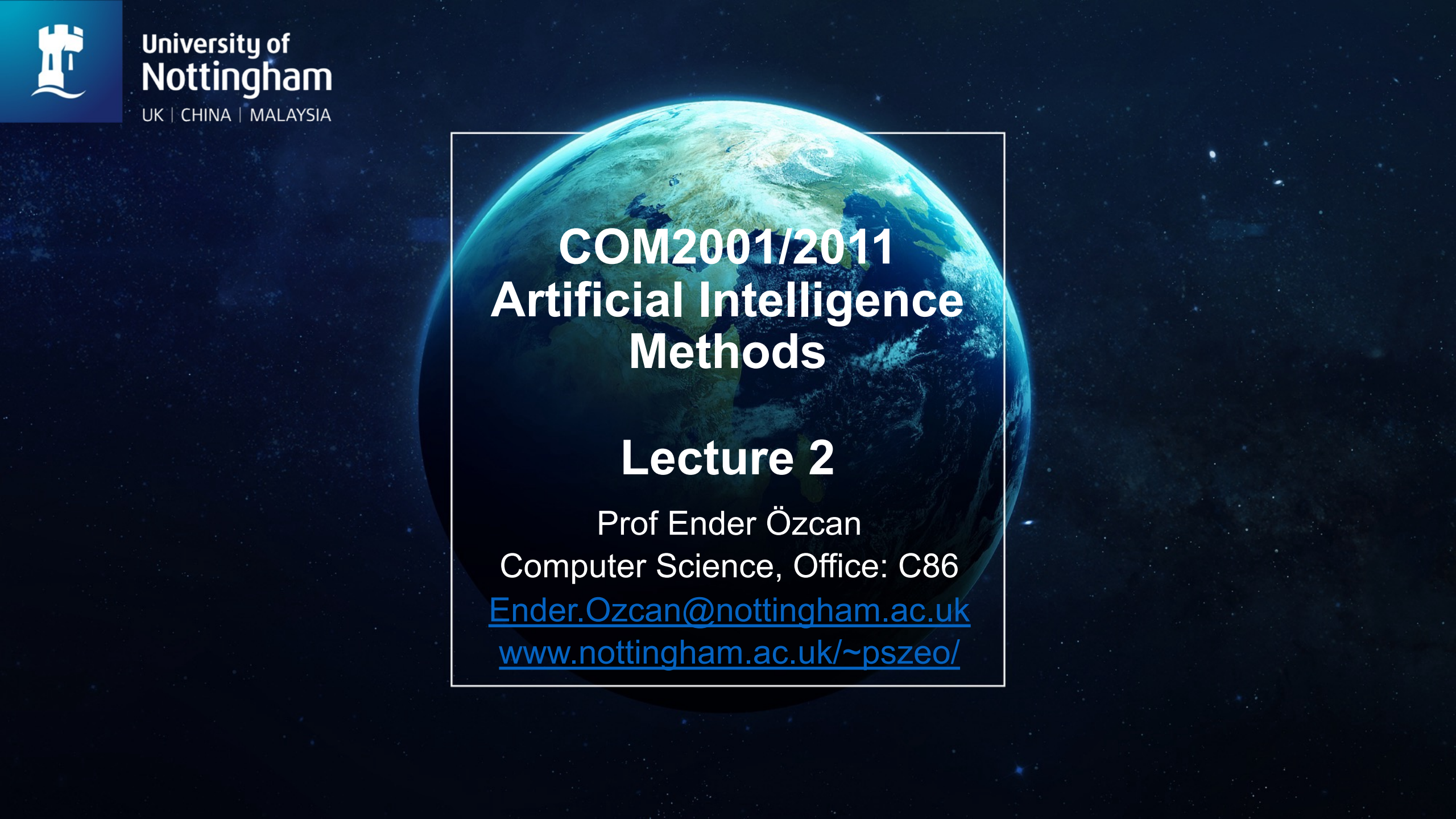


University of Nottingham
UK | CHINA | MALAYSIA

COM2001/2011 Artificial Intelligence Methods

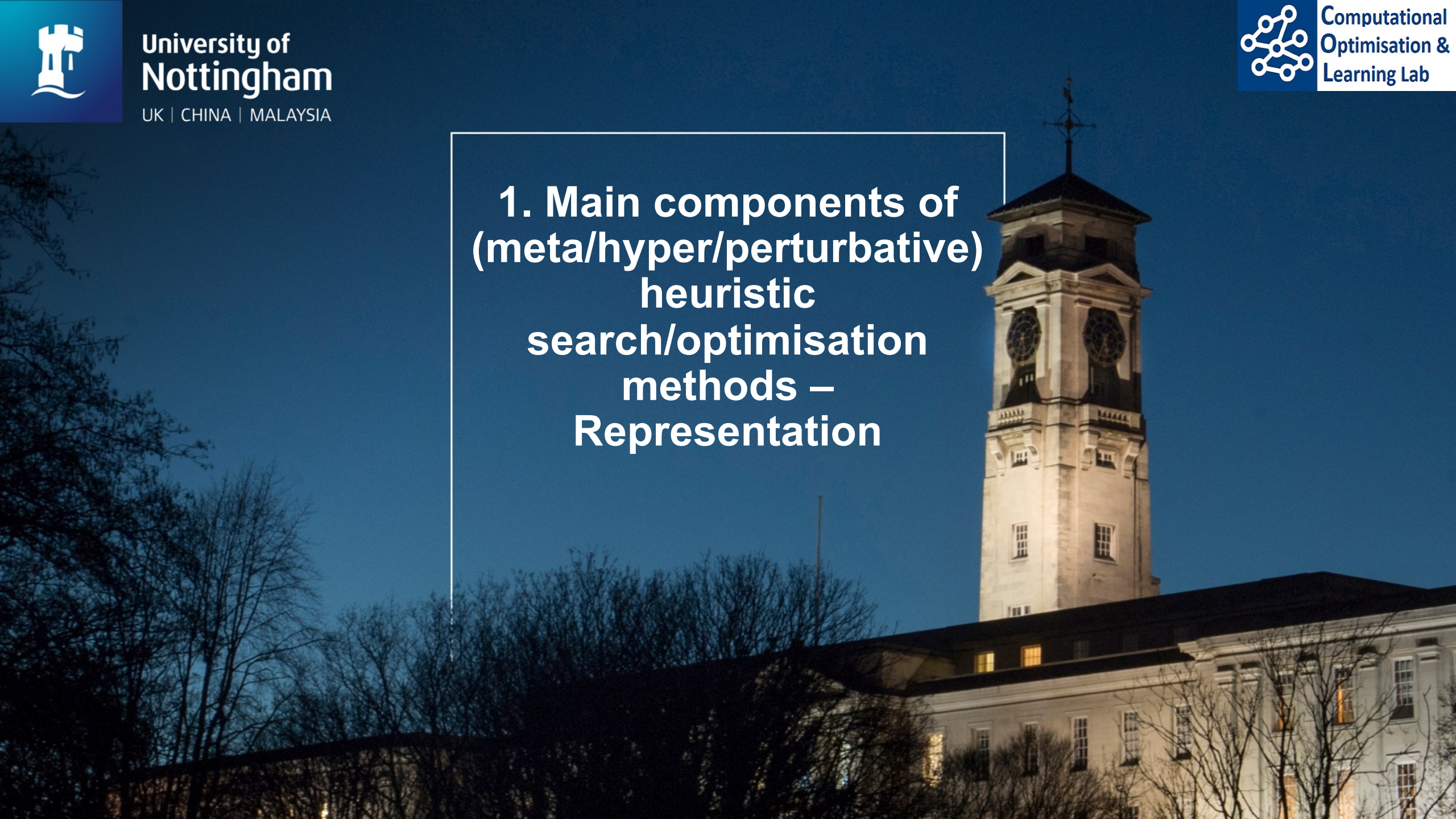
Lecture 2

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

Components of (Meta/Hyper-/Perturbative) Heuristic Search Methods and Hill Climbing

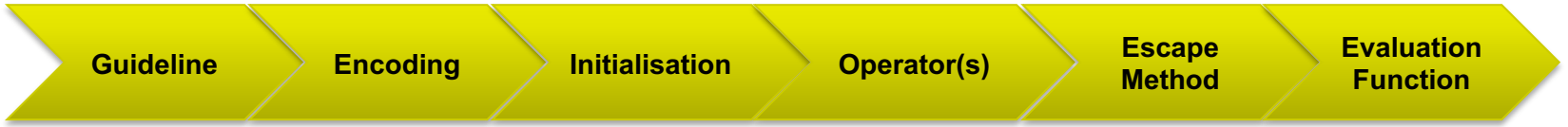
Content

1. Main components of (meta/hyper/perturbative) heuristic search/optimisation methods – Representation
2. Neighbourhoods
3. Evaluation Function
4. Hill climbing methods
5. *Reading*: Performance Analysis of Stochastic Local Search Methods – Preliminaries


University of
Nottingham
UK | CHINA | MALAYSIA
Computational
Optimisation &
Learning Lab
1. Main components of (meta/hyper/perturbative) heuristic search/optimisation methods – Representation

Main Components of (Meta/Hyper-) Heuristic Search/Optimisation Methods

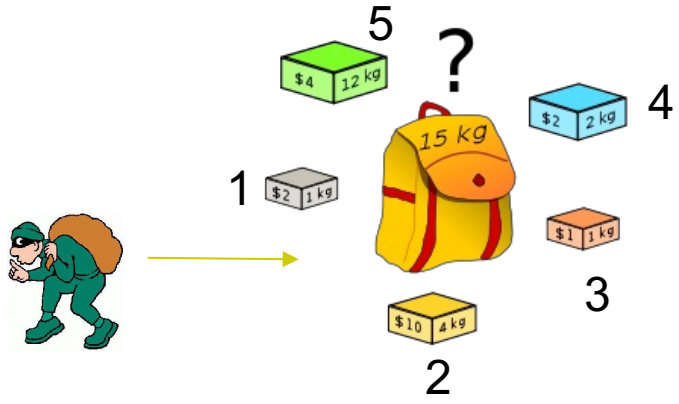
- Representation (encoding) of candidate solutions
- Neighbourhood relation (move operators)
- Evaluation function (objective function)
- *Initialisation* (e.g., random)
- *Search process (guideline)*
- *Mechanism for escaping from local optima*



Representation (Encoding of a Solution) – Characteristics

- **Completeness**: all solutions associated with the problem must be represented.
- **Connexity**: a search path must exist between any two solutions of the search space. Any solution of the search space, especially the global optimum solution, can be attained.
- **Efficiency**: The representation must be easy/fast to manipulate by the search operators.

Representation



- **Binary encoding** is the most common
  - 10110010110010…1011
- E.g.: 0/1 Knapsack problem

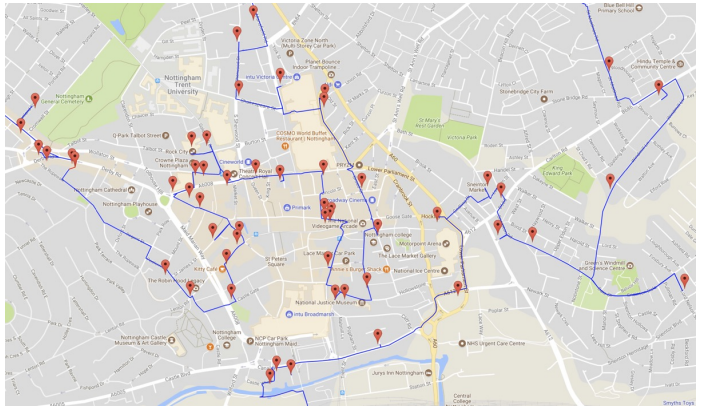
  Fill the knapsack with as much value in goods as possible – which items to take?

  10011 ($8), 11110 ($15)
- Given a binary string of length *N* (representing *N* items), search space size is 2^N

Representation (cont.)

- **Permutation encoding**
  - A candidate solution for 100 city TSP instance would be:
    21 5 38 2 … 100 64 76 9 18 3 (permutation of 1..100)
- E.g.: Travelling salesman problem, sequencing problems
- Given *N* entities (e.g., cities, pubs), search space size is
  - *N*!

A shortest-possible walking tour through the pubs of the UK (Nottingham):

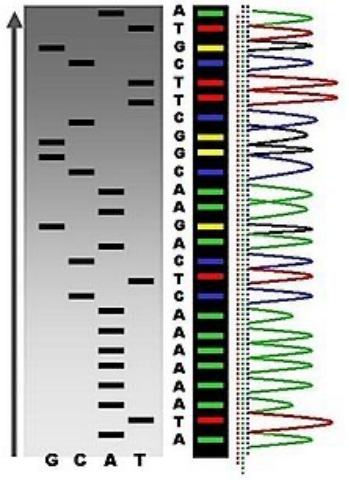


Representation (cont.)

- **Integer encoding**
  - 1 3 4 5 5 5 4 1 1 … 2 2 1
- E.g.: Personnel rostering problem, timetabling problem, layout/structure optimisation
  - Given an unlimited number of 5 different composite materials, which material would you use for each of the 15-layer composite structure maximising the sound absorption?
  - For a general problem with *M* composite materials to form an *N*-layer composite structure, search space size is M^N

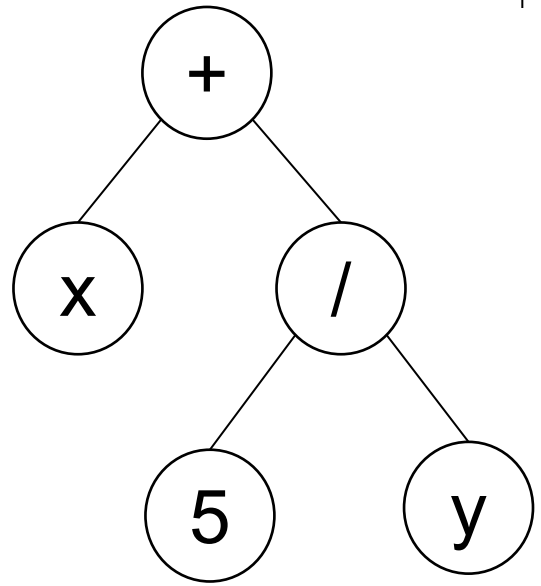
Representation (cont.)

- **Value Encoding**
  - E.g.: Parameter/continuous optimisation
    - 1.2324 5.3243 0.4556 2.3293 2.4545
  - E.g.: DNA sequencing is the process of determining the precise order of nucleotides within a DNA molecule. (Adenine, Guanine, Cytosine, and Thymine)
    - ATGCTTCGGCAAGACTCAAAAAATA
  - E.g.: Planning
    - <(back), (back), (right), (forward), (left)>



Representation (cont.)

- **Nonlinear Encoding**
  - Tree Encoding – Genetic Programming



  - E.g.: Computers generating heuristics or heuristic components

(+ x (/ 5 y))

Boolean Satisfiability Problem

- The first problem proven to be NP-Complete
- Given the formula: $\varphi = (\neg x_0 \vee \neg x_1) \wedge (x_1 \vee \neg x_2) \wedge (x_0) \wedge (x_2)$
- Boolean Satisfiability (SAT) Problem
  - Is there an assignment of true or false values to variables such that φ evaluates to true?

Maximum Satisfiability Problem – Real-world Applications

probabilistic inference	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
fault localization	[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
restoring CSP consistency	[Lynce and Marques-Silva, 2011]
reasoning over bionetworks	[Guerra and Lynce, 2012]
MCS enumeration	[Morgado, Liffiton, and Marques-Silva, 2012]
heuristics for cost-optimal planning	[Zhang and Bacchus, 2012]
optimal covering arrays	[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	[Wallner, Niskanen, and Järvisalo, 2016]
...	

[MAX-SAT Tutorial]

- Planning,
- Scheduling,
- Configuration problems,
- AI and data analysis problems,
- Combinatorial problems,
- Verification and security,
- Bioinformatics

…

[Chapter on MAX-SAT]

Exercise – Maximum Satisfiability Problem

- MAX-SAT: Given a Boolean formula in conjunctive normal form – a conjunction ($\wedge$) of clauses, where a clause is a disjunction ($\vee$) of literals (e.g., $x_0, x_1, \ldots, x_n$),

  find the maximum number of clauses that can be satisfied by some truth assignment

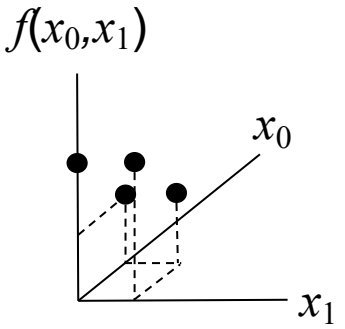
  - E.g., $(\neg x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$ => Problem instance

    $(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$ => Another problem instance

  How would you represent a candidate solution (suggest an encoding)?

MAX-SAT Problem – Candidate solution representation

$x_0\ x_1$		number of clauses satisfied
0 0:	$(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$ is true	2
0 1:	$(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$ is true	2
1 0:	$(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$ is false	1
1 1:	$(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$ is false	1



maximum number of clauses satisfied is 2

MAX-SAT Problem – Search Space Size

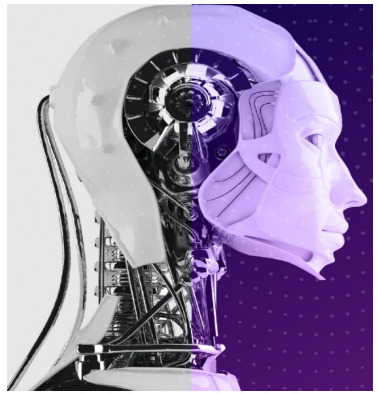
- Another problem instance

$$(x_0 \vee x_1) \wedge (\neg x_0 \vee x_1) \wedge (x_0 \vee \neg x_1) \wedge (\neg x_0 \vee \neg x_1)$$

not satisfiable, however, 3 clauses can be satisfied by any assignment (multiple solutions to this MAX-SAT instance can be found) – contradiction

- Given *n* Boolean literals/variables, what is the number of possible configurations (search space size)?
  - 2^n

2. Neighbourhoods



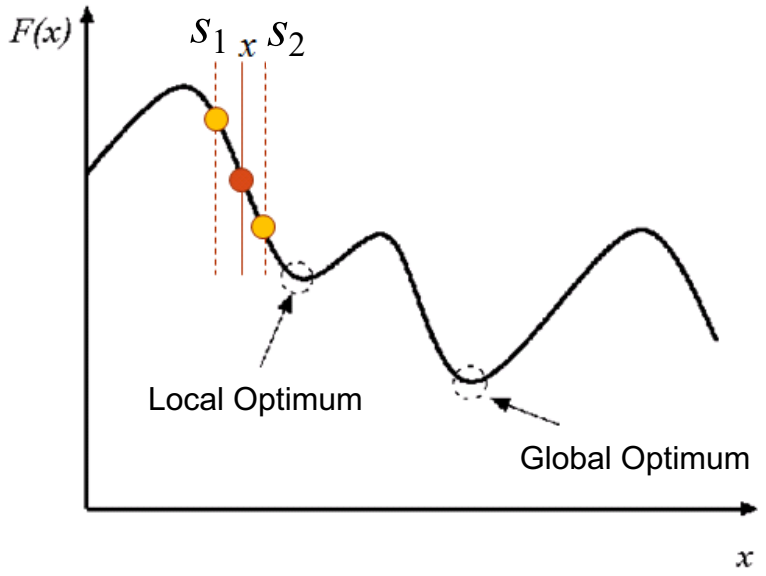
Computational Optimisation & Learning Lab

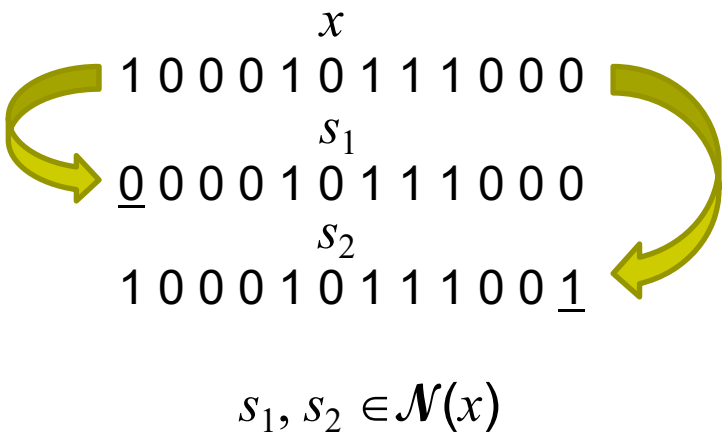
The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Definition

- A *neighbourhood* of a solution *x is a set of solutions* that can be reached from *x applying a (move) operator/heuristic*





$s_1, s_2 \in \mathcal{N}(x)$

Example Neighbourhood for Binary Representation

- Bit-flip operator: flips a bit in a given solution
- Hamming Distance between two bit strings (vectors) of equal length is the number of positions at which the corresponding symbols differ. E.g., HD(011,010)=1, HD(0101,0010)=3
- If the binary string is of size *n*, then the neighbourhood size is *n*.
- Example: **1** 0 1 0 0 0 1 1 → **0** 0 1 0 0 0 1 1

  Neighbourhood size: 8, Hamming distance: 1

Example Neighbourhood for Integer/Value Representation

- Random neighbourhood/move/perturbation/ assignment operator: a discrete value is replaced by any other character of the alphabet.
- If the solution is of size *n* and alphabet is of size *k*, then the neighbourhood size is $(k\text{-}1)n$.
- Example: **5** 7 9 6 4 4 8 3 → **0** 7 9 6 4 4 8 3

Neighbourhood size: (10-1)8=72 (alphabet:0..9)

A D J E I F → **M** D J E I F

Neighbourhood size: (26-1)6=150 (alphabet:A..Z)

Example Neighbourhood for Permutation Representation I

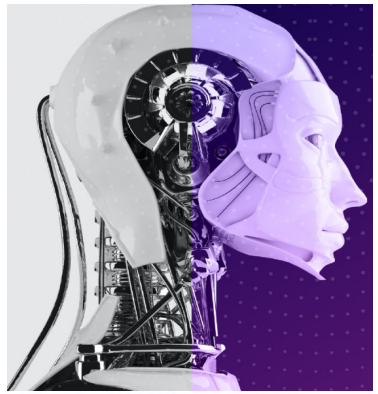
- Adjacent pairwise interchange: swap adjacent entries in the permutation
  - If permutation is of size *n*, then the neighbourhood size is *n*-1
  - Example: **5 1** 4 3 2 → **1 5** 4 3 2
- Insertion operator: take an entry in the permutation and insert it in another position
  - Neighbourhood size: *n*(*n*-1)
  - Example: **5** 1 4 3 2 → 1 4 **5** 3 2

Example Neighbourhood for Permutation Representation II

- Exchange operator: arbitrarily selected two entries are swapped
  - Example: **5** 4 3 **1** 2 → **1** 4 3 **5** 2

- Inversion operator: select two arbitrary entries and invert the sequence in between them
  - Example: 1 **4 5 3** 2 → 1 **3 5 4** 2

3. Evaluation/Objective Function



Computational Optimisation & Learning Lab

The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Evaluation Function

- Also referred to as objective, cost, fitness, penalty, etc.
  - Indicates the quality of a given solution, distinguishing between better and worse solutions
- Serves as a major link between the algorithm and the problem being solved
  - provides an important feedback for the search process
- Many types: (non)separable, uni/multi-modal, single/multi-objective, etc.

Evaluation Function (cont.)

- Evaluation functions could be computationally expensive
- Exact vs. approximate
  - Common approaches to constructing approximate/surrogate models: polynomials, regression, SVMs, etc.
  - Constructing a globally valid approximate model remains difficult, and so beneficial to selectively use the original evaluation function together with the approximate model

MAX-SAT Problem – Evaluation function

- Maximising:

  f_1= C; (count the number of satisfied clauses)

- Minimising:

  f_2 = (No. of clauses – C); No. of unsatisfied clauses

  f_3 =f_2 / (No. of clauses);

- Example: $(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$

x_0	x_1		
1	1	f_1=1, f_2=1, f_3=0.5	change x_0 to 0
0	1	f_1=2, f_2=0, f_3=0.0	

TSP – Evaluation function

- TSP requires a search for a permutation

  $\pi : \{0,\ldots, N\text{-}1\} \rightarrow \{0,\ldots, N\text{-}1\},$

  using a cost matrix $C=[c_{ij}]$, where c_{ij} denotes the cost (assumed to be known) of the travel from city i to j, that minimizes the *path length*

$$f(\pi, C) = \sum_{i=0}^{N-1} c_{\pi(i),\pi((i+1) \bmod N)},$$

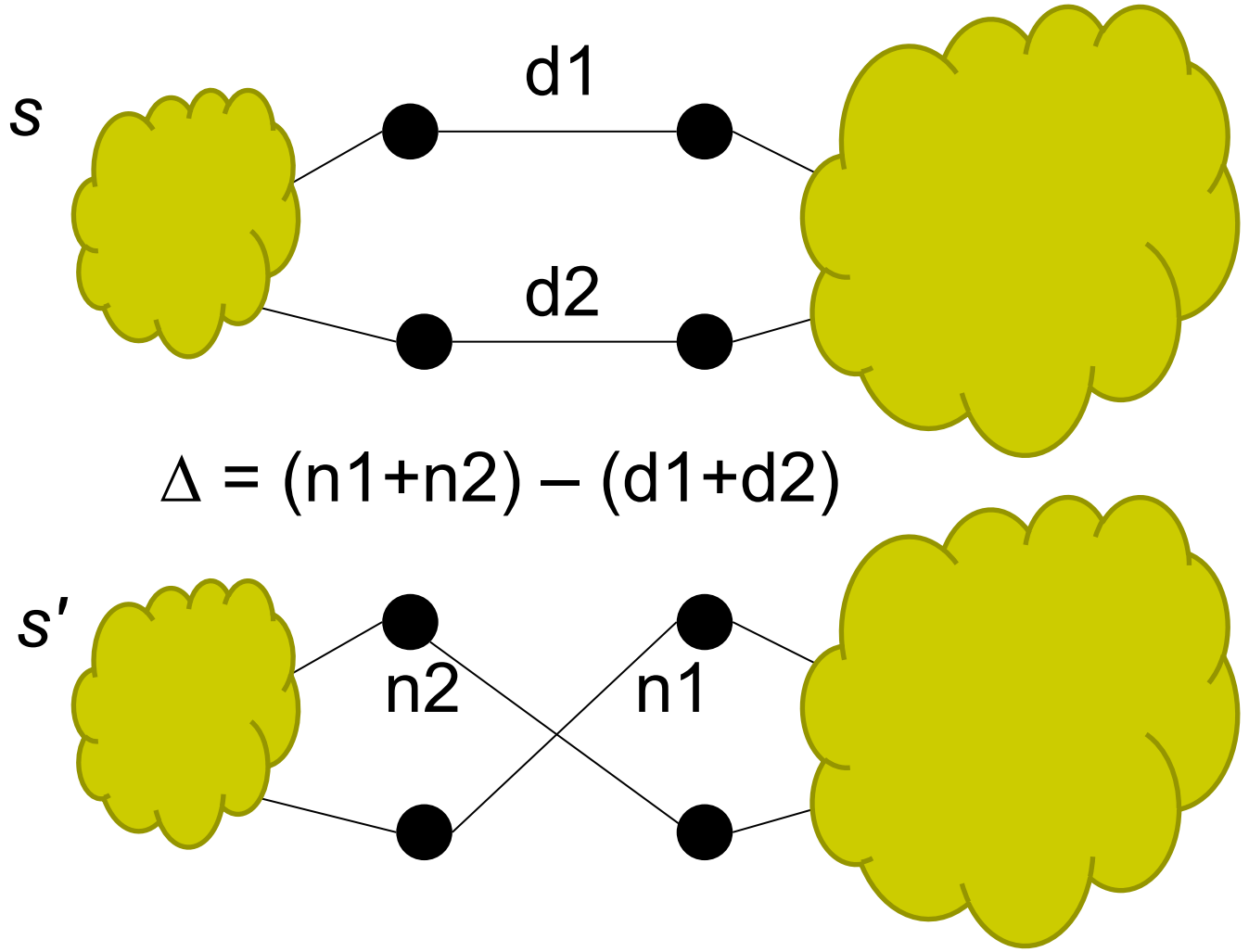
  where $\pi(i)$ denotes the city at i-th location in the tour and

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Evaluation Function – Delta (Incremental) Evaluation

- **Key idea:** calculate *effects of differences* between current search position *s* and a neighbour *s′* on the evaluation function value.
- Evaluation function values often consist of independent contributions of solution components; hence, $f(s')$ can be efficiently calculated from $f(s)$ by differences between *s* and *s'* in terms of solution components.
- Crucial for efficient implementation of heuristics/metaheuristics/ hyper-heuristics

Example: Delta Evaluation for TSP



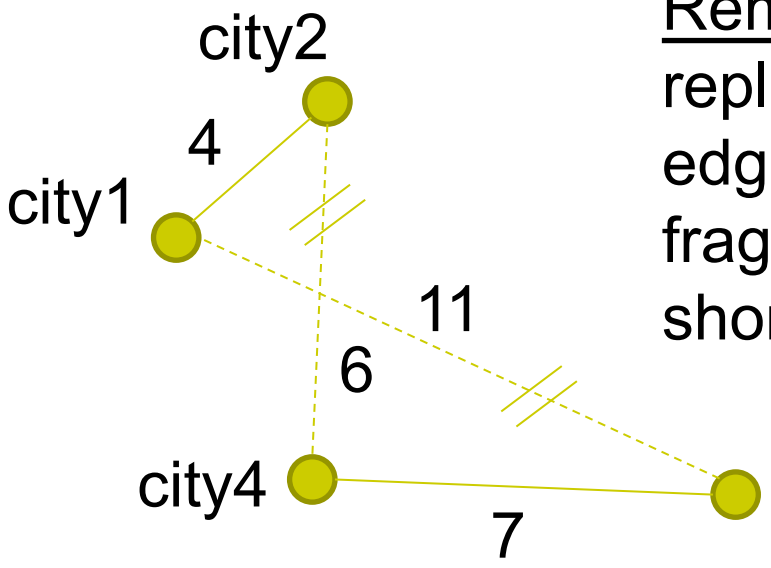
$f(s) = \Sigma$

$f(s') = \Sigma + \Delta$

Example: Delta Evaluation for TSP II

$f(s) = \Sigma$

<city2, city1, city3, city4> : 28

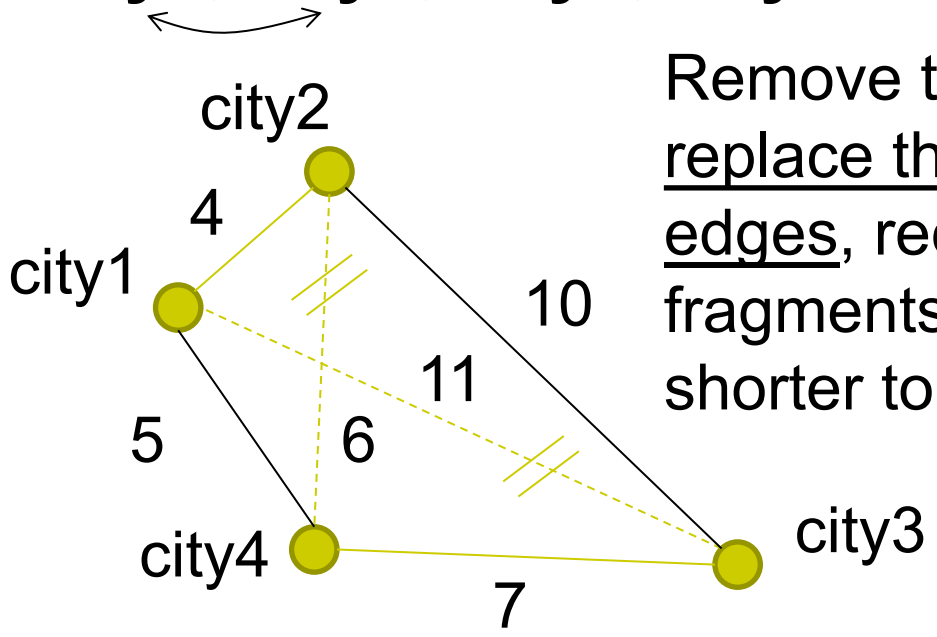


Remove two edges and replace them with two different edges, reconnecting the fragments into a new and shorter tour.

Example: Delta Evaluation for TSP III

$f(s') = \Sigma + \Delta$

<city1, city2, city3, city4> : 26 (28+(-2))



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

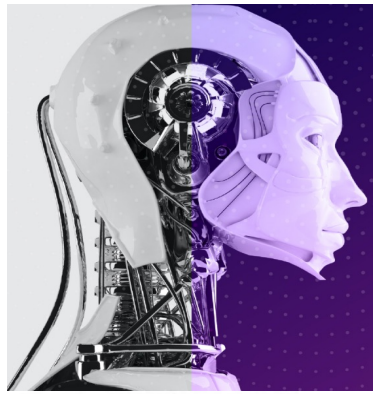
$\Delta = (n1+n2) - (d1+d2)$

$\Delta = (5+10) - (11+6) = -2$

Summary

- Choosing an appropriate encoding to represent a candidate solution is crucial in heuristic optimisation
- Initialisation could influence the performance of an optimisation algorithm.
- Evaluation function guides the search process and fast evaluation is important

4. Hill Climbing Algorithms



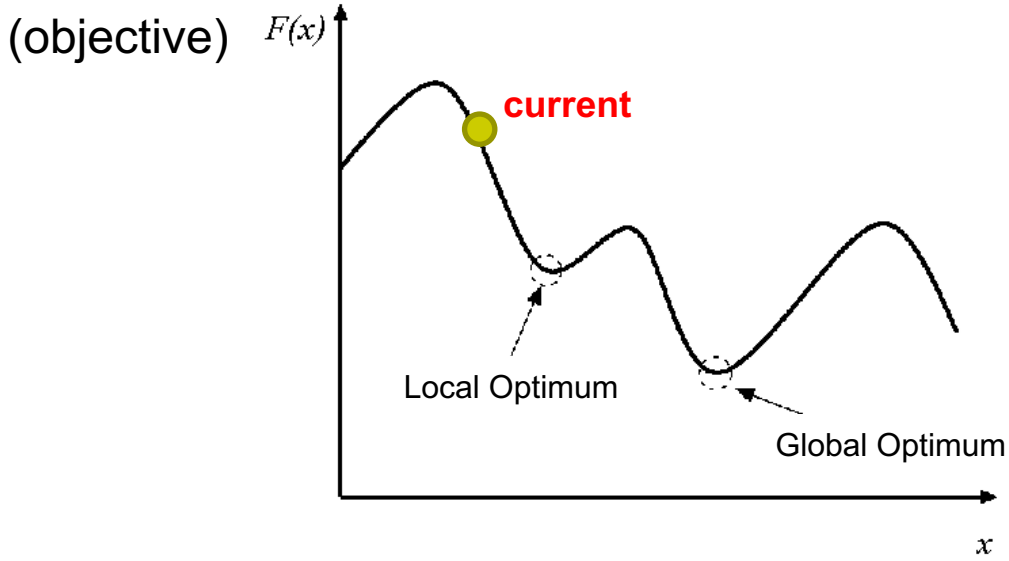
Computational Optimisation & Learning Lab

The University of Nottingham

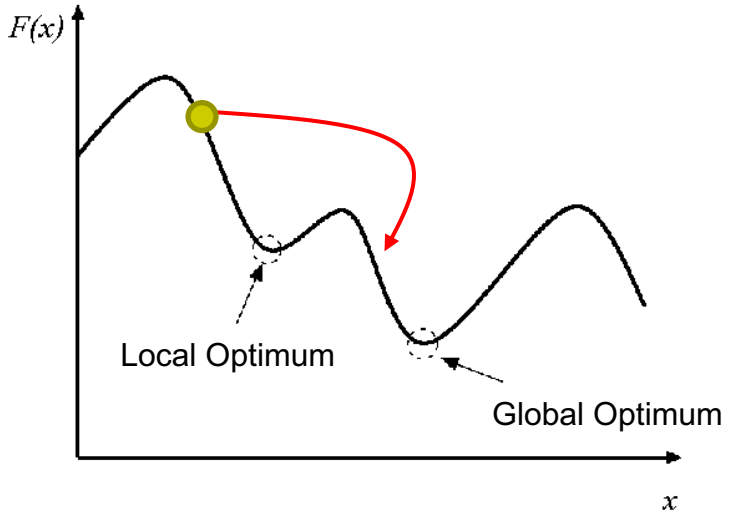
UNITED KINGDOM · CHINA · MALAYSIA

Search Paradigms – Perturbative Heuristics/Operators

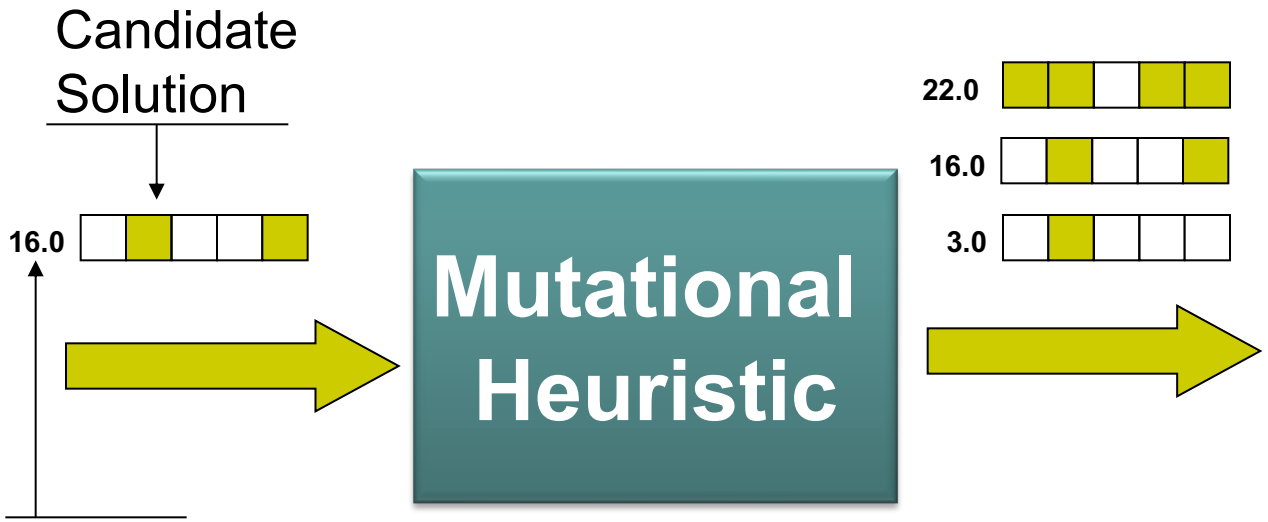
- Mutational (diversification/exploration) vs.
- Hill-climbing (intensification/exploitation)



Mutational Heuristic/Operator



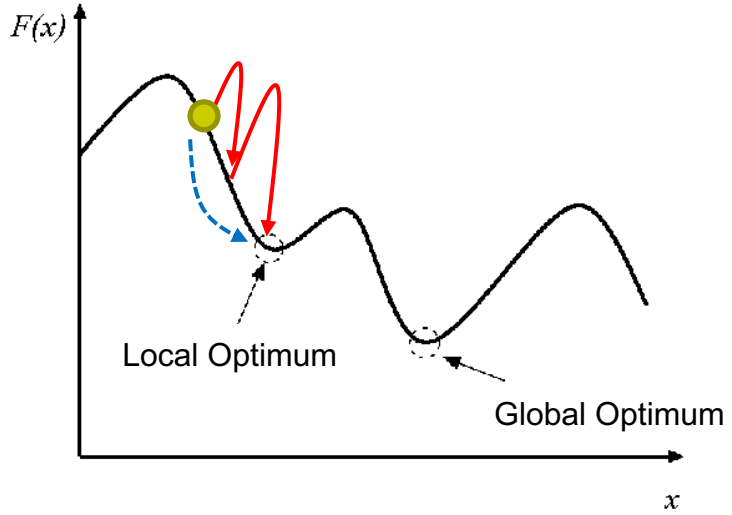
Processes a given candidate solution and generates a solution which is not guaranteed to be better than the input



Minimising Fitness /Cost/Penalty/…

e.g., total number of constraint violations or a weighted sum of violations

Hill Climbing Heuristic/Operator



Processes a given candidate solution and generates a better or equal quality solution

Minimising Fitness /Cost/Penalty/…

e.g., total number of constraint violations or a weighted sum of violations

Hill Climbing Algorithm – Minimisation/*Maximisation* Problem

A **local search algorithm** which constantly ***moves*** in the direction of decreasing/*increasing* level/objective value for a minimisation/*maximisation* problem to find the nadir (lowest)/*peak* (*highest*) point of the landscape or best/near optimal solution to the problem.

- The hill climbing algorithm halts when it detects a nadir/peak value where no neighbour has a lower/*higher* value while solving a minimisation/*maximisation* problem.

Pseudocode of a Generic Hill Climbing Algorithm

1. Pick an initial starting point (as current point/state/candidate solution) in the search space
2. Repeat
    1. Consider the *neighbor*(s) of the current point as new point(s)
    2. Compare new point(s) in the *neighborhood* of the current point with the current point using an evaluation function and choose one with the best quality (among them) and move to that point
3. Until termination criteria satisfied (usually,there is no more improvement or when a predefined maximum number of iterations is reached)
4. Return the current point as the best solution found so far

More on Hill Climbing

- Initial starting point(s) may be chosen,
  - randomly
  - use a constructive heuristic/operator(s)
  - according to some regular pattern
  - based on other information (e.g. results of a prior search)
- Variations of hill-climbing algorithms differ in the way a new solution/state/string is selected for comparisons with the current (incumbent) solution/state/string

Simple Hill Climbing Heuristics

- Simple Hill Climbing examining neighbours:
  - Best improvement (steepest descent/ascent)
  - First improvement (next descent/ascent)
- Stochastic Hill Climbing (randomly choose neighbours)
  - Random selection/random mutation hill climbing
- Random-restart (shotgun) hill climbing is built on top of hill climbing and operates by changing the starting solution for the hill climbing, randomly and returning the best

Best Improvement (steepest descent/ ascent) Hill-climbing

```
bestEval = evaluate(currentSolution); improved = false;
for(j=0;(j<length[currentSolution]);j++){ // left to right scan, single pass
   bitFlip(currentSolution, j); // flips jth bit of current solution
   tmpEval = evaluate(currentSolution);
   if (tmpEval < bestEval) {  // strict improvement
            // remember the bit which yields the best value after evaluation
            bestIndex= j;
            bestEval = tmpEval;
            improved = true;
   } // end if
   bitFlip(currentSolution, j); // go back to the initial current solution
} // end for
if (improved)  bitFlip(currentSolution, bestIndex);
```

Exercise – Applying Best Improvement to a MAX-SAT Problem Instance

- $(\neg x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

$x_0\ x_1\ x_2\ x_3 : f_2$

⇨ 0 0 0 0:1, minimising f_2 (No. of unsatisfied clauses)

j

iteration#0: <u>1</u> 0 0 0:1

iteration#1: 0 <u>1</u> 0 0:1

iteration#2: 0 0 <u>1</u> 0:**0** ✓

iteration#3: 0 0 0 <u>1</u>:0

```
bestEval = evaluate(currentSolution); improved = false;
for(j=0;(j<length[currentSolution]);j++){ // left to right scan
  bitFlip(currentSolution, j); // flips jth bit of current solution
  tmpEval = evaluate(currentSolution);
  if (tmpEval < bestEval) {  // strict improvement
        // remember the bit which yields the best value after evaluation
        bestIndex= j;
        bestEval = tmpEval;
        improved = true;
  } // end if
  bitFlip(currentSolution, j); // go back to the initial current solution
} // end for
if (improved)  bitFlip(currentSolution, bestIndex);
```

First Improvement (next descent/ascent) Hill-climbing

```
bestEval = evaluate(currentSolution);
for(j=0;(j<length[currentSolution]);j++){// single pass
   bitFlip(currentSolution, j); // flips jth bit of solution producing s' from s
   tmpEval = evaluate(currentSolution);
   if (tmpEval < bestEval)        // in case there is improvement
         bestEval = tmpEval;  // accept the bit flip
   else                                  // if no improvement, reject the bit flip
         bitFlip(currentSolution, j); // go back to s from s'
} // end for
```

Exercise – Applying First Improvement to a MAX-SAT Problem Instance

- $(\neg x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

$x_0\ x_1\ x_2\ x_3{:}f_2$

0 0 0 0:1, minimising f_2

1 0 0 0:1

0 0 0 0:1

0 1 0 0:1

0 0 0 0:1

✓0 0 1 0:0

0 0 1 1:0

```
bestEval = evaluate(currentSolution);
for(j=0;(j<length[currentSolution]);j++){// single pass
  bitFlip(currentSolution, j); // flips jth bit of solution producing s' from s
  tmpEval = evaluate(currentSolution);
  if (tmpEval < bestEval)       // in case there is improvement
        bestEval = tmpEval;  // accept the bit flip
  else                                  // if no improvement, reject the bit flip
        bitFlip(currentSolution, j); // go back to s from s'
} // end for
```

Davis's (Bit) Hill-climbing

```
bestEval = evaluate(currentSolution);
// creates a random permutation of integers (index values) from 0..length[currentSolution]-1
// in array perm, e.g., int perm[] ={0,1,2} (of length/size 3) becomes {1, 2, 0}
perm=createRandomPermutation(length[currentSolution]);
for(j=0;(j<length[currentSolution]);j++){// single pass
   // flips the bit pointed by perm[j] of solution producing s' from s
   bitFlip(currentSolution, perm[j]);
   tmpEval = evaluate(currentSolution);
   if (tmpEval < bestEval)       // in case there is improvement
            bestEval = tmpEval;  // accept the bit flip
   else                                  // if no improvement, reject the bit flip
            bitFlip(currentSolution, perm[j]); // go back to s from s'
} // end for
```

Exercise – Applying Davis's Bit Hill Climbing to a MAX-SAT Problem Instance

- $(\neg x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

perm = [1, 2, 0, 3]

x_0 x_1 x_2 x_3:f_2

0 0 0 0:1, minimising f_2

0 <u>1</u> 0 0:1 [**1**, 2, 0, 3]

0 0 0 0:1

✓ 0 0 <u>1</u> 0:0 [1, **2**, 0, 3]

could stop here

<u>1</u> 0 1 0:0 [1, 2, **0**, 3]

0 0 1 0:0

0 0 1 <u>1</u>:0 [1, 2, 0, **3**]

Random Mutation Hill-climbing

```
bestEval = evaluate(currentSolution);
for(k=0;(k<noOfSteps);k++){ // single pass if k=length[solution]
   j=random(0, length[solution]);
   bitFlip(currentSolution, j); // flips jth bit of solution producing s' from s
   tmpEval = evaluate(currentSolution);
   if (tmpEval < bestEval)        // in case there is improvement
         bestEval = tmpEval;  // accept the bit flip
   else                                    // if no improvement, reject the bit flip
         bitFlip(currentSolution, j); // go back to s from s'
} // end for
```

Exercise – Applying Random Mutation Hill Climbing to a MAX-SAT Problem Instance

- $(\neg x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

random= [2, 0, 2, 3]

x_0 x_1 x_2 x_3:f_2

0 0 0 0:1, minimising f_2

0 0 <u>1</u> 0:0 [**2**, 0, 2, 3] ← could stop here

<u>1</u> 0 1 0:0 [2, **0**, 2, 3]

0 0 1 0:0

0 0 <u>0</u> 0:1 [2, 0, **2**, 3]

0 0 1 0:0

0 0 1 <u>1</u>:0 [2, 0, 2, **3**]

Hill Climbing Algorithm – Improving vs. Non-worsening

```
while (termination criteria not satisfied){
    …
    for …{ // single pass
            …
            if (tmpEval ≤ bestEval) {// accept non-worsening moves
                …
            }  // end if
    } // end for
            …
} //end while
```

Exercise – Applying Best Improvement to a MAX-SAT Problem Instance (Accepting Non-worsening Moves)

- $(\neg x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

$x_0\ x_1\ x_2\ x_3{:}f_2$

➡ 0 0 0 0:1, minimising f_2 (No. of unsatisfied clauses)

j

iteration#0: <u>1</u> 0 0 0:1

iteration#1: 0 <u>1</u> 0 0:1

iteration#2: 0 0 <u>1</u> 0:0

iteration#3: 0 0 0 <u>1</u>:0 ✓

```
bestEval = evaluate(currentSolution); improved = false;
for(j=0;(j<length[currentSolution]);j++){ // left to right scan, single pass
  bitFlip(currentSolution, j); // flips jth bit of current solution
  tmpEval = evaluate(currentSolution);
  if (tmpEval ≤ bestEval) {  // accept non-worsening solutions
        // remember the bit which yields the best value after evaluation
        bestIndex= j;
        bestEval = tmpEval;
        improved = true;
  } // end if
  bitFlip(currentSolution, j); // go back to the initial current solution
} // end for
if (improved)  bitFlip(currentSolution, bestIndex);
```

Hill Climbing Algorithms – When to Stop

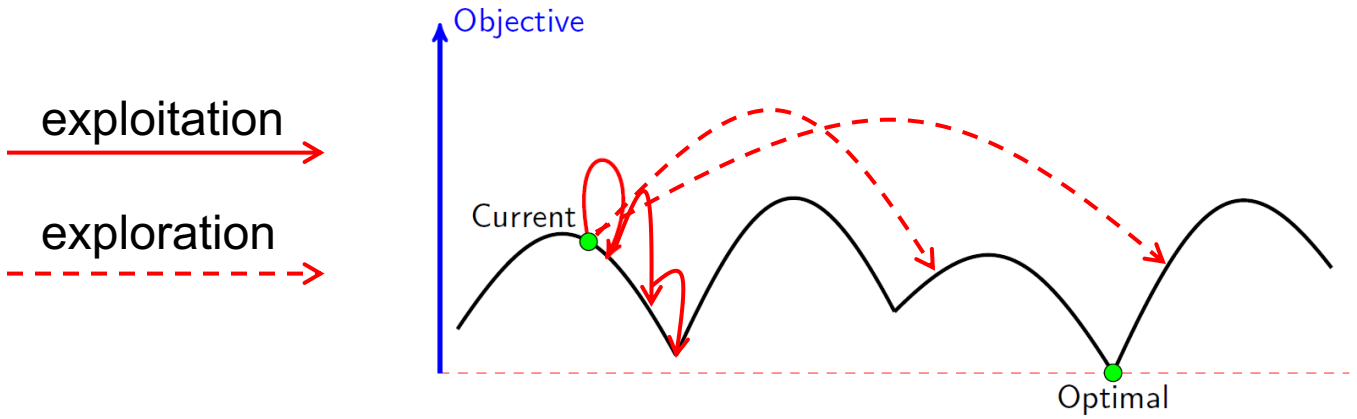
```
while (termination criteria not satisfied){
    …
    for …{ // single pass
                …

    } // end for
          …
} //end while
```

Hill Climbing Algorithms – When to Stop (cont.)

- If the target objective is known (e.g., minimum value for f_2 is known which is 0), then the search can be stopped when that target objective value is achieved.
- Hill climbing could be applied repeatedly until a termination criterion is satisfied (e.g. maximum number of evaluations is exceeded which is a factor of the string length)
  - Note that there is no point applying Best Improvement, Next Improvement and Davis's (Bit) Hill Climbing if there is no improvement after any single pass over a solution.
  - Random Mutation Hill Climbing requires consideration.

Hill Climbing versus Random Walk

- **A Hill-climbing** method ***exploits*** the best available solution for possible improvement but neglect exploring a large portion of the search space
- **Random walk** (performs search in the search space, sampling new points with equal probability, e.g., random bit flip, random swap) ***explores*** the search space thoroughly but misses exploiting promising regions.

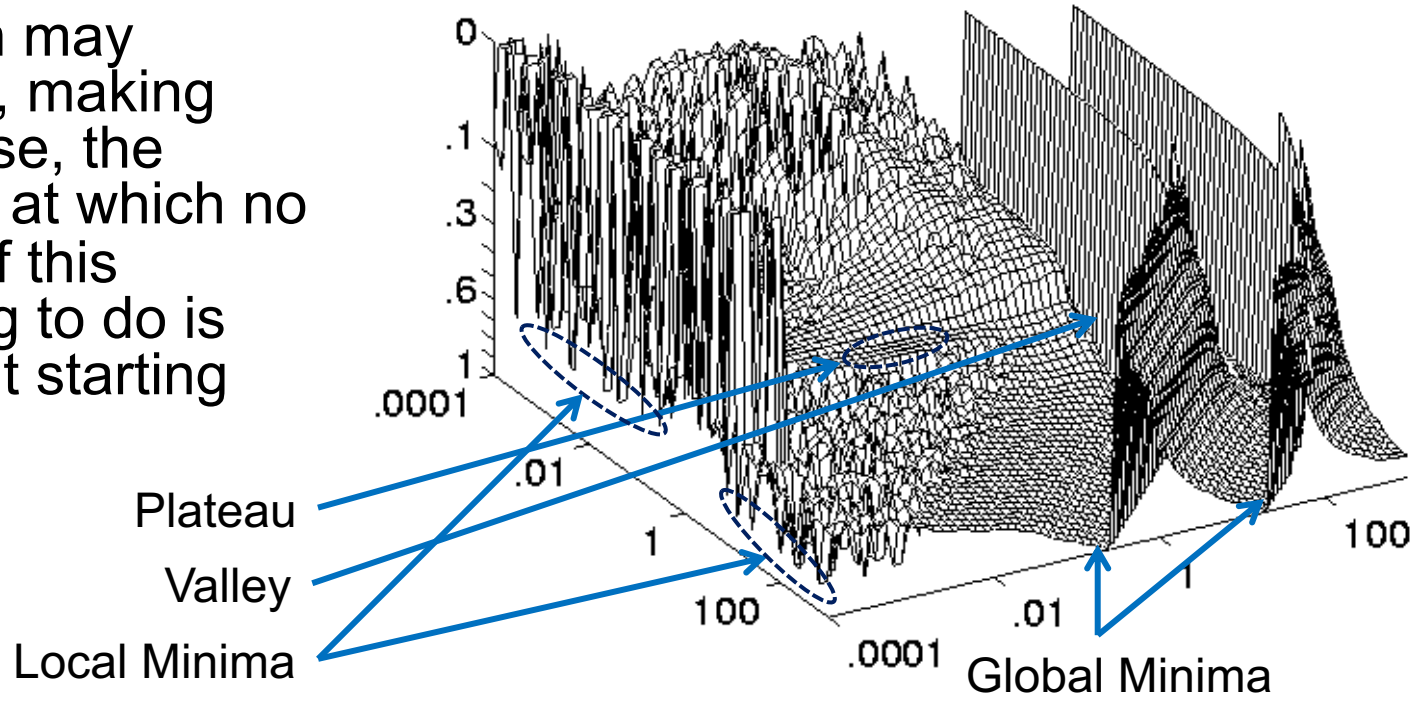


Strengths of Hill Climbing

- Very easy to implement, requiring:
  - a representation,
  - an evaluation function,
  - a measure that defines the neighbourhood around a point in the search space.

Weaknesses of Hill Climbing I

- **Local Optimum:** If all neighboring states are worse or the same. The algorithm will halt even though the solution may be far from satisfactory.
- **Plateau (neutral space/shoulder):** All neighboring states are the same as the current state. In other words the evaluation function is essentially flat. The search will conduct a random walk.
- **Ridge/valley:** The search may oscillate from side to side, making little progress. In each case, the algorithm reaches a point at which no progress is being made. If this happens, an obvious thing to do is start again from a different starting point.



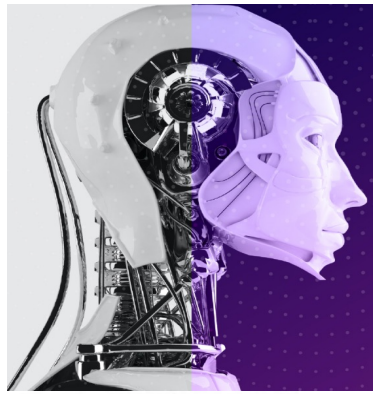
Weaknesses of Hill Climbing II

- As a result, HC may not find the optimal solution and may get stuck at a local optimum
- No information as to how much the discovered local optimum deviates from the global (or even other local optima)
- Usually no upper bound on computation time
- Success/failure of each iteration depends on starting point
  - success defined as returning a local or a global optimum

Home Exercise
(Forum Discussion is Open)

- Assume that there are *N* examinations to timetable within *M* weeks (5 working/exam day per week), and 3 exam sessions per day. You are provided with a list of students and the exams that each one of them takes. The goal is to schedule all exams in such a way that there are no clashes for the students.
  - What representation would you use for the solution algorithm?
  - How would you design your objective function?
  - Is it possible to design delta evaluation?

READING



5. Performance Analysis of Stochastic Local Search Methods – Preliminaries

Computational Optimisation & Learning Lab

The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Which Stochastic Search Algorithm Performs Better for Solving Problem X?

- Algorithm A is new, B & C are previous approaches applied to the instance Inst1 of the minimising problem X
- Assume all algorithms are run for the same number of objective function evaluations, and experiments are fair
- Each experiment is repeated for 30 times, that is, an algorithm is run for 30 times, independently, on an instance

	Algorithm A				Algorithm B				Algorithm C			
Instance	avg.	std.	med.	min.	avg.	std.	med.	min.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	5.7	3.3	6	1	11.6	4.9	12	3

run/trial	A	B	C
1	1	8	20
2	1	6	11
3	0	2	15
4	1	1	12
5	0	3	11
6	2	4	13
7	0	8	3
8	1	6	19
9	1	1	15
10	0	8	3
11	1	8	11
12	2	11	9
13	1	3	14
14	2	11	12
15	1	5	6

So, which algorithm performs the best for solving Problem X?

- **avg.**: mean objective value computed by averaging the objective values of 30 solutions returned by an algorithm from 30 independent trials/runs
- **std.**: standard deviation associated with avg.
- **med.**: median objective value
- **min.**: objective value of the best solution found in all trials/runs

Which Stochastic Search Algorithm Performs Better for Solving Problem X?

- Algorithm A is new, B & C are previous approaches applied to the instance Inst1 of the minimising problem X
- Assume all algorithms are run for the same number of objective function evaluations, and experiments are fair
- Each experiment is repeated for 30 times, that is, an algorithm is run for 30 times, independently, on an instance

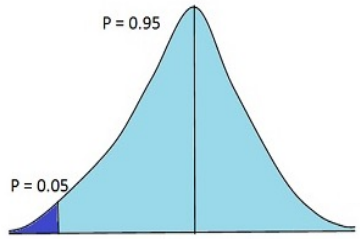
	Algorithm A				Algorithm B				Algorithm C			
Instance	avg.	std.	med.	min.	avg.	std.	med.	min.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	5.7	3.3	6	1	11.6	4.9	12	3

run/trial	A	B	C
1	1	8	20
2	1	6	11
3	0	2	15
4	1	1	12
5	0	3	11
6	2	4	13
7	0	8	3
8	1	6	19
9	1	1	15
10	0	8	3
11	1	8	11
12	2	11	9
13	1	3	14
14	2	11	12
15	1	5	6

So, which algorithm performs the best for solving Problem X?
Any comment for 1 instance is valid for 1 problem instance.
So, Algorithm A performs the best for the instance with the label Inst1 of the problem X based on mean, median, best (minimum) objective values.

Which Stochastic Search Algorithm Performs Better for Solving Problem X?

- Apply non-parametric statistical test – one tailed:
  - E.g.: Given two algorithms; X versus Y, > (<) denotes that X (Y) is better than Y (X) and this performance difference is statistically significant within a confidence interval of 95% and X ≥ Y (X ≤ Y) indicates that X (Y) performs better on average than Y (X) but no statistical significance (Wilcoxon signed rank test – e.g., http://vassarstats.net/wilcoxon.html)



	Algorithm A					Algorithm B					Algorithm C			
Instance	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	>	5.7	3.3	6	1	>	11.6	4.9	12	3

run/trial	A	B	C
1	1	8	20
2	1	6	11
3	0	2	15
4	1	1	12
5	0	3	11
6	2	4	13
7	0	8	3
8	1	6	19
9	1	1	15
10	0	8	3
11	1	8	11
12	2	11	9
13	1	3	14
14	2	11	12
15	1	5	6

- **A stronger conclusion can be provided for one instance (Inst1)**
- **Important**: Always repeat the experiments more than or equal to 30 times for any given instance for a meaningful statistical comparison

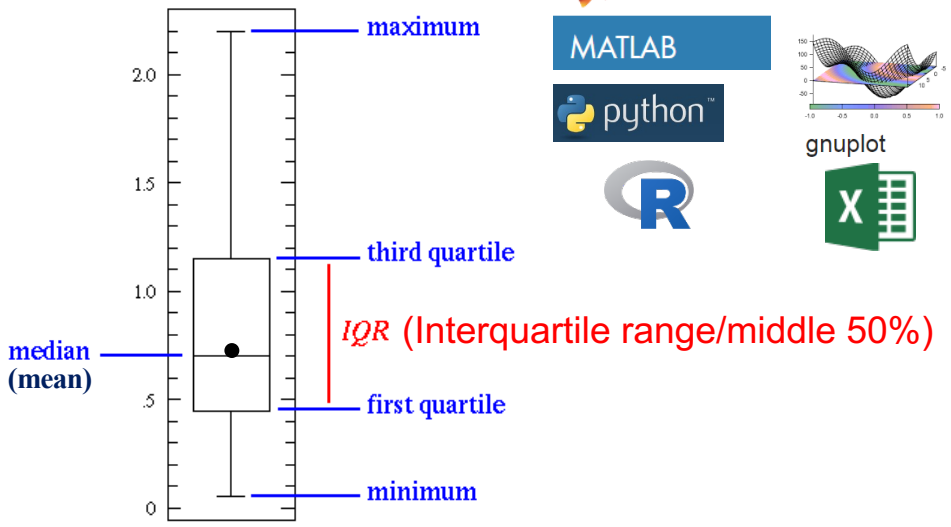
Which Stochastic Search Algorithm Performs Better for Solving Problem X?

Boxplots

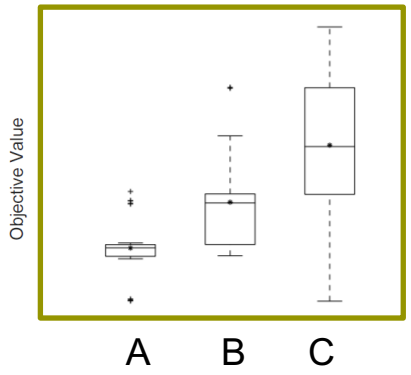
- Boxplots illustrates groups of numerical data through their quartiles.

  BoxPlotR: http://shiny.chemgrid.org/boxplotr/

- Example: boxplot of objective values obtained from 30 runs on the instance Inst1 from 3 algorithms
  - Outliers may be plotted as individual points.
  - **A stronger conclusion for Inst1 – next slide**



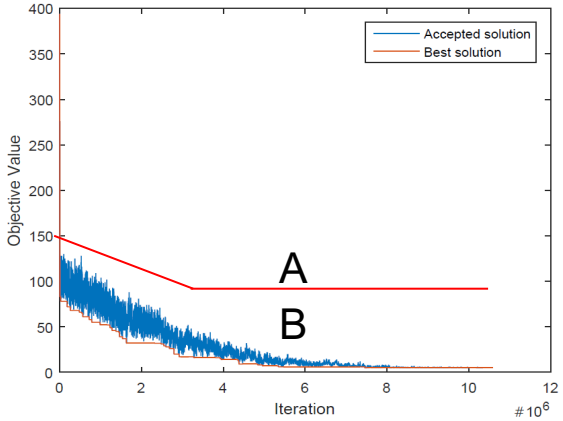
MathWorks MATLAB python gnuplot R X



run/trial	A	B	C
1	1	8	20
2	1	6	11
3	0	2	15
4	1	1	12
5	0	3	11
6	2	4	13
7	0	8	3
8	1	6	19
9	1	1	15
10	0	8	3
11	1	8	11
12	2	11	9
13	1	3	14
14	2	11	12
15	1	5	6

Performance Analysis Using Plots – Other Methods

- **Progress plot** – per instance: Objective value from a run or mean of objective values from multiple runs per iteration/time unit



Objective value of best and accepted solutions versus iteration from Algorithms A and B for the same instance from a sample run


University of
Nottingham
UK | CHINA | MALAYSIA
Q&A
ender.ozcan@nottingham.ac.uk
www.cs.nott.ac.uk/~pszeo/