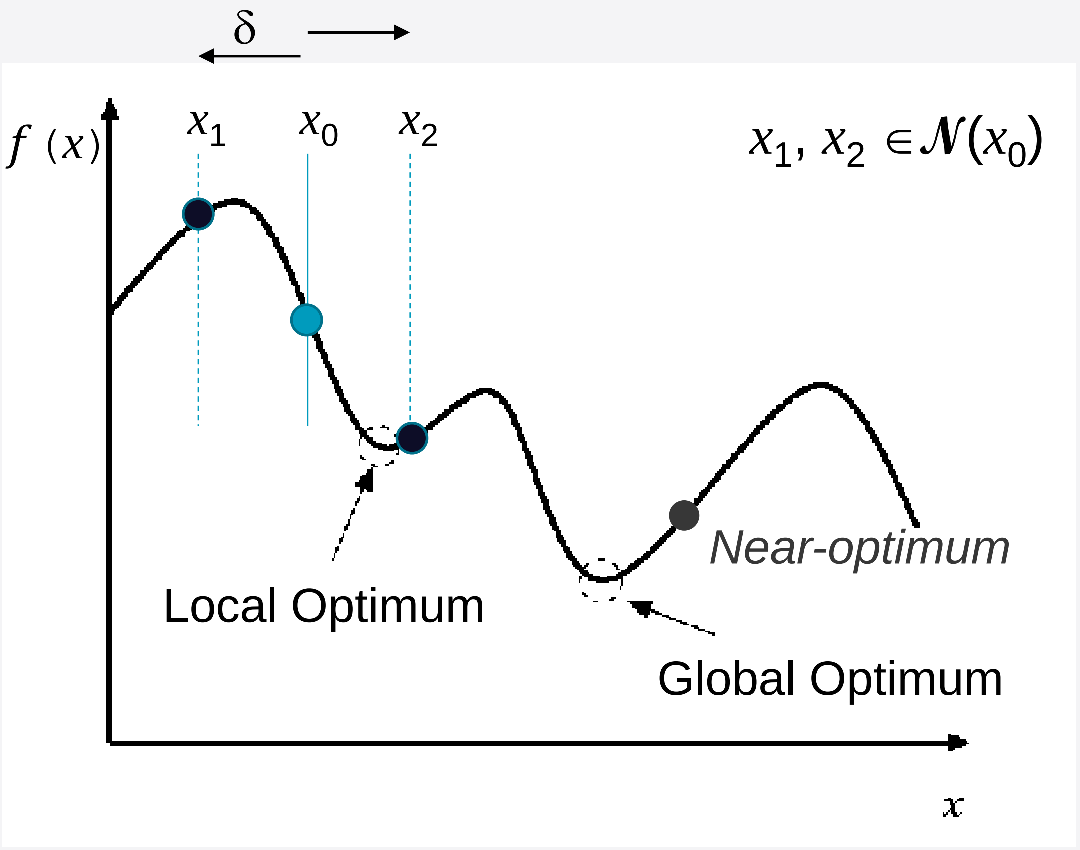
- Solving a mathematical optimisation problem:
- First choose a quantity (typically a function of several variables – objective function) to be maximised or minimised, which might be subject to one or more constraints (constraint optimisation).
  - maximise/minimise $z = f(\boldsymbol{X}),\ \{g_i(\boldsymbol{X}) \leq \boldsymbol{b_i}\ ,\} \ (= | \geq)$
  - where $\boldsymbol{X}$ is a vector of variables < x_1 , x_2 ,…, x_n>
- Next choose a mathematical or search method to solve the optimisation problem (searching the space of solutions and detecting the best/optimal solution)

Definition – Global Optimisation

- Global optimisation is the task of finding the absolutely best set of admissible conditions to achieve your objective, formulated in mathematical terms.
- Fundamental problem of optimisation is to arrive at the best possible (optimal) decision/solution in any given set of circumstances.
- In most cases “the best” (optimal) is unattainable

Global vs Local Optimum

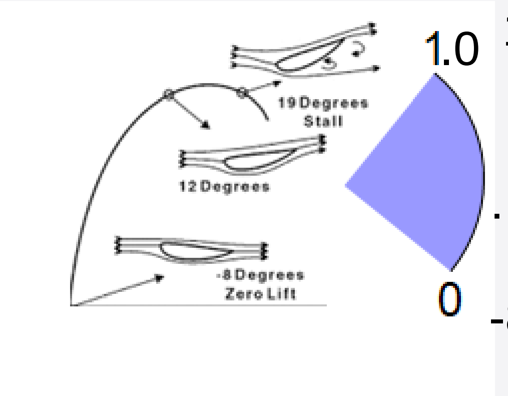
- Global Optimum: better than all other solutions (best)
- Local Optimum: better than all solutions in a certain neighbourhood, $\mathcal{N}$



δ

Search in
Continuous vs Discrete Space

- Find the optimum setting for the angle of the wings of a race car providing the best performance



19.0 Degrees

…

-8.0 Degrees

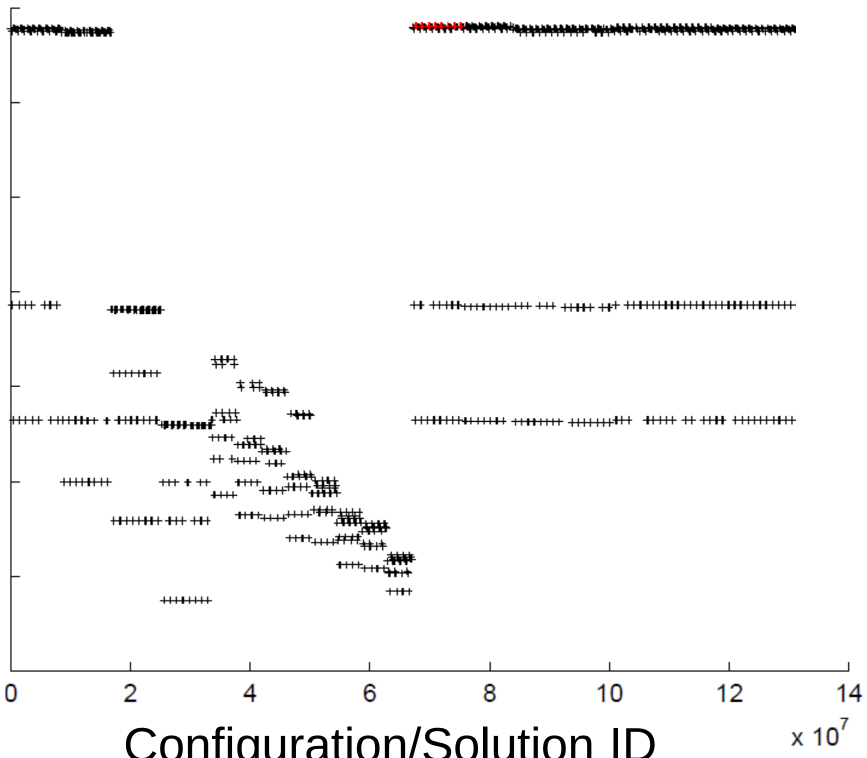
- Organise the optimum (minimum) number of tour busses for groups of 30, 10, 60, 40, 50, 20,…, 35 tourists (a tour bus has 70 seats)

Search in Continuous vs Discrete Space

Performance

Real-valued (parameter)/
Continuous Optimisation

Average bus fullness



Configuration/Solution ID

Organising tour busses

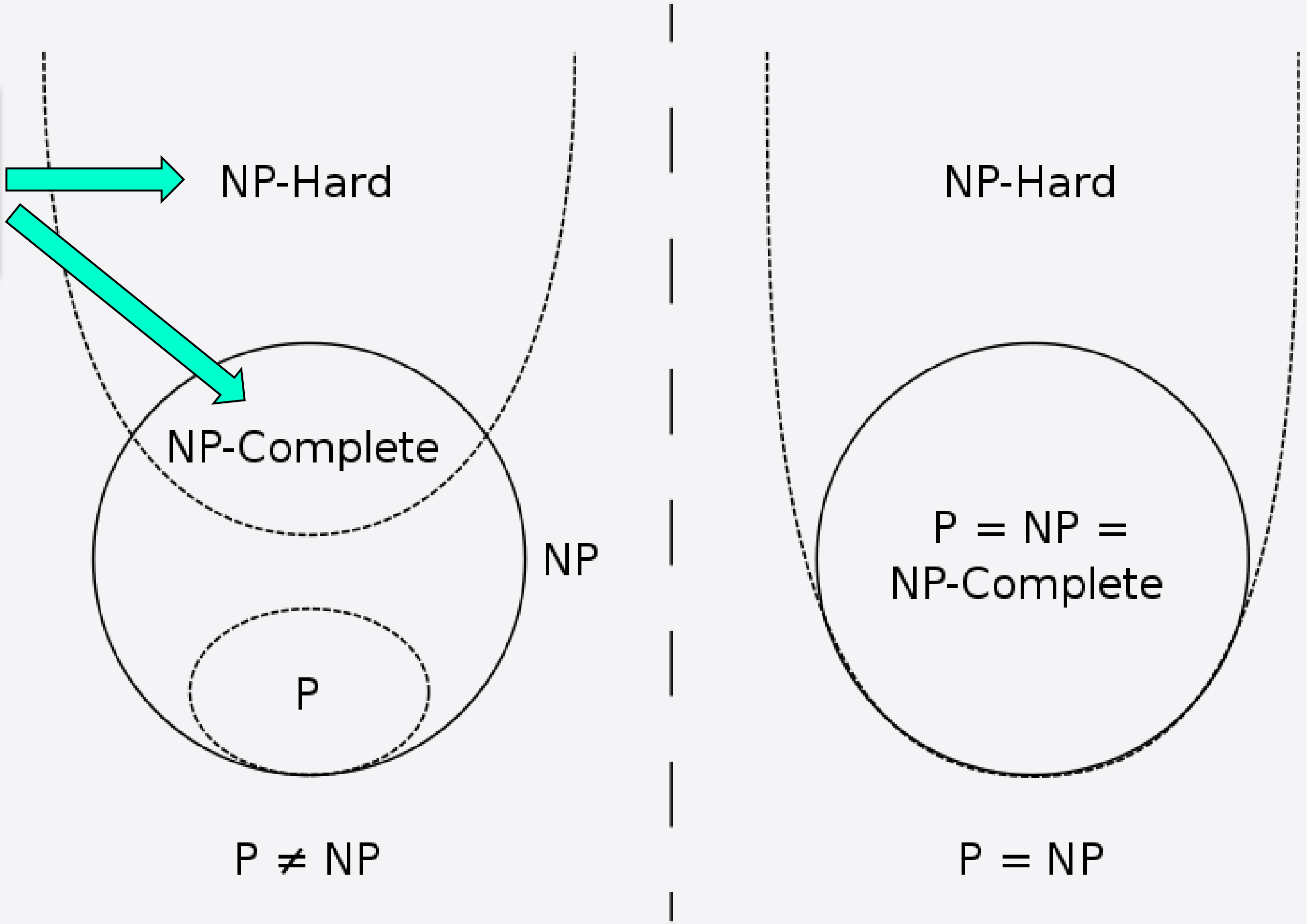
Definition – Problem and Problem Instance

- Problem refers to the high level question or optimisation issue to be solved.
- An instance of this problem is the concrete expression, which represents the input for a decision or optimisation problem.
- Example: Optimal assignment of groups to busses (minimising the number of busses) is an optimisation (minimisation) problem
  - Optimal assignment of 3 groups of 10, 15, 43 to busses, each with 45 seats and company having 10 busses at max is an instance of this problem
  - Optimal assignment of 5 groups of 19, 25, 30, 30, 45 to busses, each with 60 seats and company having 10 busses at max is another instance of this problem

Analysis of Algorithms 101: Problem Classes

COMP2009
COMP3001

Many real-world problems are in here



Combinatorial Optimisation Problems

- Require finding an optimal object from a finite set of objects
- For NP-hard COPs, the time complexity of finding solutions can grow exponentially with instance size.
- NP-hard (many COPs)
  - Determining the optimal route to deliver packages
  - Optimal scheduling of shifts/jobs subject to various constraints
  - Optimal packing of items
  - See slides #2-8
  - and more...

Optimisation/Search Methods

- Optimisation methods can be broadly classified as:
- Exact/Exhaustive/Systematic Methods
  - E.g., Dynamic Programming, Branch&Bound, Constraint Satisfaction,…
- Inexact/Approximate/Local Search Methods
  - E.g., heuristics, metaheuristics, hyper-heuristics,…

Search Paradigms I

- Single point (trajectory) based search vs. Multi-point (population) based search

- Perturbative
  - complete solutions

- Constructive
  - partial solutions

Search Paradigms II

$\boxed{\text{deterministic} \longleftrightarrow \text{stochastic}}$

systematic $\longleftrightarrow$ $\boxed{\text{local search}}$

$\boxed{\text{sequential}}$ $\longleftrightarrow$ parallel

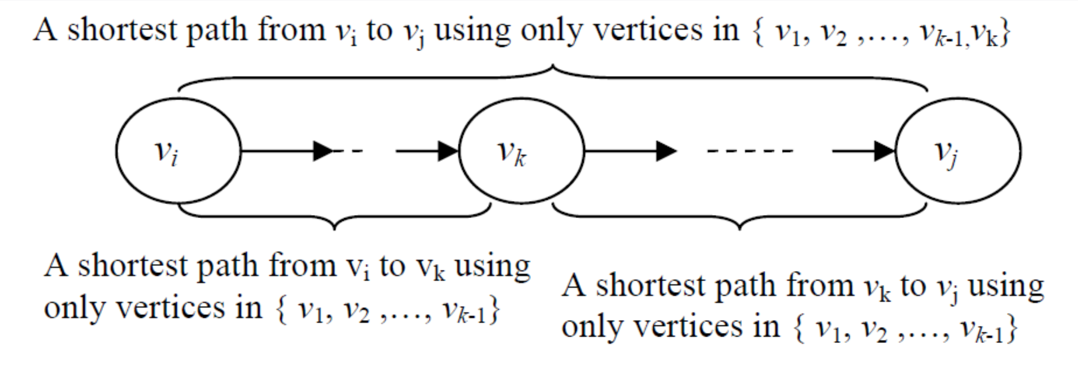
$\boxed{\text{single objective}}$ $\longleftrightarrow$ multi-objective

[See http://www.cs.ubc.ca/~hoos/SLS-Internal/ch1.pdf pp.23-30]

Example Optimisation Techniques I

<u>Exact/Exhaustive/Deterministic Systematic Methods</u>

- Dynamic Programming

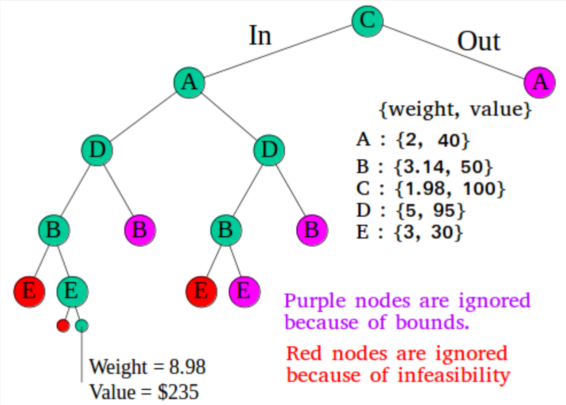


- Constraint Satisfaction

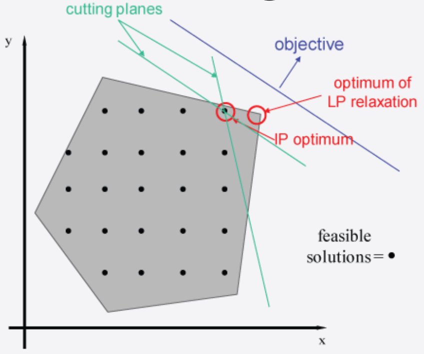
Limitations:

- Only work if the problem is structured – in many cases for small problem instances
- Quite often used to solve sub-problems

- Branch & Bound



- Math programming
  - ILP
  - MILP



Cutting Planes

Example I – Bin Packing Problem Instance

- maximise/minimise $z = f(\boldsymbol{X}),\ \{g_i(\boldsymbol{X}) \leq \boldsymbol{b_i}\ ,\}\ (= \mid \geq)$
- where $\boldsymbol{X}$ is a vector of variables $< x_1\ , x_2\ , \ldots, x_n >$

x_1 x_2 ....

different notation
same logic $\rightarrow$ a_1 a_2 ....

- V=524
- n=33
- Item sizes (33 items to pack): 85, 442, 10, 10, 10, 10, 10, 10, 252, 252, 252, 252, 252, 252, 252, 9, 9 , 127, 127, 127, 127, 127, 106, 106, 106, 106, 12, 12, 12, 84, 46, 37
- How can we solve this problem using an exact method?

Example I – Math Programming Solution

Model

$$\text{minimize } B = \sum_{i=1}^{n} y_i$$

$$\text{subject to } B \geq 1,$$

$$\sum_{j=1}^{n} a_j x_{ij} \leq V y_i, \forall i \in \{1, \ldots, n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in \{1, \ldots, n\}$$

$$y_i \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\} \forall j \in \{1, \ldots, n\}$$

where $y_i = 1$ if bin i is used and $x_{ij} = 1$ if item j is put into bin i.[5]

IBM CPLEX

```
{string} bins = ...;
{string} items = ...;
float itemSize[bins] = ...;
float binCapacity[bins] = ...;
range r = 0..1;
dvar int X[items][bins] in r;

minimize sum(u in bins) pow(((sum(a in items) itemSize[a]*X[a][u])), 2);

subject to {
    /* Item in only one bin constraint */
    forall(i in items)
        sum(b in bins)
          X[i, b] == 1;

    /* Capacity Constraint */
    forall(b in bins)
          sum(i in items) X[i][b]*itemSize[i] <= binCapacity[b];
};

execute DISPLAY {
  writeln("Mapping:");
  for (var i in items) {
    for (var b in bins) {
      if (X[i][b] == 1) {
        writeln("Item " + i + " is in bin " + b);
      }
    }
```

itemSize = #[1: 85, 2: 442, 3: 10, 4: 10, **…**, 32: 46, 33: 37]#;
binCapacity = #[1: 524, 2: 524, 3: 524, **…**, 33: 524]#;

Example I – Performance Comparison of Different Branch and Bound Algorithms for Optimal Bin Packing

- The solution time increases with the size of the problem instance substantially

N	Optimal	L2 bound	% Optimal		Martello + Toth		Bin Completion		Ratio
			FFD	BFD	Nodes	Time	Nodes	Time	Time
5	3.215	3.208	100.000%	100.000%	0.000	7	.013	6	1.17
10	5.966	5.937	99.515%	99.541%	.034	15	.158	13	1.15
15	8.659	8.609	99.004%	99.051%	.120	25	.440	19	1.32
20	11.321	11.252	98.570%	98.626%	.304	37	.869	27	1.37
25	13.966	13.878	98.157%	98.227%	.741	55	1.500	36	1.53
30	16.593	16.489	97.790%	97.867%	2.146	87	2.501	44	1.98
35	19.212	19.092	97.478%	97.561%	7.456	185	4.349	55	3.36
40	21.823	21.689	97.153%	97.241%	39.837	927	8.576	73	12.70
45	24.427	24.278	96.848%	96.946%	272.418	6821	20.183	103	66.22
50	27.026	26.864	96.553%	96.653%	852.956	20799	57.678	189	110.05
55	29.620	29.445	96.304%	96.414%	6963.377	200998	210.520	609	330.05
60	32.210	32.023	96.036%	96.184%	58359.543	2153256	765.398	2059	1045.78
65	34.796	34.598	95.780%	95.893%			11758.522	28216	
70	37.378	37.167	95.556%	95.684%			16228.245	41560	
75	39.957	39.736	95.322%	95.447%			90200.736	194851	
80	42.534	42.302	95.112%	95.248%			188121.626	408580	
85	45.108	44.866	94.854%	94.985%			206777.680	412576	
90	47.680	47.428	94.694%	94.832%			1111759.333	2522993	

See http://www.aaai.org/Papers/AAAI/2002/AAAI02-110.pdf

25 days

Example Optimisation Techniques II

Inexact/Approximate Methods

- The focus of this course will be:
  - utilising modern optimisation/search techniques in particular heuristics, metaheuristics and hyper-heuristics to solve “discrete” combinatorial optimisation problems effectively and efficiently, facilitating the use of data, models, and structured decision processes for decision support.

University of Nottingham
UK | CHINA | MALAYSIA

Computational Optimisation & Learning Lab

Part 2. Heuristic Search/Optimisation

COM2001/2011
Artificial Intelligence Methods

Ender Özcan

Lecture 1

Heuristic Search Methods

A **heuristic** is a rule of thumb method derived from human intuition.

- A heuristic is a problem dependent search method which seeks good, i.e. near-optimal solutions, at a reasonable cost (e.g. speed) without being able to guarantee optimality.
- Good for solving ill-structured problems, or complex well-structured problems (large-scale combinatorial problems that have many potential solutions to explore)

Example I – Bin Packing Problem Instance

- V=524
- n=33
- Item sizes (33 items to pack): 85, 442, 10, 10, 10, 10, 10, 10, 252, 252, 252, 252, 252, 252, 252, 9, 9 , 127, 127, 127, 127, 127, 106, 106, 106, 106, 12, 12, 12, 84, 46, 37
- How can we solve this problem using a heuristic (inexact method)?

How would you do it?

Example I – A Heuristic Solution
The problem with heuristics?

a_1	442	252	127	106	37	10	10
a_2	252	252	127	106	37	10	9
	252	252	127	85	12	10	9
	252	127	106	84	12	10	
	252	127	106	**46**	12	10	

Largest item first fit [A Deterministic Local Search/Greedy Constructive Heuristic Algorithm]

Sort the items by its *size* in decreasing order, then place each item into the first bin that will accommodate that object. The bins are also sorted in the order they came into use.

Largest item first fit rule/heuristic

	Bin1	Bin2	Bin3	Bin4	Bin5	Bin6	Bin7
442	442						
252		252					
252		252					
252			252				
252			252				
252				252			
252				252			
252					252		
127					127		
127					127		
127						127	
127						127	
127						127	
106						106	
106							106
106							106
106							106
85							85
84							84
46 – removed							
37						37	
37							37
12	12						
12	12						
12	12						
10		10					
10		10					
10			10				
10			10				
10				10			
10				10			
9					9		
9					9		
	524	524	524	524	524	524	524

Instance#1

Bin1	Bin2	Bin3	Bin4	Bin5	Bin6	Bin7	Bin8
442							
	252						
	252						
		252					
		252					
			252				
			252				
				252			
				127			
				127			
					127		
					127		
					127		
					106		
						106	
						106	
						106	
						85	
						84	
37							
37							
	12						
		12					
			12				
				10			
					10		
					10		
					10		
						10	
						10	
						9	
							9
516	516	516	516	516	517	516	9

Instance#2

A Case Study:
Traveling Salesman Problem
Stochastic Local Search – Perturbative vs Constructive Algorithms

Computational Optimisation & Learning Lab

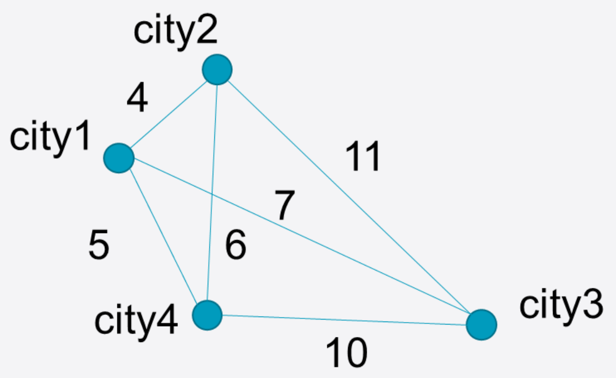
Travelling Salesman Problem (TSP)

- "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" – NP hard.
- Underpins many real world problems: Planning, logistics, vehicle routing, telecommunication, manufacturing of microchips, genome sequencing, clustering of data arrays, scheduling, cutting&packing and more…

  - maximise/minimise $z = f(\boldsymbol{X}),\ \{g_i(\boldsymbol{X}) \leq \boldsymbol{b_i},\} (= | \geq)$
  - where $\boldsymbol{X}$ is a vector of variables < x_1 , x_2 ,…, x_n >

Instance: city1, city2, city3, city4, where n=4
Example solutions:

- **<city3, city1, city2, city4> : 27**
- **<city2, city1, city4, city3> : 30**
- …



Need for Search Methodologies (e.g. Heuristics, Metaheuristics) – Example

- Travelling salesman problem with N cities
- N=4, 24
- N=5, 120
- N=7, 5 040
- N=10, 3 628 800
- N=81, 5.797×10^{120}

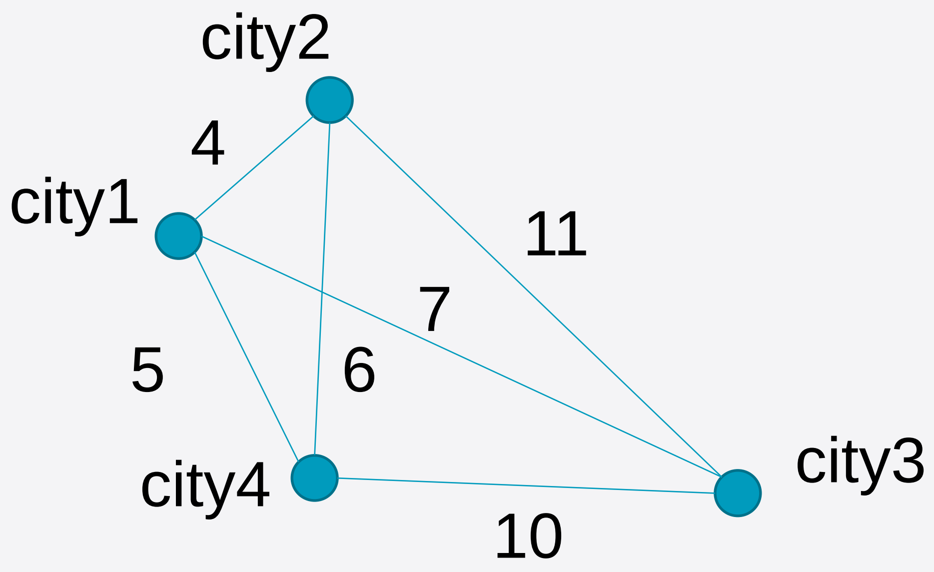
- Number of configurations to search from is N!
- Number of particles in the universe is in between 10^{72} – 10^{87}
- US Frontier (2022): ~1,102.01 petaFLOPS (one thousand million (10^{15}) floating-point operations per second) – ~1.67×10^{95} years (from [TOP500] project).

Examples – Heuristics for TSP

- The nearest neighbour (NN) algorithm
  - Constructive (Stochastic, Systematic)

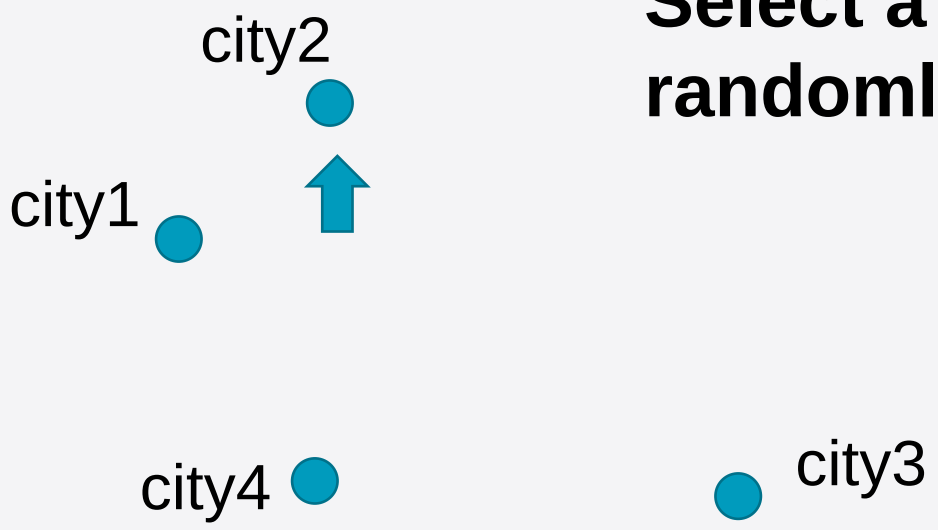
- Pairwise exchange (2-opt)
  - Perturbative (Stochastic, Local Search)

The nearest neighbour (NN) algorithm



The nearest neighbour (NN) algorithm

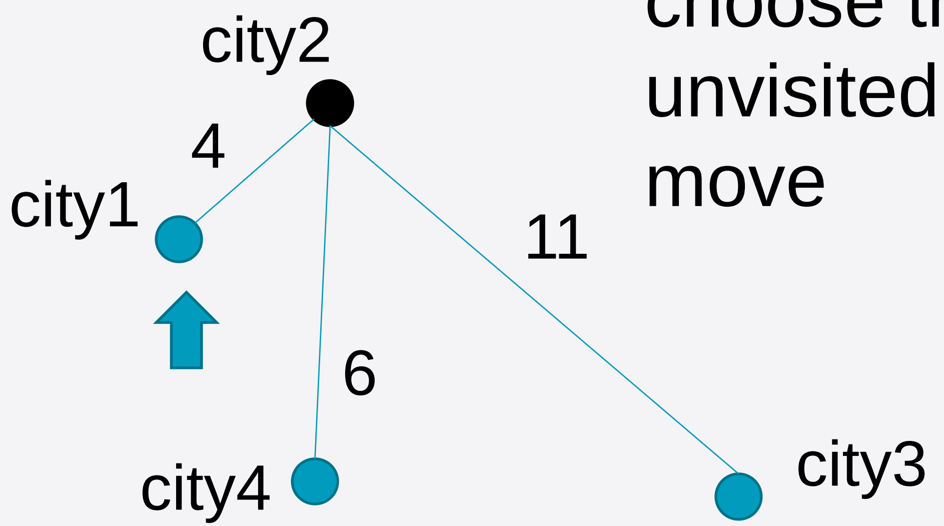
<city2>



Select a starting city randomly

The nearest neighbour (NN) algorithm

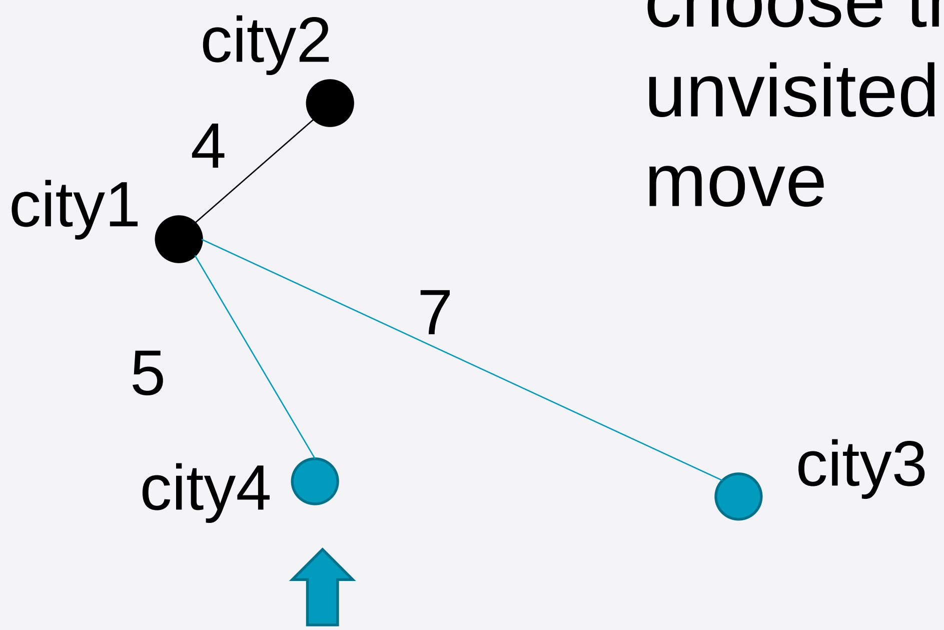
<city2, >



choose the nearest unvisited city as the next move

The nearest neighbour (NN) algorithm

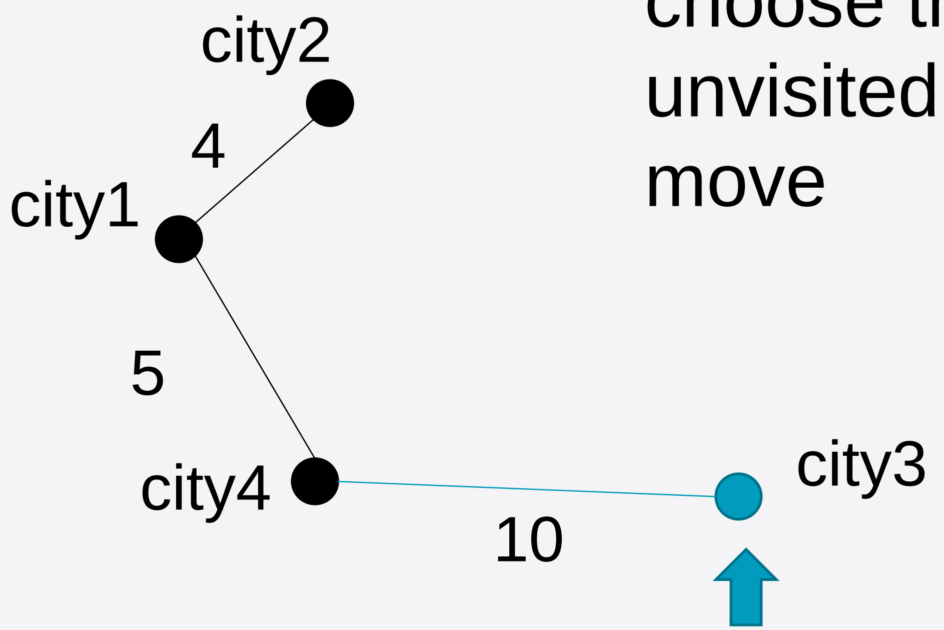
<city2, city1, > : 4



choose the nearest unvisited city as the next move

The nearest neighbour (NN) algorithm

<city2, city1, city4, > : 9

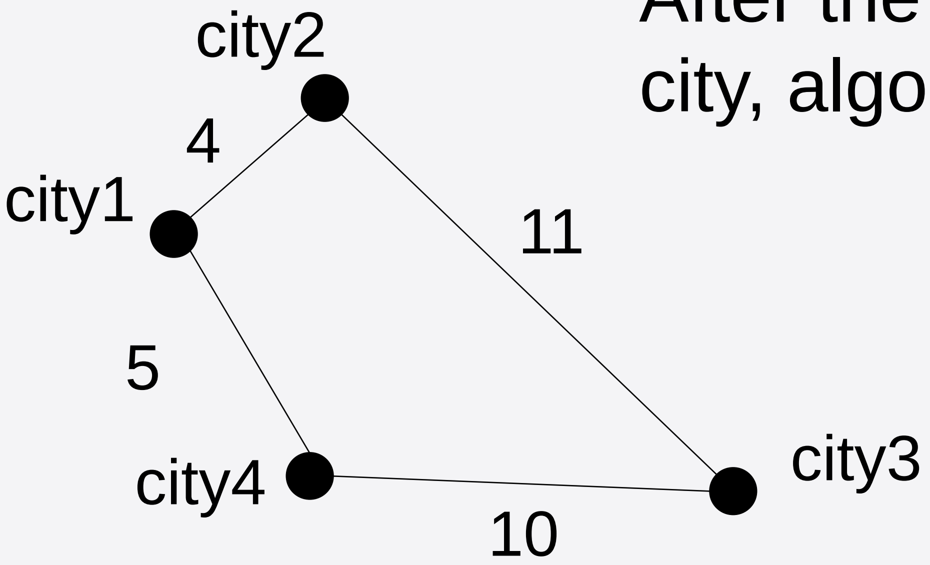


choose the nearest unvisited city as the next move

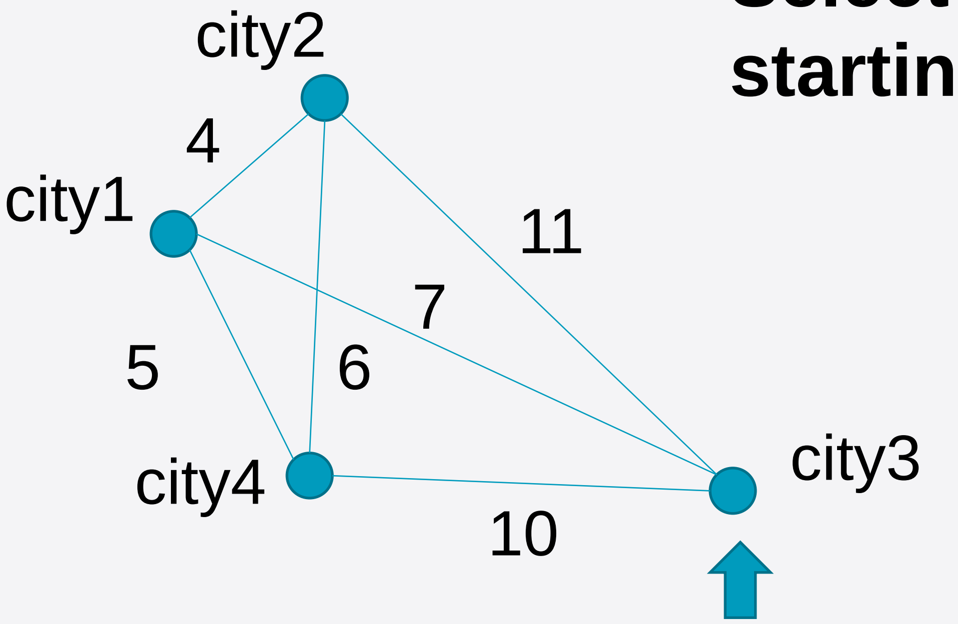
The nearest neighbour (NN) algorithm

<city2, city1, city4, city3> : 30

After the choice of the last city, algorithm terminates



The nearest neighbour (NN) algorithm – Notice



Select a different starting city

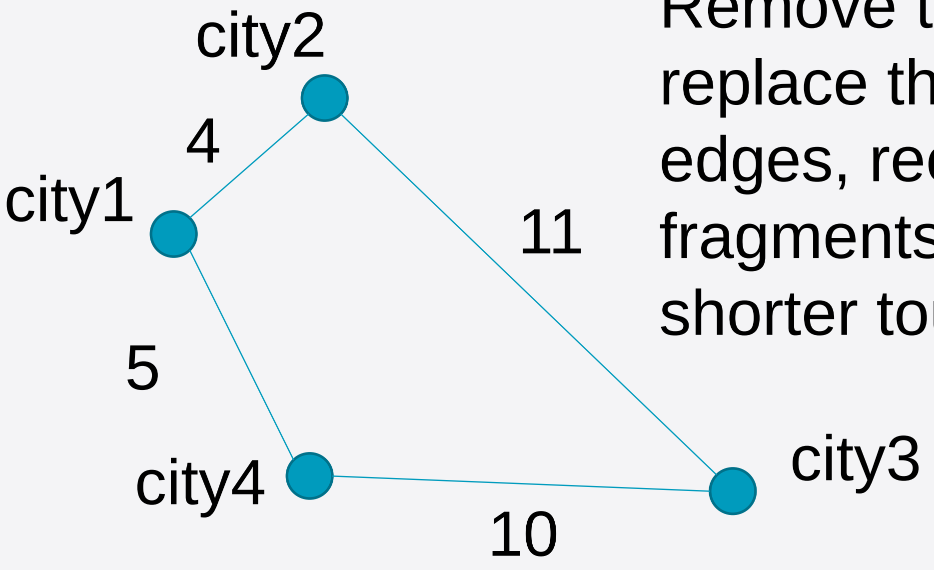
<city3, city1, city2, city4> : 27

A Constructive Stochastic Local Search Algorithm for TSP

- Step 1: Choose a random city
- Step 2: Apply nearest neighbour to construct a complete solution
- Step 3: Compare the new solution to the best found so far and update the best solution as appropriate
- Step 4: Go-to Step 1 and repeat while the maximum number of iterations is not exceeded (parameter)
- Step 5: Return the best solution

Pairwise exchange (2-opt)

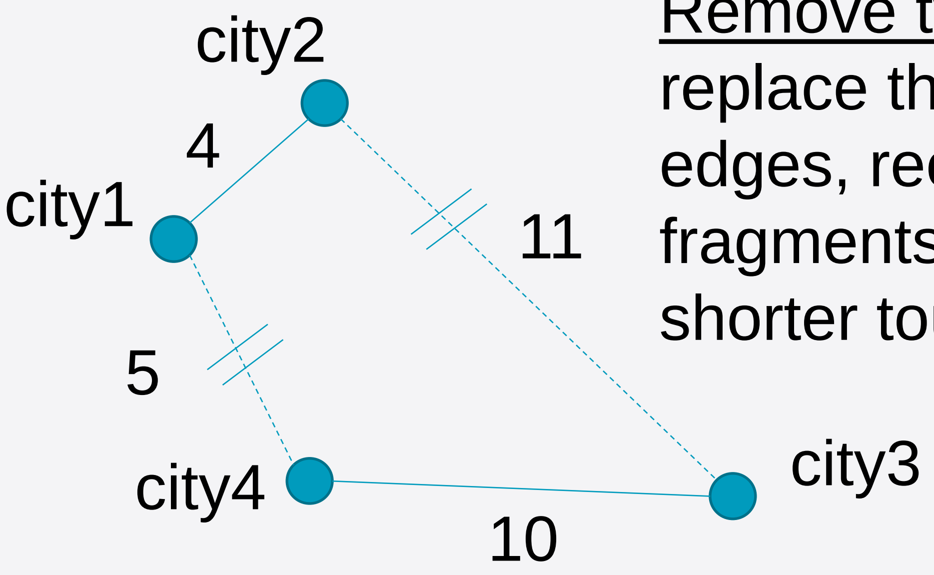
<city2, city1, city4, city3> : 30



Remove two edges and replace them with two different edges, reconnecting the fragments into a new and shorter tour.

Pairwise exchange (2-opt)

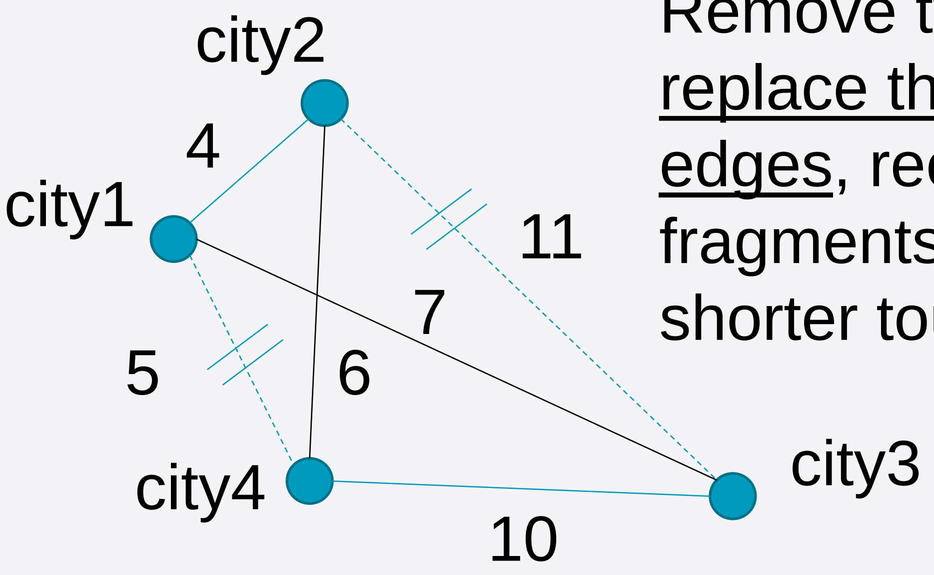
<city2, city1, city4, city3> : 30



Remove two edges and replace them with two different edges, reconnecting the fragments into a new and shorter tour.

Pairwise exchange (2-opt)

<city2, city1, city4, city3> : 30

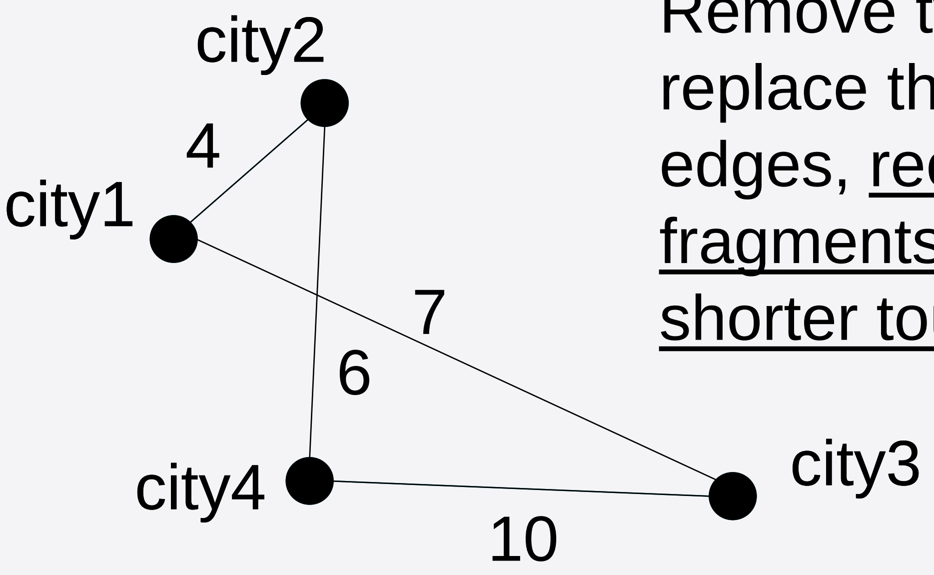


Remove two edges and <u>replace them with two different edges</u>, reconnecting the fragments into a new and shorter tour.

<city1, city2, city4, city3> : 27

Pairwise exchange (2-opt)

<city1, city2, city4, city3> : 27

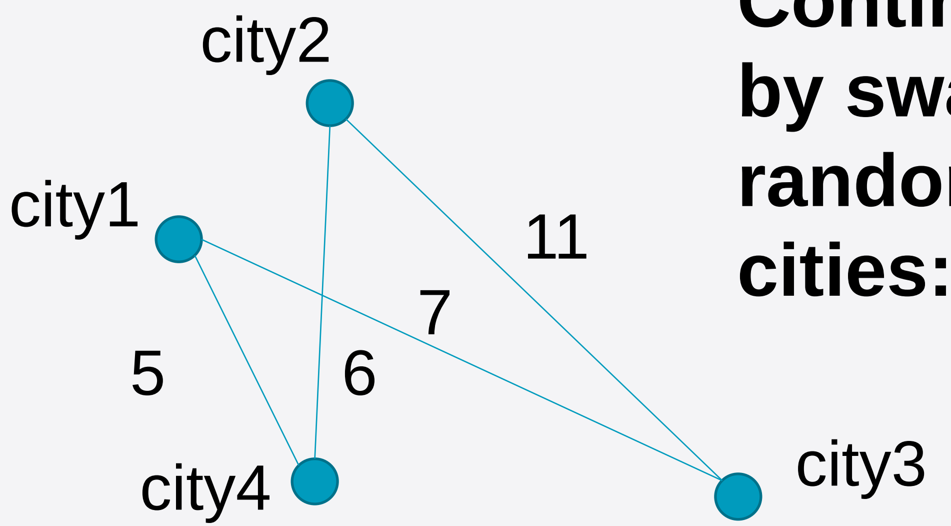


Remove two edges and replace them with two different edges, reconnecting the fragments into a new and shorter tour.

Pairwise exchange (2-opt) – Notice

<city1, city2, city4, city3> : 27

Continue the search by swapping randomly chosen two cities: city4 and city2



<city1, city4, city2, city3> : 29

This solution is worse than the incumbent solution so why not reject.

A Perturbative Stochastic Local Search Algorithm for TSP

- Step 1: Create a random current solution (build a permutation array and shuffle its content)
- Step 2: Apply 2-opt: swap two randomly chosen cities forming a new solution
- Step 3: Compare the new solution to the current solution and if there is improvement make the new solution current solution, otherwise continue
- Step 4: Go-to Step 2 and repeat while the maximum number of iterations is not exceeded (parameter)
- Step 5: Return the current solution

More on Euclidean/Symmetric TSP

- Other Heuristics
  - Christofides' algorithm (1976)
  - Match Twice and Stitch [doi:10.1016/j.orl.2004.04.001]
  - Lin–Kernighan [doi:10.1016/S0377-2217(99)00284-2]
- Exact algorithm – Concorde: http://www.math.uwaterloo.ca/tsp/concorde/index.html
- TSP benchmark data: [University of Waterloo]

Drawbacks of Heuristic Search

- There is no guarantee for the optimality of the obtained solutions.
- Usually can be used only for the specific situation for which they are designed.
- Often, heuristics have some parameters
  - Performance of a heuristic could be sensitive to the setting of those parameters
- May give a poor solution.

Part 3.Pseudo-random numbers

COM2001/2011
Artificial Intelligence Methods

Ender Özcan

Lecture 1

Deterministic vs Stochastic Heuristic Search

- Deterministic heuristic search algorithms provide the same solution when run on the given problem instance regardless of how many times.
- Stochastic algorithms contain a random component and may return a different solution at each time they are run on the same instance
  - Multiple trials/runs should be performed for the experiments/simulations
  - Being able to repeat/replicate those multiple trials/runs in the experiments/simulations is crucial in science, and
  - This also enables average performance comparison of different stochastic heuristic search algorithms applying statistical tests

Pseudo-random numbers

- A long sequence of numbers that is produced using a deterministic process but which appear to be random.
- Note that most computers and programming languages have support to produce pseudo-random numbers, and often with seeding
  - E.g, assume a pseudo-random number generator producing values with lower limit: 0.00, upper limit: 1.00
  - seed(12345): <0.19, 0.03, 0.87, 0.54, …>
  - seed(4927): <0.48, 0.91, 0.02, 0.26, …>

Some problems with pseudo-random numbers

- Shorter than expected periods for some seed states; such seed states may be called 'weak' in this context. E.g.,

  1000th entry

  seed(12345): <0.19, 0.03, …, 0.19, 0.03 …>

- Lack of uniformity of distribution. E.g., 0.17 appears 100 times in 10000 successive numbers while 0.29 appears 5 times more
- Correlation of successive values.
- The distances between where certain values occur are distributed differently from those in a random sequence distribution

Example – Middle Square Method, an early pseudo-random number generator

- Presented by John Von Neumann in 1949
- To generate a sequence of n-digit pseudorandom numbers
- Example: 4-digit case
  - Starts with an initial value (seed) (2156)
  - Takes its square (21562 = 04648336)
  - Middle digits are used as a random number 6483
  - Then this process is repeated
  - (64832 = 42029289)
- Problem: all sequences eventually repeat themselves

Notice the difference – Exercise 1

```
for (int trial=0; trial<5; trial++)
// Assume this represents 5 trials/runs of an experiment
{
    long seed = 123456789;
    Random generator = new Random(seed);
    // or Random generator = new Random(); generator.setSeed(seed);
    double num = generator.nextDouble() ; // rand. value between 0.0 and 1.0
    System.out.print(num);
    System.out.print(' ');
}
```

Imagine you repeat this experiment, running the code at two different times.

- Always the same value for “num” is printed out for each trial
- Wrong experimentation as a different behaviour is expected at each trial (iteration)

Notice the difference – Exercise 2

```
for (int trial=0; trial<5; trial++)
// Assume this represents 5 trials/runs of an experiment
{
    long seed = System.currentTimeMillis();
    Random generator = new Random(seed);
    // or Random generator = new Random(); generator.setSeed(seed);
    double num = generator.nextDouble() ; // rand. value between 0.0 and 1.0
    System.out.print(num);
    System.out.print(' ');
}
```

Imagine you repeat this experiment, running the code at two different times.

- The value for “num” changes at each trial/run (iteration).
- Experiments cannot be replicated: different results (print-outs) at each time

Notice the difference – Exercise 3

```
// Assume this represents an experiment
long seed = 123456789;
Random generator = new Random(seed);
// or Random generator = new Random(); generator.setSeed(seed);
for (int trial=0; trial<5; trial++) // 5 trials/runs
{
   double num = generator.nextDouble() ; // rand. value between 0.0 and 1.0
    System.out.print(num);
    System.out.print(' ');
}
```

Imagine you repeat this experiment, running the code at two different times.

- The value for “num” changes at each trial/run (iteration).
- Experiment can be replicated: same results (print-outs) at each time

Notice the difference – Exercise 4

```
// Assume this represents an experiment
long[] seed = {123456, 789000, 323241, 5523525, 2432342};
Random generator = new Random(seed);
// or Random generator = new Random(); generator.setSeed(seed);
for (int trial=0; trial<5; trial++) // 5 trials/runs
{
   double num = generator.nextDouble(seed[i]); // rand. value in [0.0,1.0)
    System.out.print(num);
    System.out.print(' ');
}
```

Imagine you repeat this experiment, running the code at two different times.

- The value for “num” changes at each trial/run (iteration).
- Experiment can be replicated: same results (print-outs) at each time

University of Nottingham
UK | CHINA | MALAYSIA

Q&A

ender.ozcan@nottingham.ac.uk

www.cs.nott.ac.uk/~pszeo/