

A large, semi-transparent image of the Earth from space serves as the background for the slide. The planet is shown in its entirety, with a bright blue glow at the horizon and a dark blue star-filled void in the background.

COM2001/2011 Artificial Intelligence Methods

Lecture 1

Prof Ender Özcan

Computer Science, Office: C86

Ender.Ozcan@nottingham.ac.uk

www.nottingham.ac.uk/~pszeo/



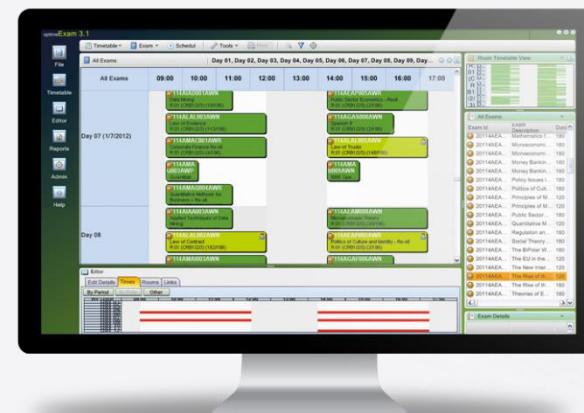
What Do the Following Things Have in Common?



Timetables



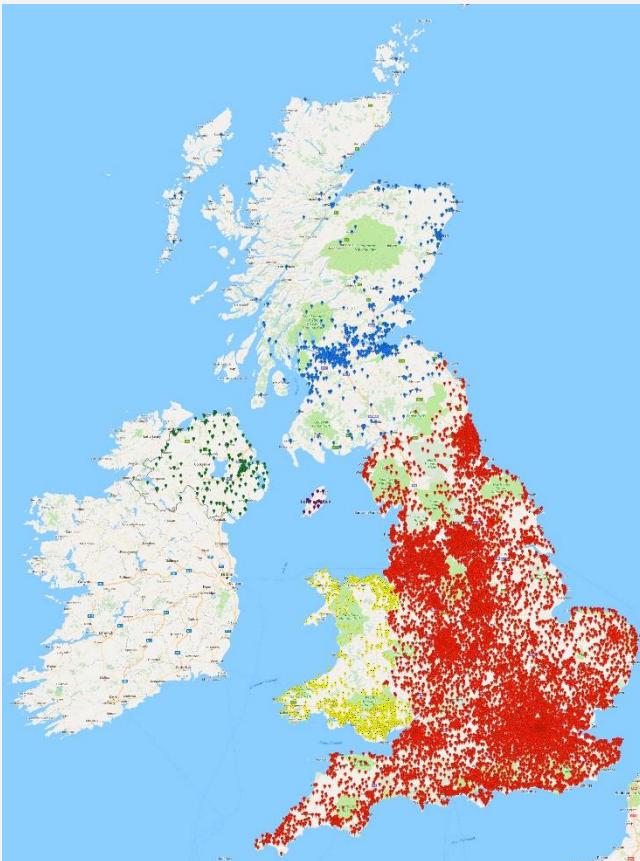
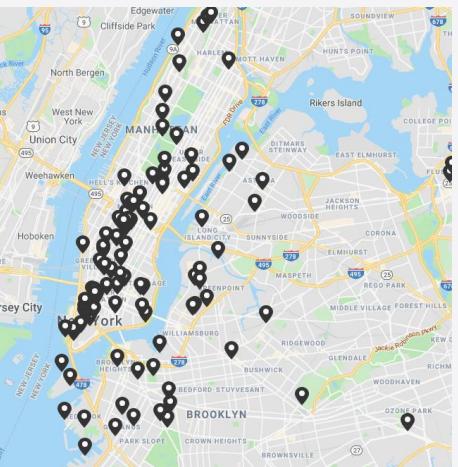
Nurse Rosters



Exam Timetables



Routing





Supply Chains



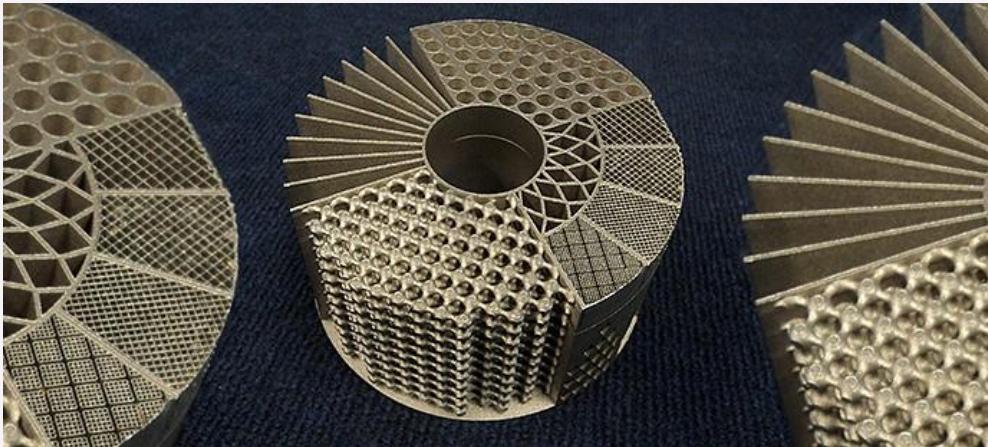


Wind-farms



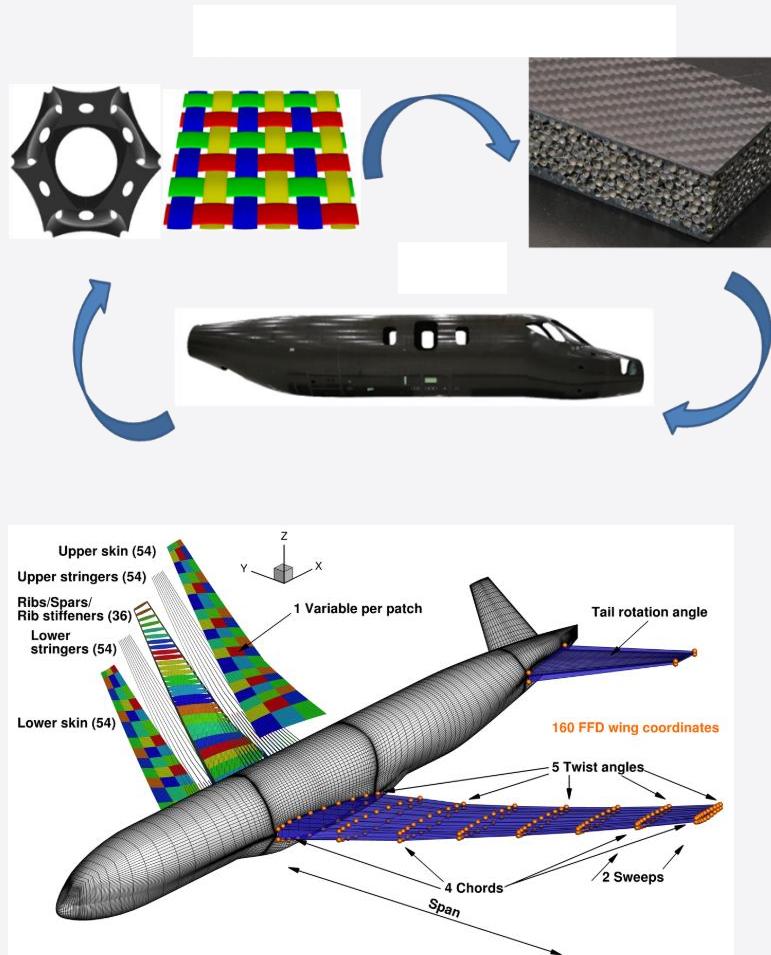


Additive Manufacturing (3D Printing)





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%]





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

- 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



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 $z = f(X)$, $\{g_i(X) \leq b_i, \} (= | \geq)$
 - where X is a vector of variables $\langle x_1, x_2, \dots, x_n \rangle$
- Next choose a mathematical or search method to solve the optimisation problem (searching the space of solutions and detecting the best/optimal solution)



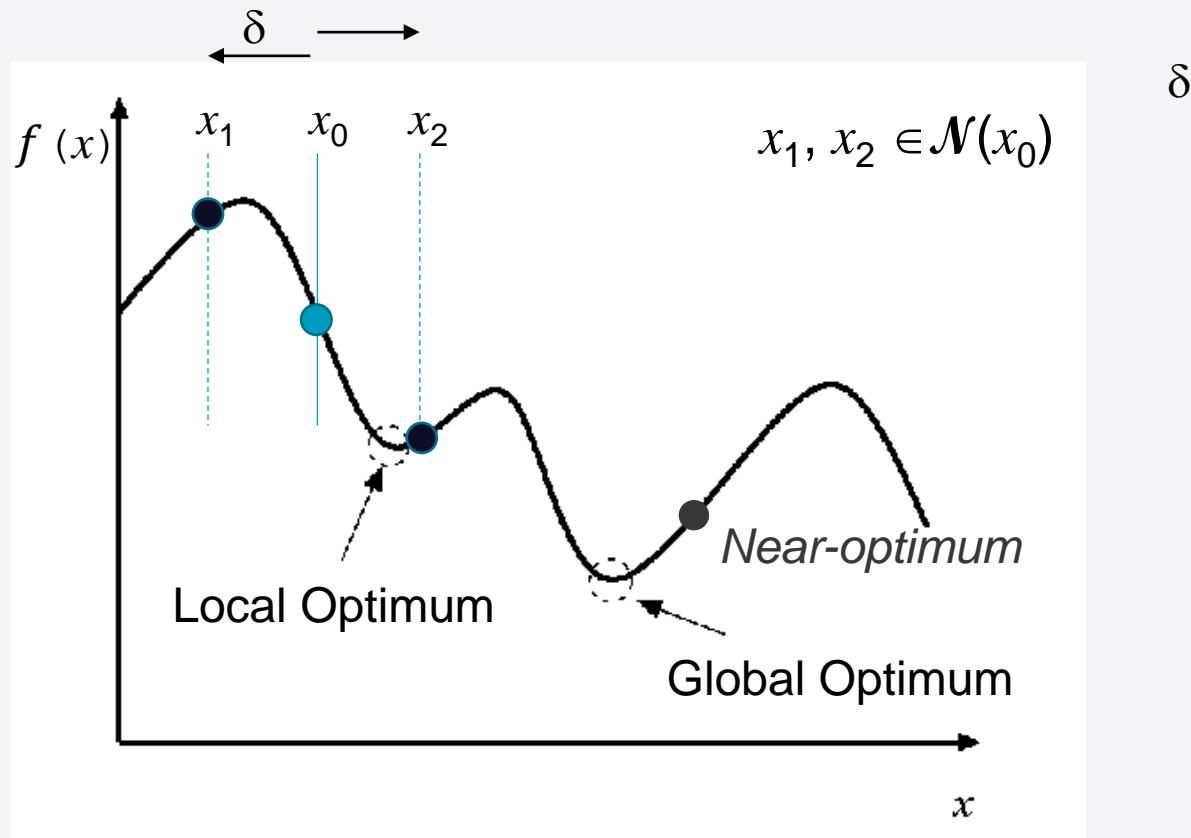
Definition – Global Optimisation

- 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

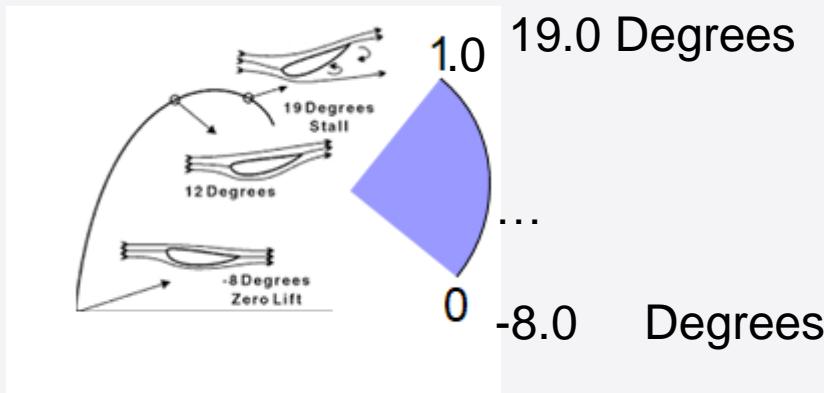
- Global Optimum: better than all other solutions (best)
- Local Optimum: better than all solutions in a certain neighbourhood, \mathcal{N}





Search in Continuous vs Discrete Space

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



- 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)



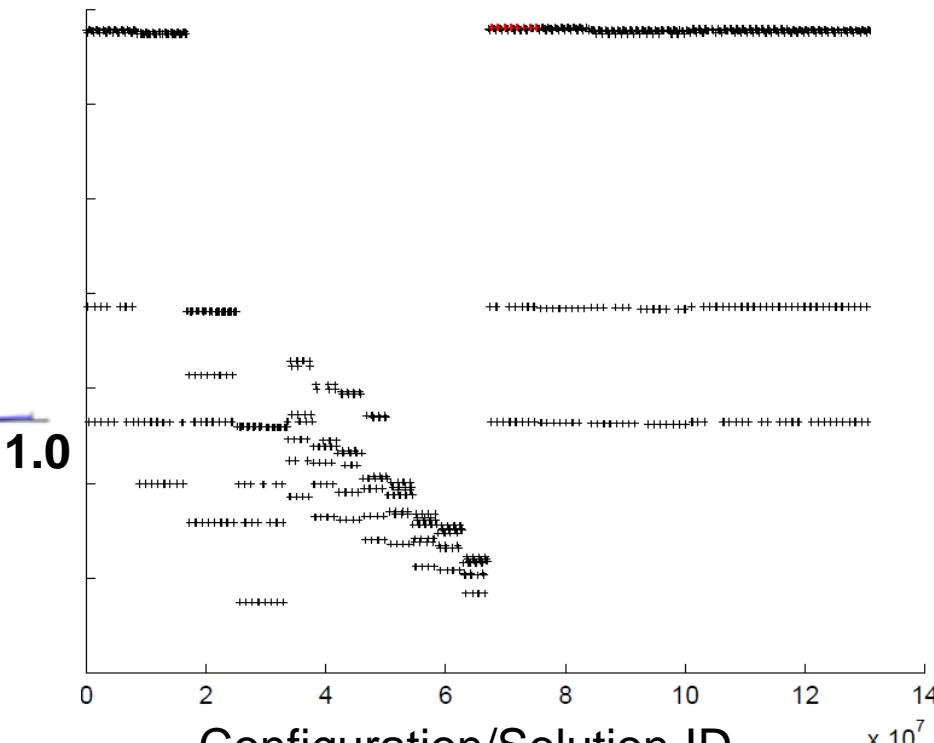


Search in Continuous vs Discrete Space

Performance



Average bus fullness



Real-valued (parameter)/
Continuous Optimisation

Organising tour busses

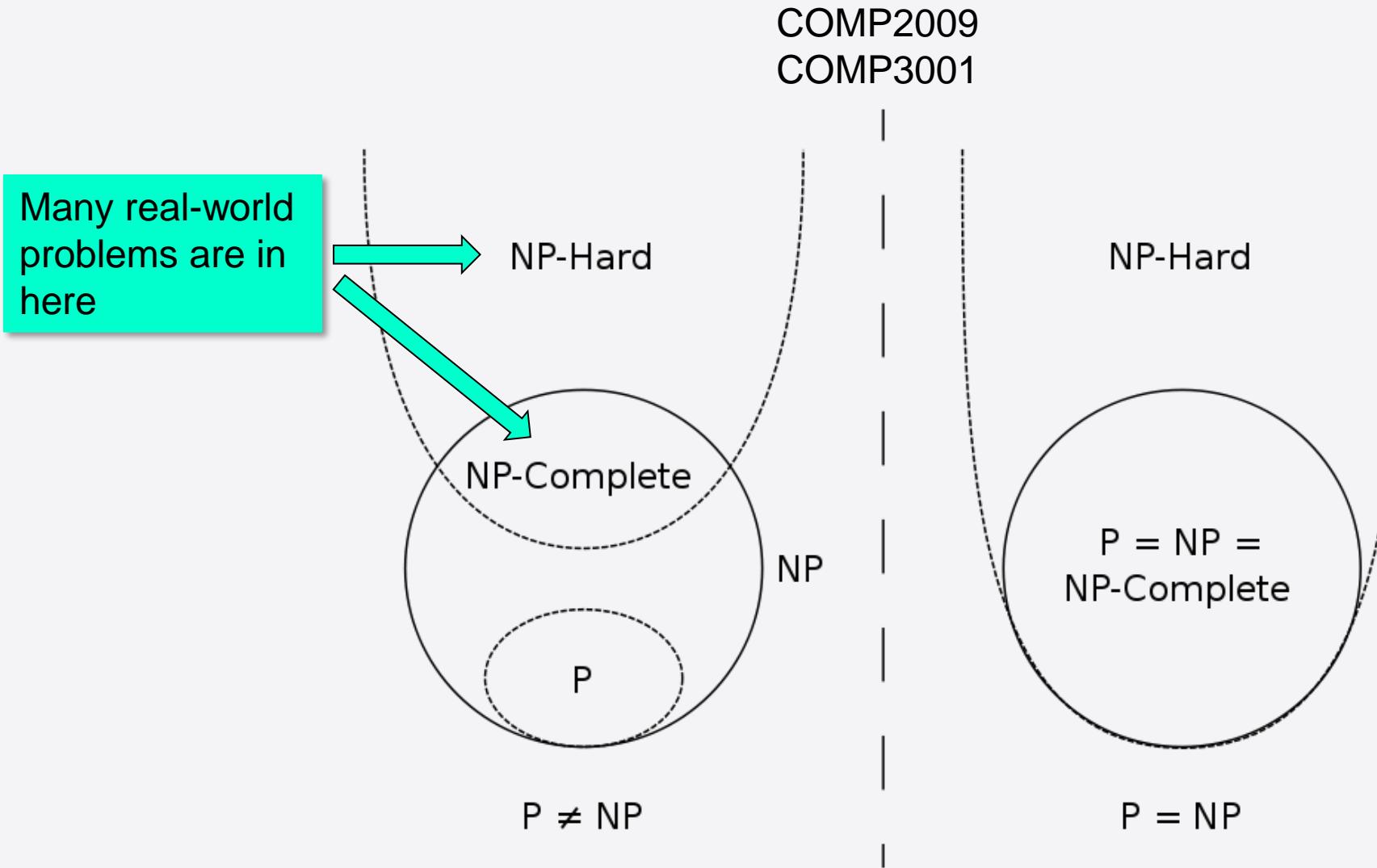


Definition – Problem and Problem Instance

- 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





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

deterministic ↔ stochastic

systematic ↔ local search

sequential ↔ parallel

single objective ↔ multi-objective

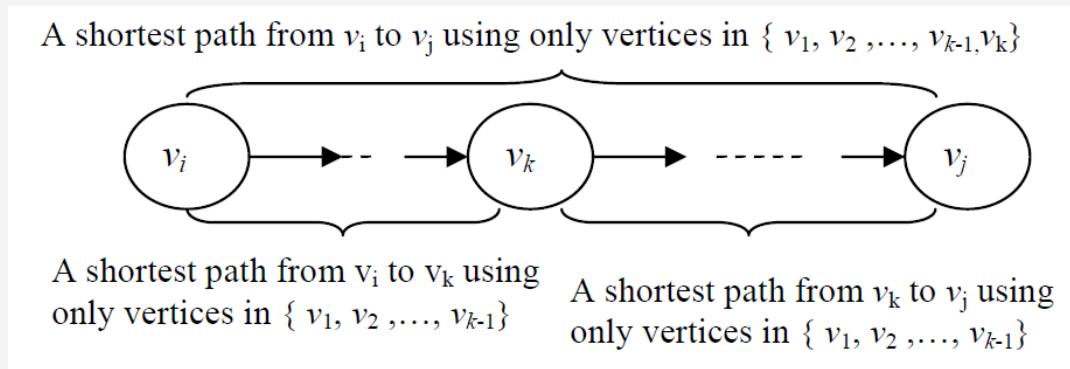
[See <http://www.cs.ubc.ca/~hoos/SLS-Internal/ch1.pdf> pp.23-30]



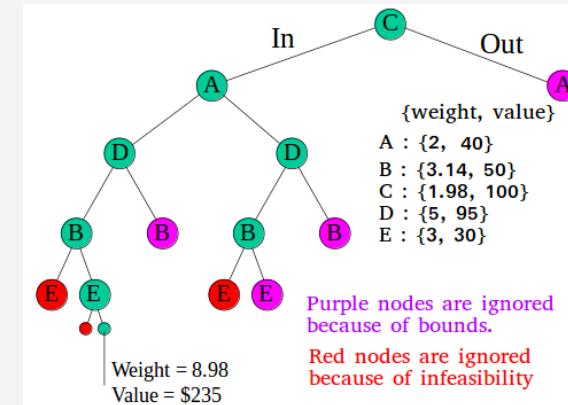
Example Optimisation Techniques I

Exact/Exhaustive/Deterministic Systematic Methods

■ Dynamic Programming



■ Branch & Bound



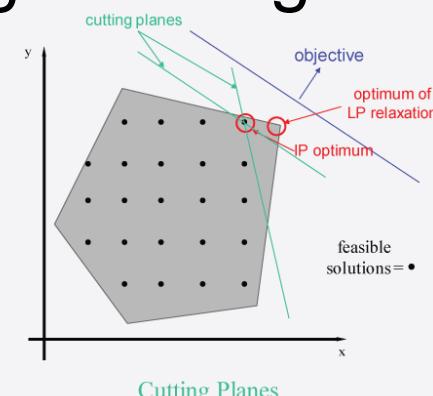
■ Constraint Satisfaction

Limitations:

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

■ Math programming

- ❖ ILP
- ❖ MILP





Example I – Bin Packing Problem Instance

- $V=524$

- $n=33$

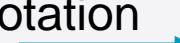
- Item sizes (33 items to pack): 85, 442, 10, 10, 10, 10, 10, 10, 10, 252, 252, 252, 252, 252, 252, 9, 9, 127, 127, 127, 127, 127, 127, 106, 106, 106, 12, 12, 12, 84, 46, 37

- How can we solve this problem using an exact method?

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

different notation
same logic

$x_1 \quad x_2 \quad \dots$
 $a_1 \quad a_2 \quad \dots$





Example I – Math Programming Solution COMP4038

Model

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

subject to $B \geq 1$,

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

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

$$y_i \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\} \forall j \in \{1, \dots, n\}$$

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



```
{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	
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.



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, 10, 252, 252, 252, 252, 252, 252, 9, 9, 127, 127, 127, 127, 127, 127, 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	
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				37	
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			
				252			
					252		
					127		
					127		
					127		
					127		
						106	
						106	
						106	
						85	
						84	
	37						
	37						
		12					
		12					
		12					
			10				
			10				
				10			
				10			
					10		
						9	
						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



Travelling Salesman Problem (TSP)

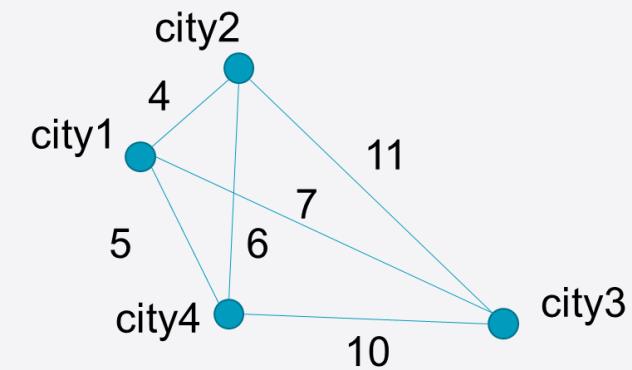
- “Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?” – NP hard.
- Underpins many real world problems: Planning, logistics, vehicle routing, telecommunication, manufacturing of microchips, genome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

- maximise/minimise $z = f(\mathbf{X})$, $\{g_i(\mathbf{X}) \leq b_i, \} (= | \geq)$
- where \mathbf{X} is a vector of variables $\langle x_1, x_2, \dots, x_n \rangle$

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) – $\sim 1.67 \times 10^{95}$ years (from [TOP500](#) project).

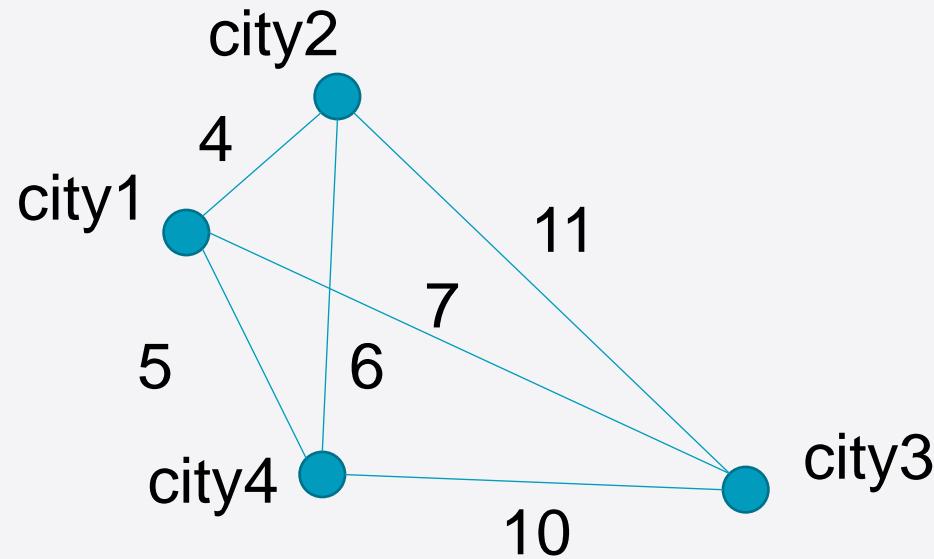


Examples – Heuristics for TSP

- The nearest neighbour (NN) algorithm
 - Constructive (Stochastic, Systematic)
- Pairwise exchange (2-opt)
 - Perturbative (Stochastic, Local Search)



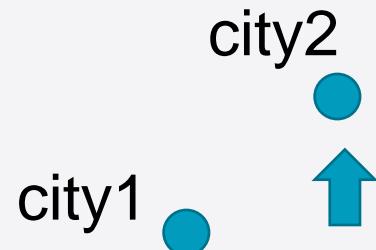
The nearest neighbour (NN) algorithm





The nearest neighbour (NN) algorithm

<city2>



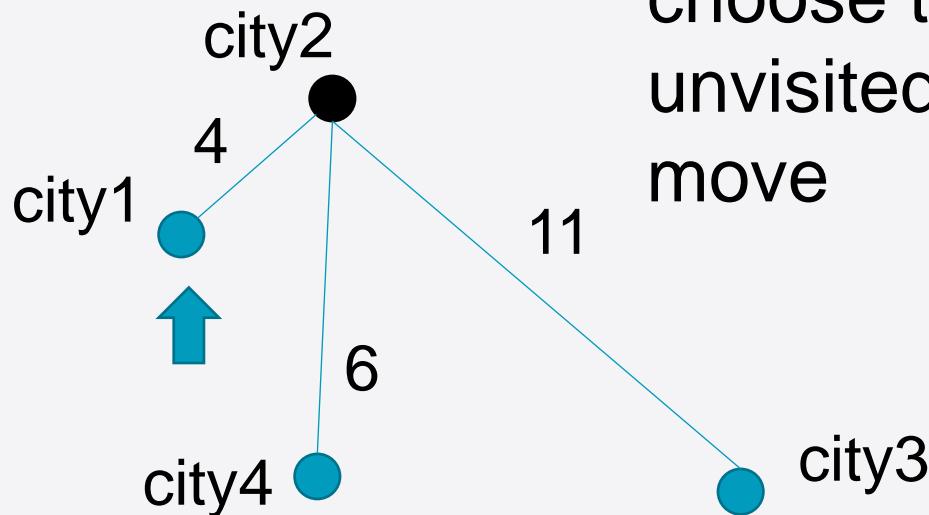
Select a starting city randomly

city4 ● city3



The nearest neighbour (NN) algorithm

<city2, >

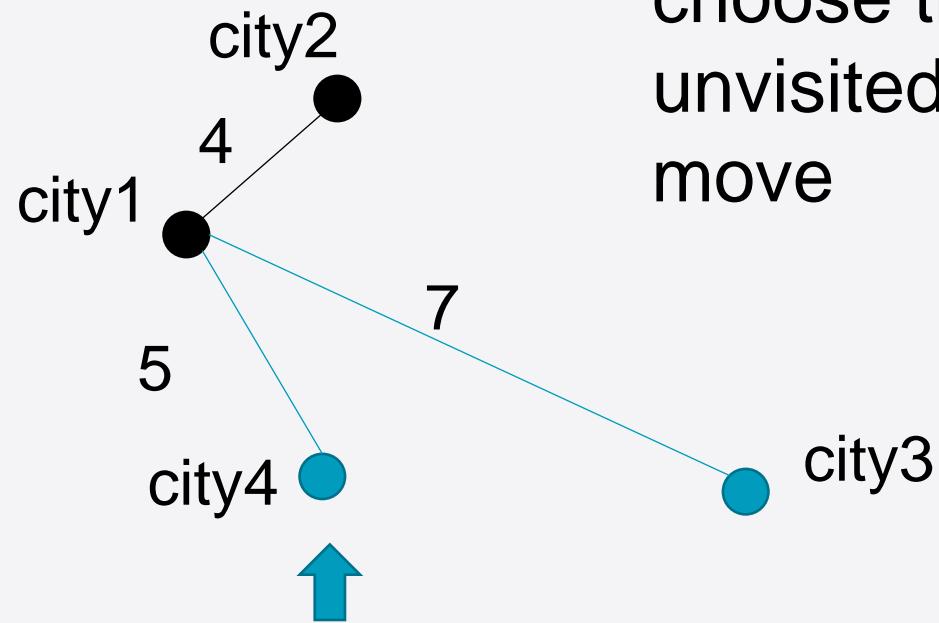


choose the nearest
unvisited city as the next
move



The nearest neighbour (NN) algorithm

<city2, city1, > : 4

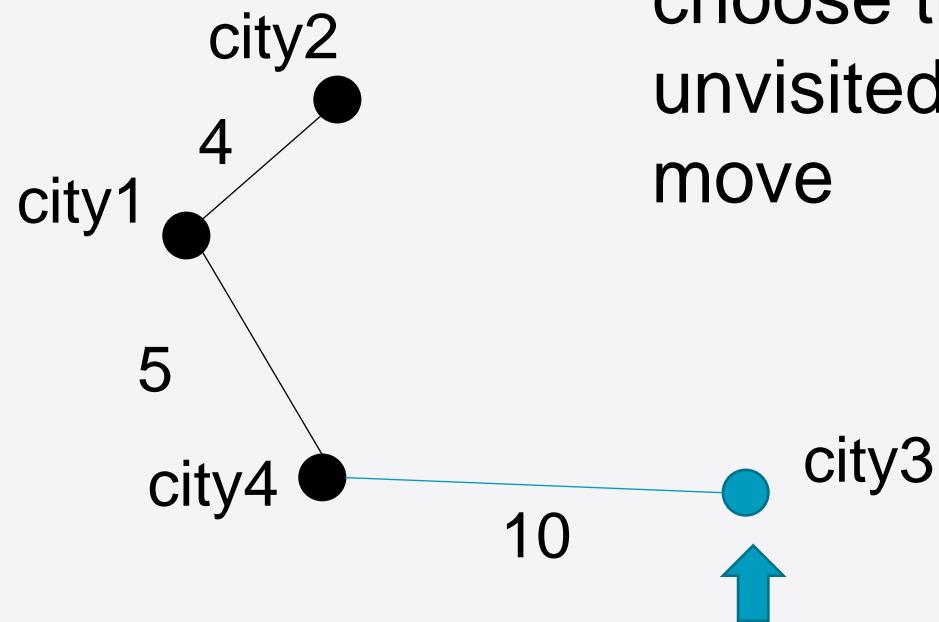


choose the nearest
unvisited city as the next
move



The nearest neighbour (NN) algorithm

<city2, city1, city4, > : 9



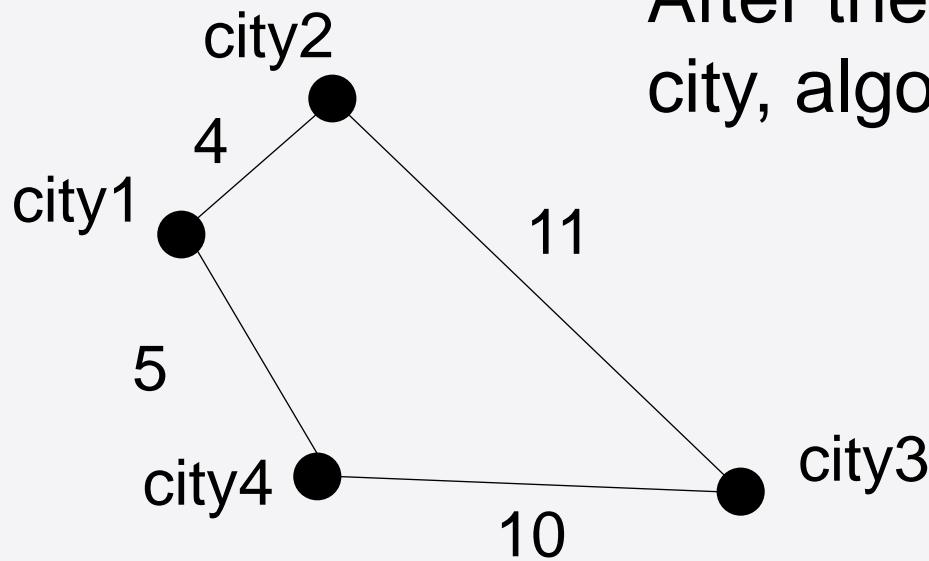
choose the nearest
unvisited city as the next
move



The nearest neighbour (NN) algorithm

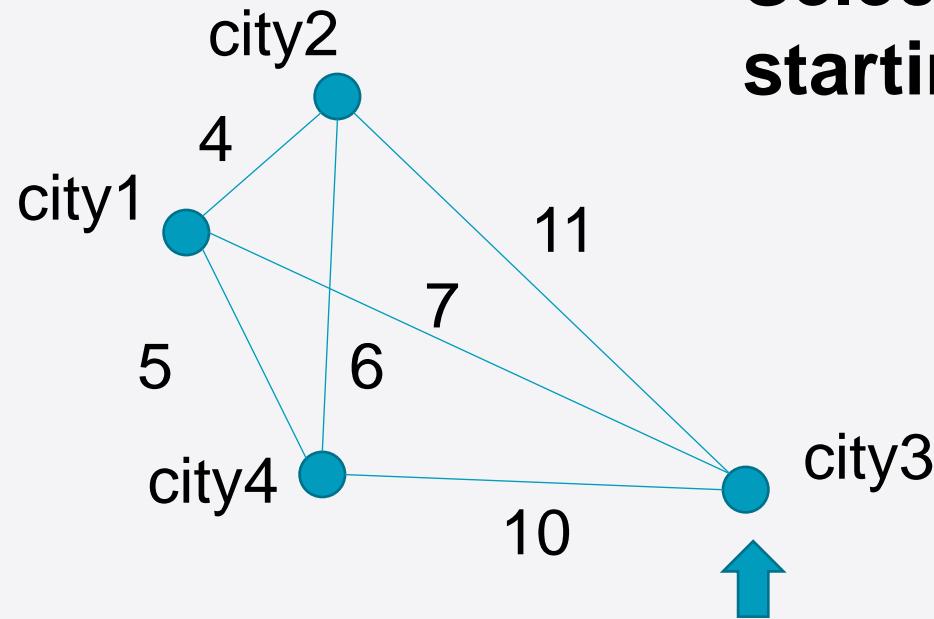
<city2, city1, city4, city3> : 30

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



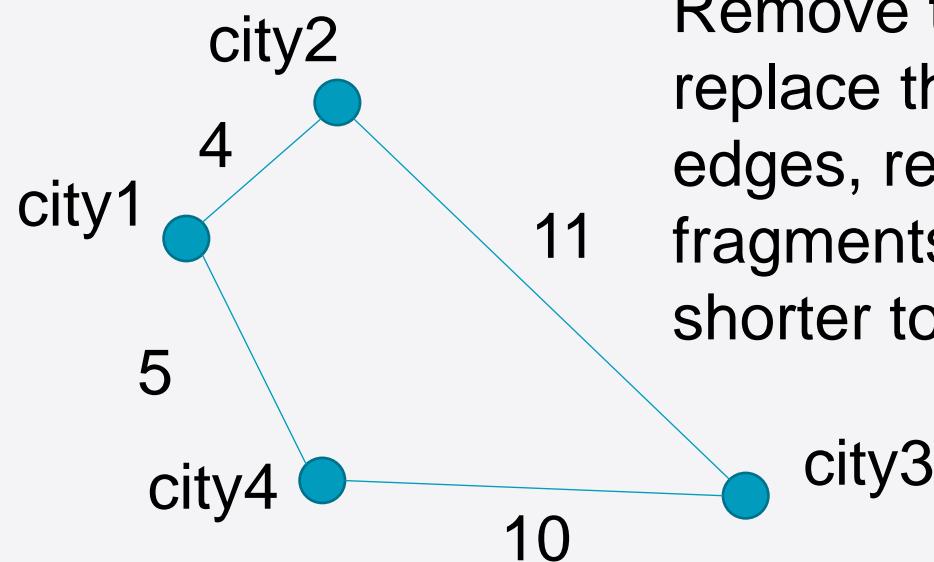
A Constructive Stochastic Local Search Algorithm for TSP

- 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

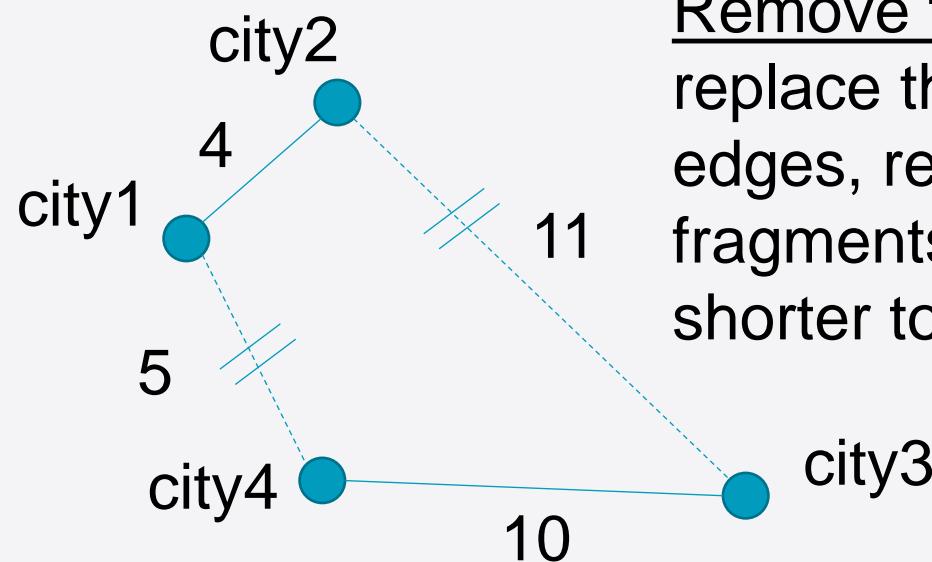


Remove two edges and replace them with two different edges, reconnecting the fragments into a new and shorter tour.



Pairwise exchange (2-opt)

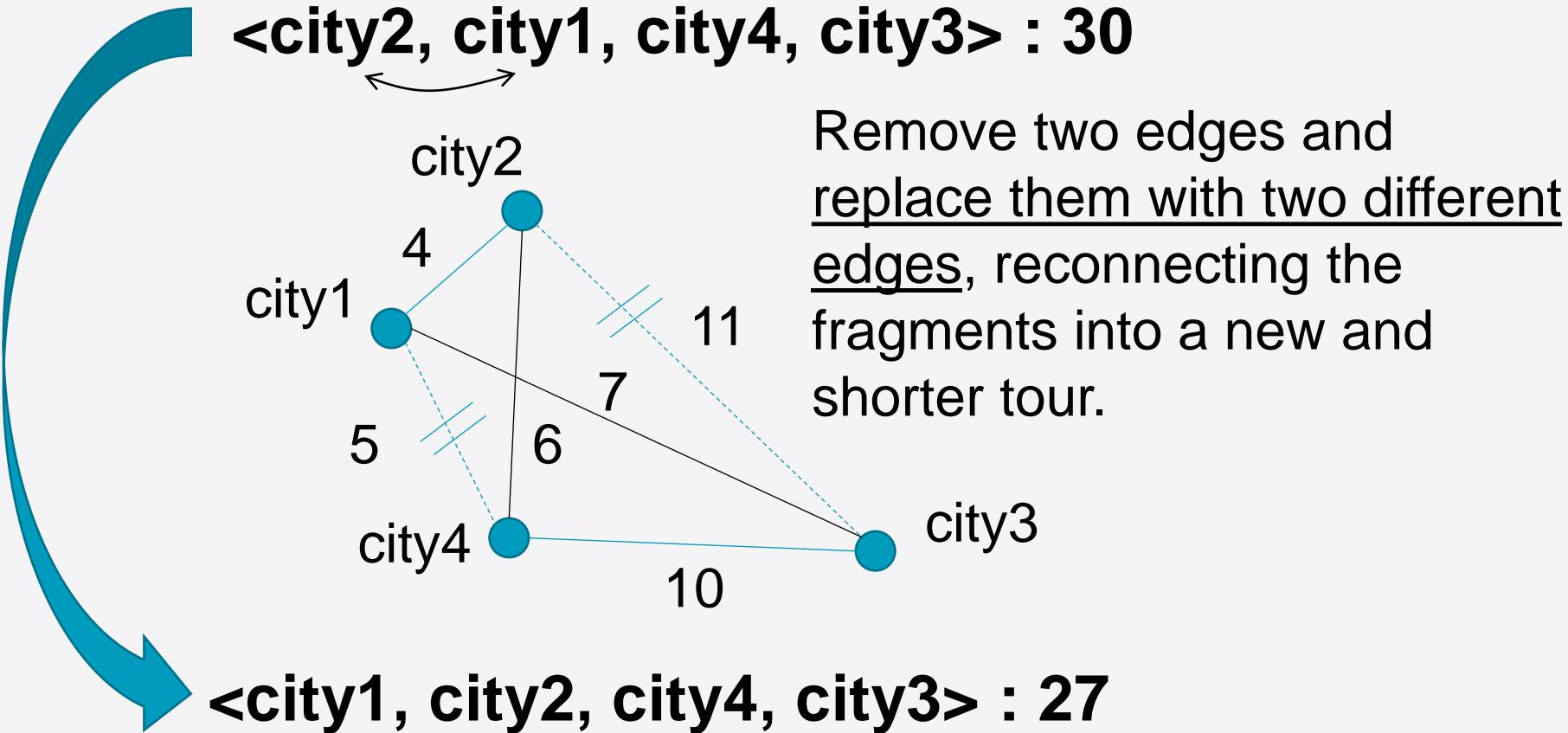
<city2, city1, city4, city3> : 30



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



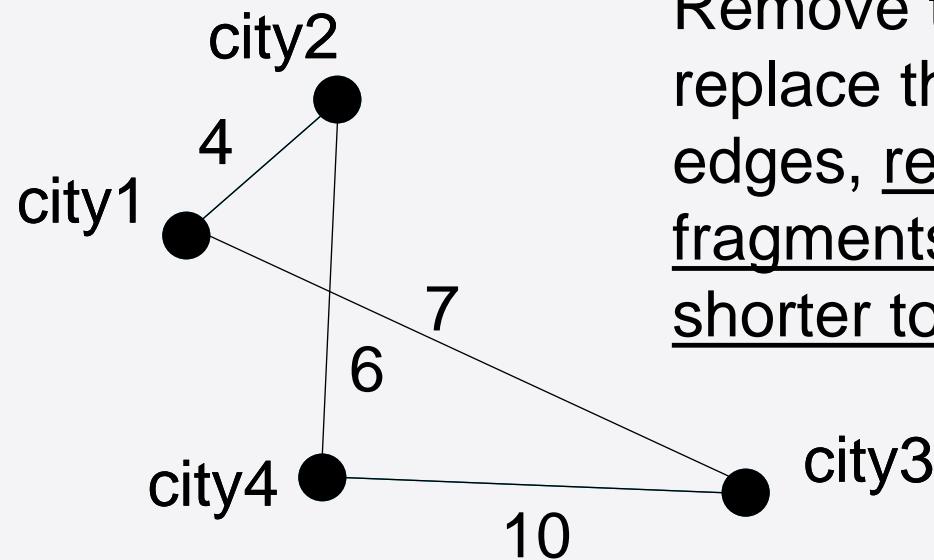
Pairwise exchange (2-opt)





Pairwise exchange (2-opt)

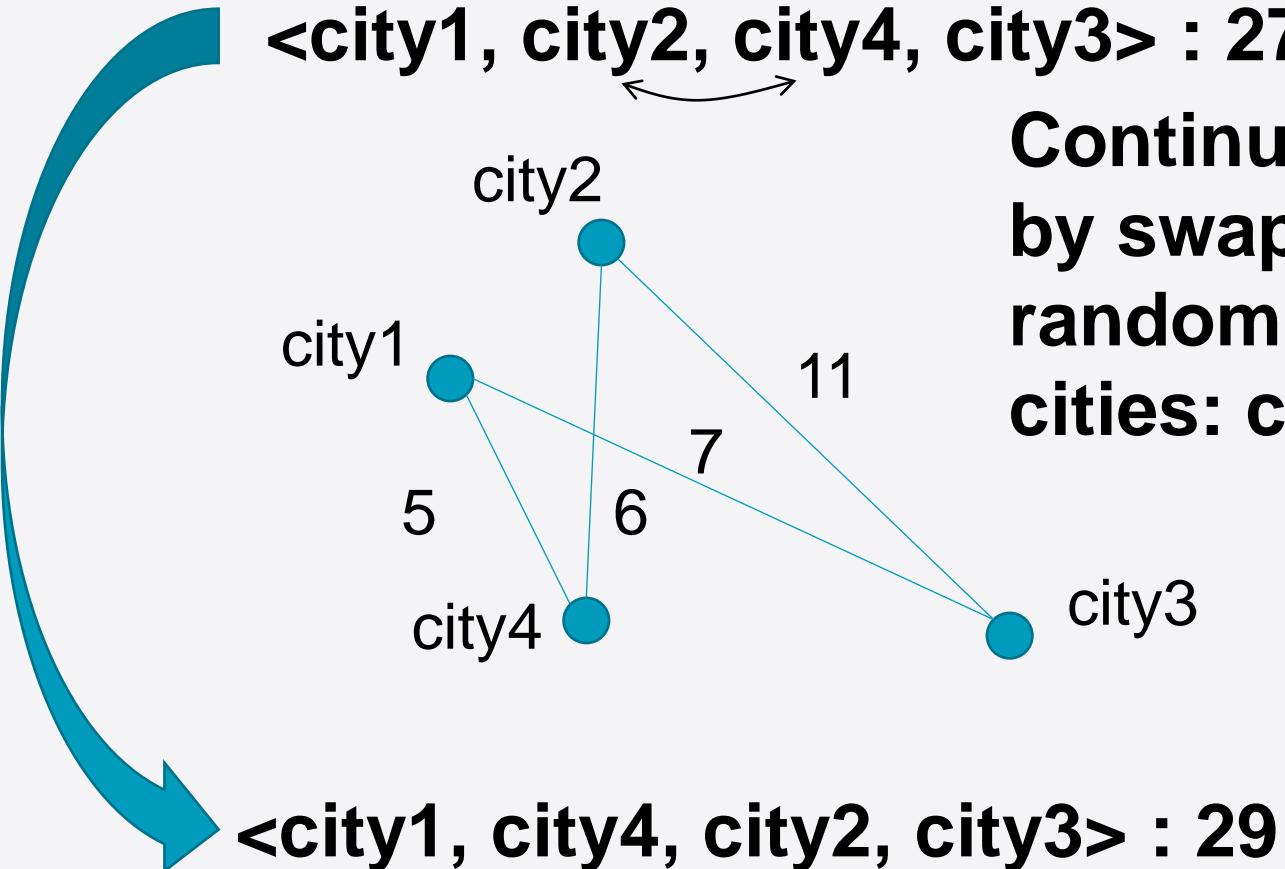
<city1, city2, city4, city3> : 27



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



Pairwise exchange (2-opt) – Notice



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

**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](https://doi.org/10.1016/j.orl.2004.04.001)]
 - Lin–Kernighan [[doi:10.1016/S0377-2217\(99\)00284-2](https://doi.org/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 ($2156^2 = 04648336$)
 - Middle digits are used as a random number 6483
 - Then this process is repeated
 - ($6483^2 = 42029289$)
- 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



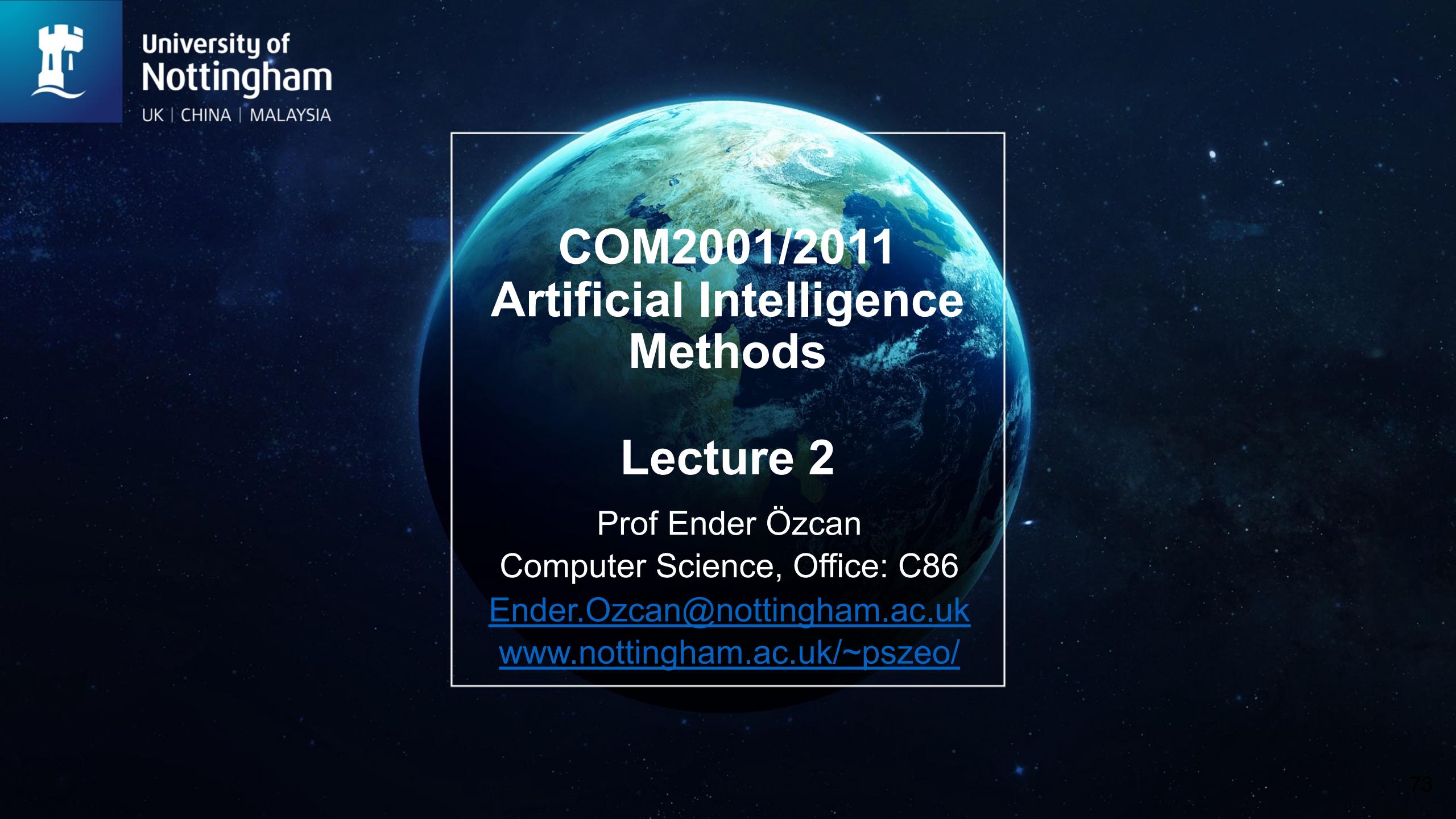
University of
Nottingham

UK | CHINA | MALAYSIA

Q&A

ender.ozcan@nottingham.ac.uk

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



A large, semi-transparent image of the Earth from space serves as the background for the slide. The planet is shown in its entirety, with visible continents and clouds, set against a dark blue background with small white stars.

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



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

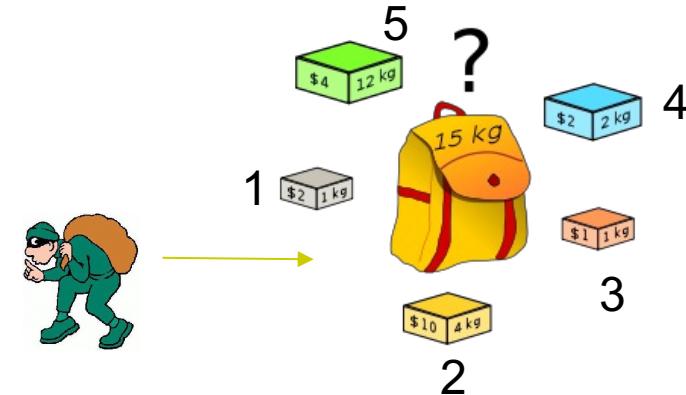
- **Binary encoding** is the most common
 - ▶ 10110010110010...1011

- 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

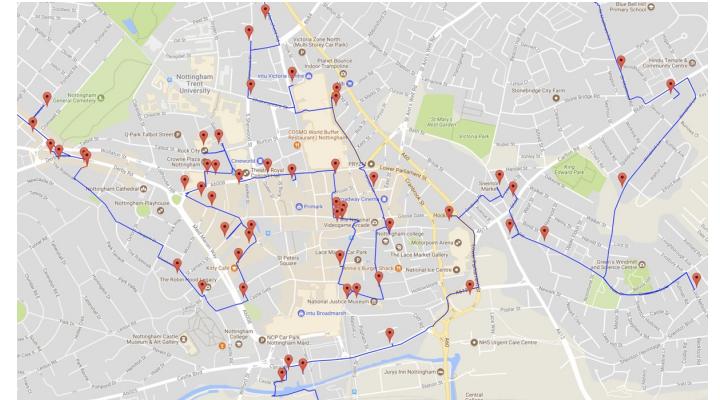




Representation (cont.)

- **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.)

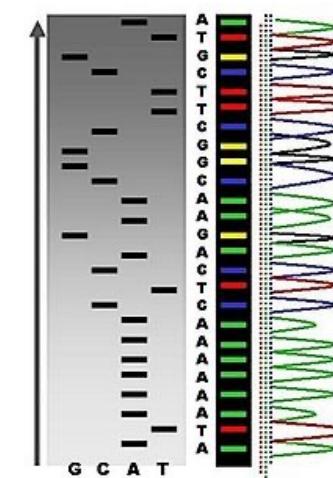
- **Integer encoding**
 - ▶ 1 3 4 5 5 5 4 1 1 ... 2 2 1
- E.g.: Personnel rostering problem, timetabling problem, layout/structure optimisation
 - ▶ Given an unlimited number of 5 different composite materials, which material would you use for each of the 15-layer composite structure maximising the sound 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)>

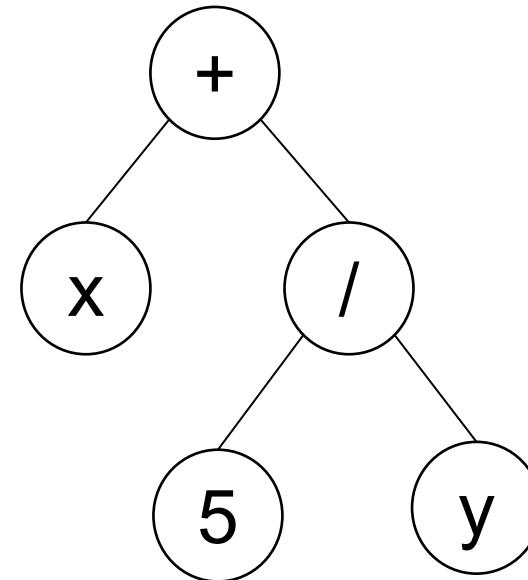




Representation (cont.)

- **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
design debugging

maximum quartet consistency
software package management

Max-Clique
fault localization
restoring CSP consistency
reasoning over bionetworks

MCS enumeration
heuristics for cost-optimal planning

optimal covering arrays
correlation clustering
treewidth computation

Bayesian network structure learning
causal discovery

visualization

model-based diagnosis

cutting planes for IPs

argumentation dynamics

...

[Park, 2002]

[Chen, Safarpour, Veneris, and Marques-Silva, 2009]

[Chen, Safarpour, Marques-Silva, and Veneris, 2010]

[Morgado and Marques-Silva, 2010]

[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]

[Ignatiev, Janota, and Marques-Silva, 2014]

[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]

[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]

[Lynce and Marques-Silva, 2011]

[Guerra and Lynce, 2012]

[Morgado, Liffiton, and Marques-Silva, 2012]

[Zhang and Bacchus, 2012]

[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]

[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]

[Berg and Järvisalo, 2014]

[Berg, Järvisalo, and Malone, 2014]

[Hyttinen, Eberhardt, and Järvisalo, 2014]

[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]

[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]

[Saikko, Malone, and Järvisalo, 2015]

[Wallner, Niskanen, and Järvisalo, 2016]

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

[\[Chapter on MAX-SAT\]](#)

[\[MAX-SAT Tutorial\]](#)

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, \dots, x_n), find the maximum number of clauses that can be satisfied by some truth assignment

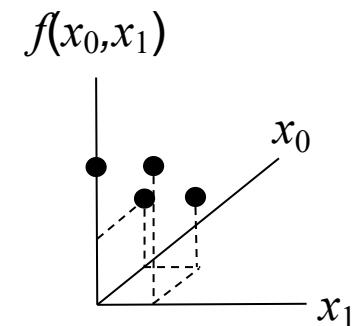
- ▶ E.g., $(\neg x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \Rightarrow$ Problem instance
 $(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1) \Rightarrow$ Another problem instance

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

MAX-SAT Problem – Candidate solution representation



$x_0 \ x_1$	<u>number of clauses satisfied</u>
<input type="checkbox"/> 0 0: $(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$ is true	2
<input type="checkbox"/> 0 1: $(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$ is true	2
<input type="checkbox"/> 1 0: $(\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$ is false	1
<input type="checkbox"/> 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

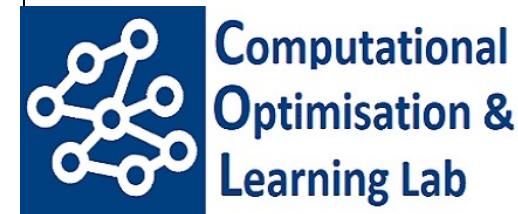
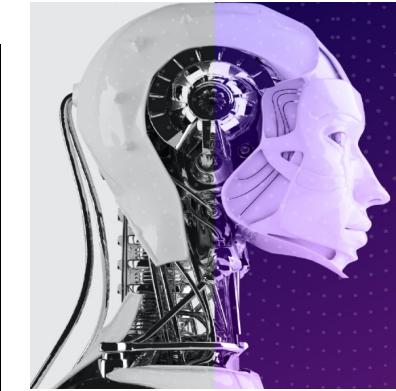
- Another problem instance

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

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

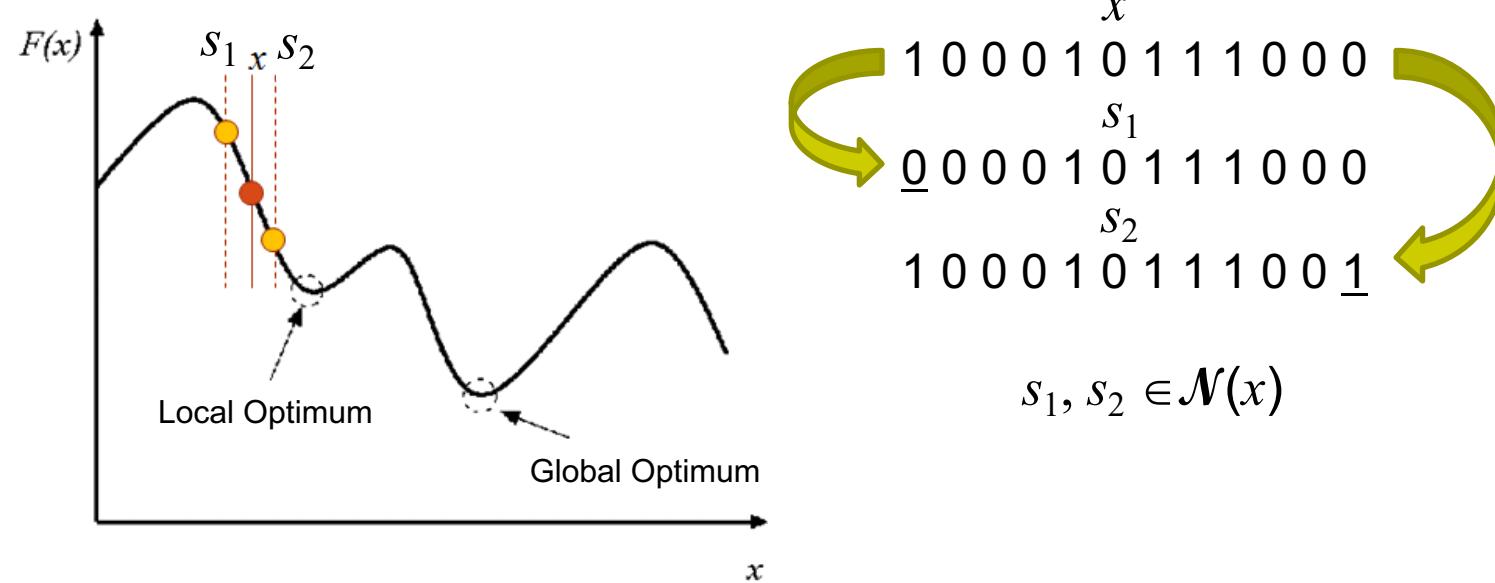


UNITED KINGDOM • CHINA • MALAYSIA



Definition

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



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., $\text{HD}(011,010)=1$, $\text{HD}(0101,0010)=3$
 - If the binary string is of size n , then the neighbourhood size is n .
 - Example: $1\ 0\ 1\ 0\ 0\ 0\ 1\ 1 \rightarrow 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1$
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 $(k-1)n$.
- 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



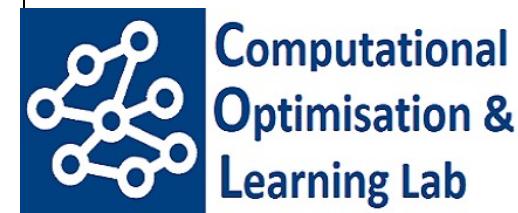
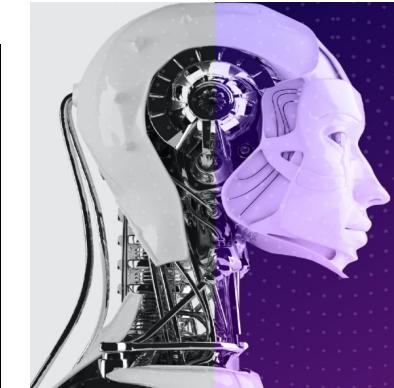
- Adjacent pairwise interchange: swap adjacent entries in the permutation
 - If permutation is of size n , then the neighbourhood size is $n-1$
 - Example: **5 1 4 3 2** → **1 5 4 3 2**
- Insertion operator: take an entry in the permutation and insert it in another position
 - Neighbourhood size: $n(n-1)$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





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:

$f_2 = (\text{No. of clauses} - C)$; No. of unsatisfied clauses

$f_3 = f_2 / (\text{No. of clauses})$;

- 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

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

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

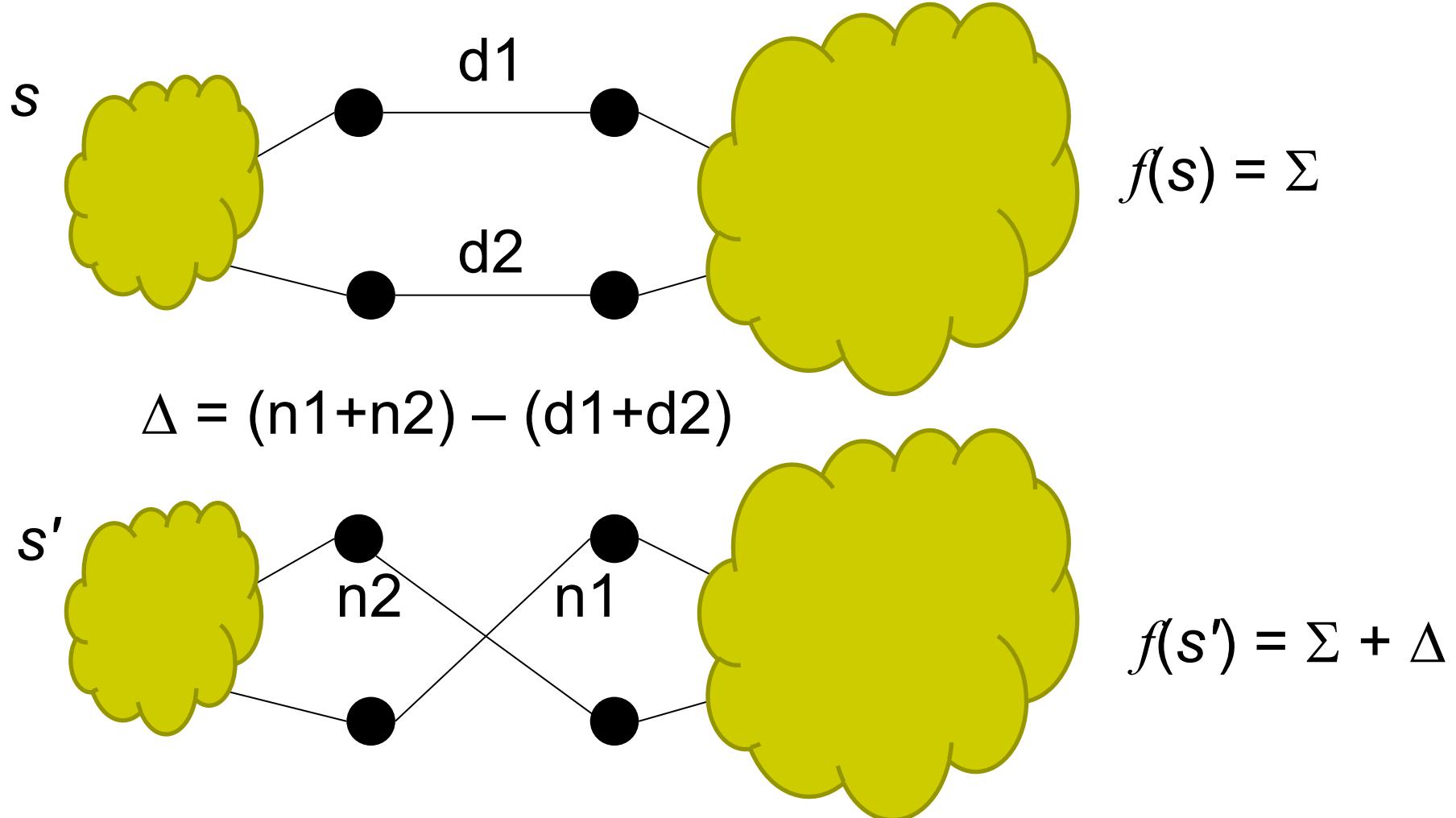
$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

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

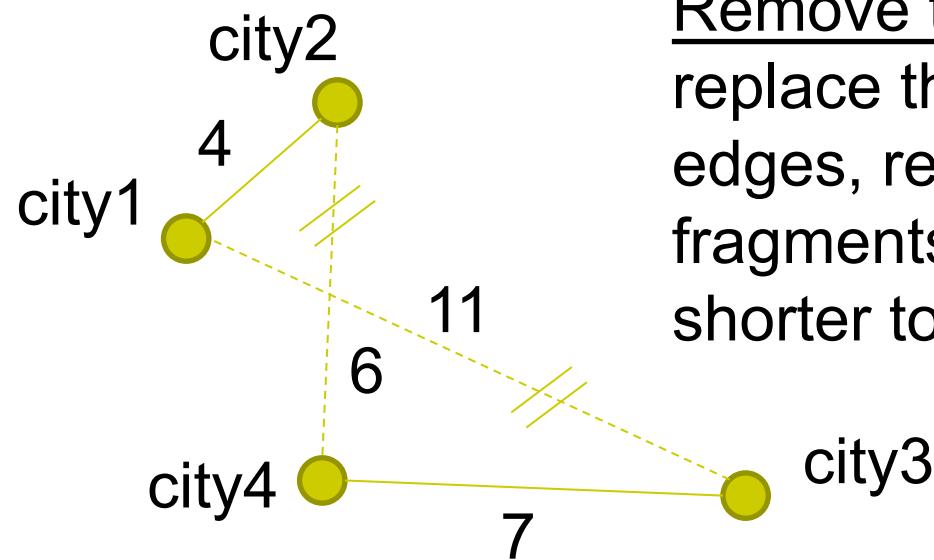


Example: Delta Evaluation for TSP II



$$f(s) = \Sigma$$

<city2, city1, city3, city4> : 28



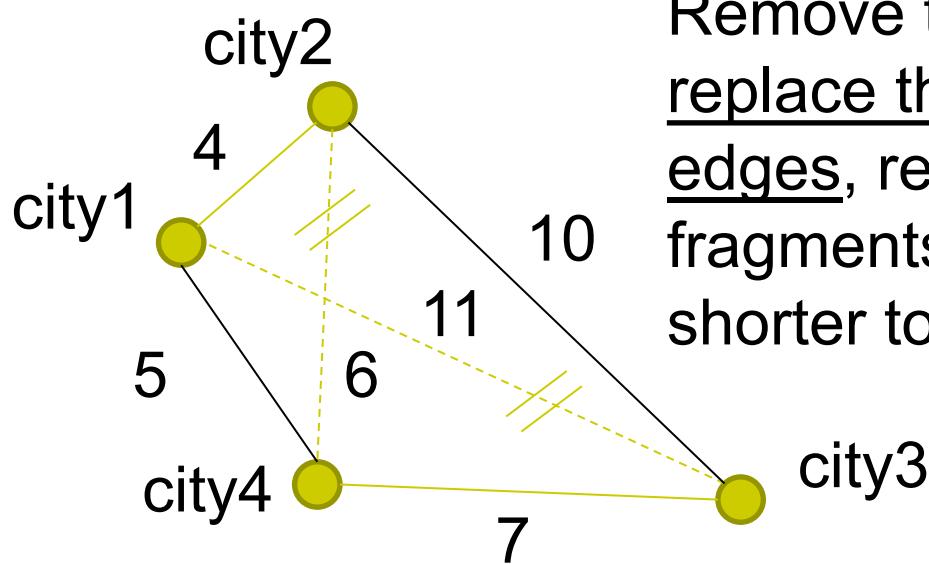
Remove two edges and replace them with two different edges, reconnecting the fragments into a new and shorter tour.

Example: Delta Evaluation for TSP III



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

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



$$\Delta = (n_1+n_2) - (d_1+d_2)$$

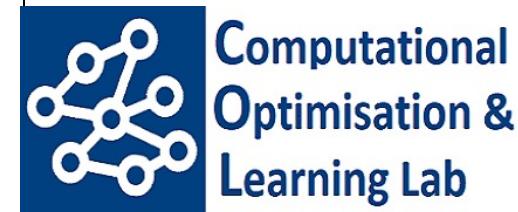
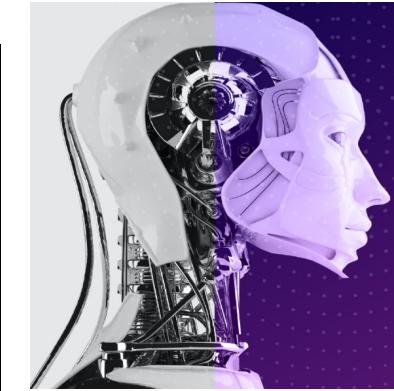
$$\Delta = (5+10) - (11+6) = -2$$



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

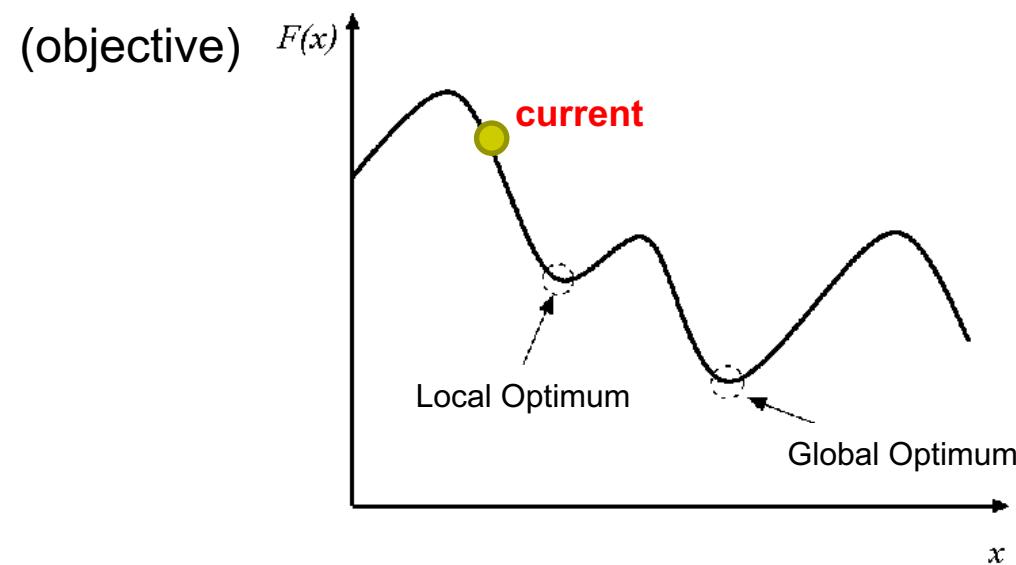


UNITED KINGDOM • CHINA • MALAYSIA

Search Paradigms – Perturbative Heuristics/Operators



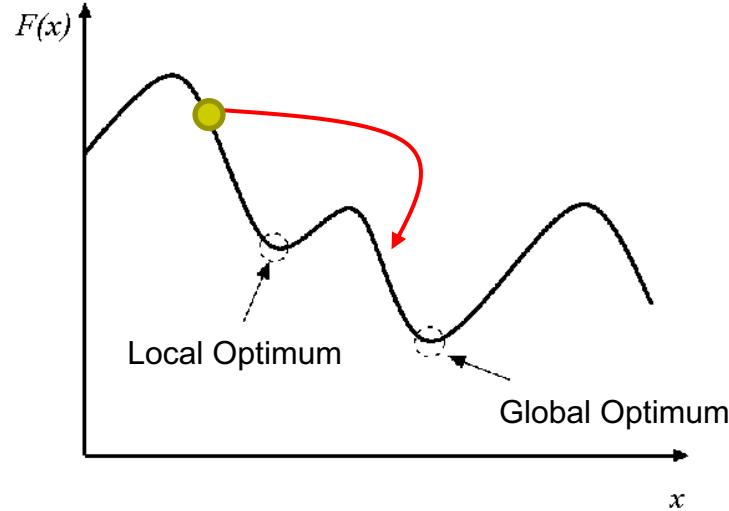
- Mutational (diversification/exploration) vs.
- Hill-climbing (intensification/exploitation)



Mutational Heuristic/Operator



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

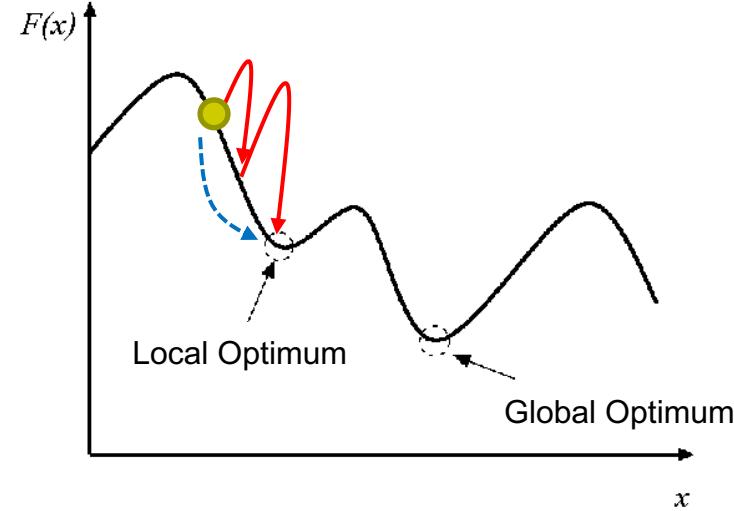


e.g., total number of constraint violations or a weighted sum of violations

Hill Climbing Heuristic/Operator



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



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: 1 0 0 0:1

iteration#1: 0 1 0 0:1

iteration#2: 0 0 1 0:0 ✓

iteration#3: 0 0 0 1:0

```
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/ascend) Hill-climbing

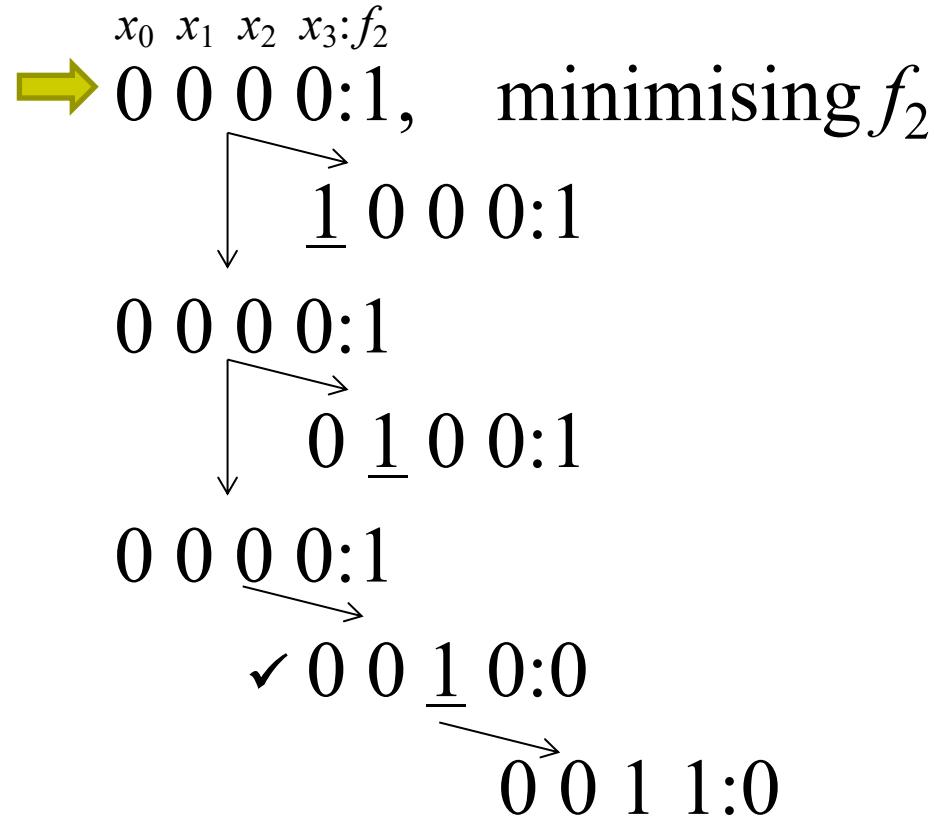


```
bestEval = evaluate(currentSolution);
for(j=0;(j<length[currentSolution]);j++){// single pass
    bitFlip(currentSolution, j); // flips jth bit of solution producing s' from s
    tmpEval = evaluate(currentSolution);
    if (tmpEval < bestEval)      // in case there is improvement
        bestEval = 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

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



```
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]



$\stackrel{x_0 \ x_1 \ x_2 \ x_3: f_2}{0 \ 0 \ 0 \ 0:1}$, minimising f_2

\downarrow
 $0 \underline{1} \ 0 \ 0:1$ [1, 2, 0, 3]

$0 \ 0 \ 0 \ 0:1$

\swarrow ✓ $0 \ 0 \ \underline{1} \ 0:0$ [1, 2, 0, 3] could
stop here

\downarrow
 $\underline{1} \ 0 \ 1 \ 0:0$ [1, 2, 0, 3]

$0 \ 0 \ 1 \ 0:0$

\swarrow
 $0 \ 0 \ 1 \ \underline{1}:0$ [1, 2, 0, 3]



Random Mutation Hill-climbing

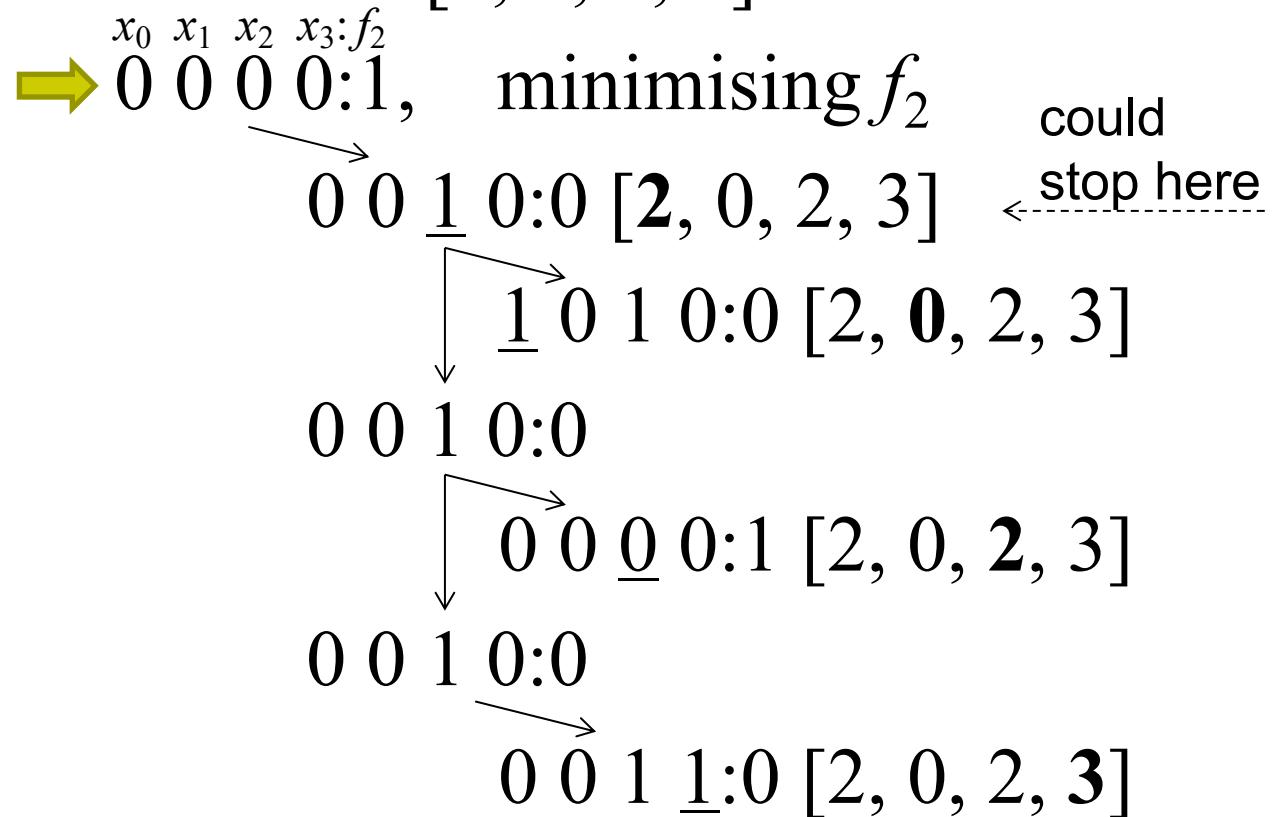
```
bestEval = evaluate(currentSolution);
for(k=0;(k<noOfSteps);k++){ // single pass if k=length[solution]
    j=random(0, length[solution]);
    bitFlip(currentSolution, j); // flips jth bit of solution producing s' from s
    tmpEval = evaluate(currentSolution);
    if (tmpEval < bestEval)      // 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]



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: 1 0 0 0:1

iteration#1: 0 1 0 0:1

iteration#2: 0 0 1 0:0

iteration#3: 0 0 0 1:0 ✓

```
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.)

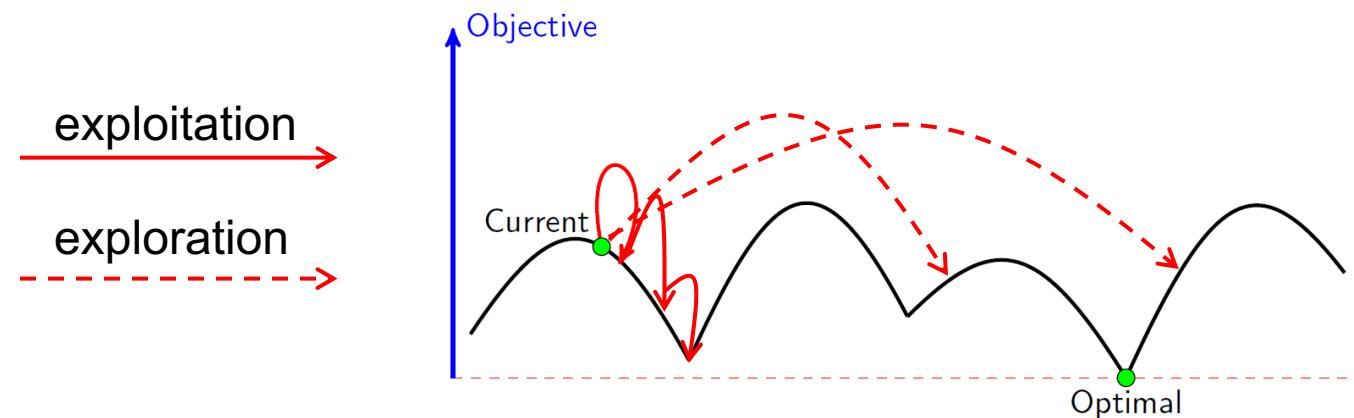


- If the target objective is known (e.g., minimum value for f_2 is known which is 0), then the search can be stopped when that target objective value is achieved.
- Hill climbing could be applied repeatedly until a termination criterion is satisfied (e.g. maximum number of evaluations is exceeded which 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.





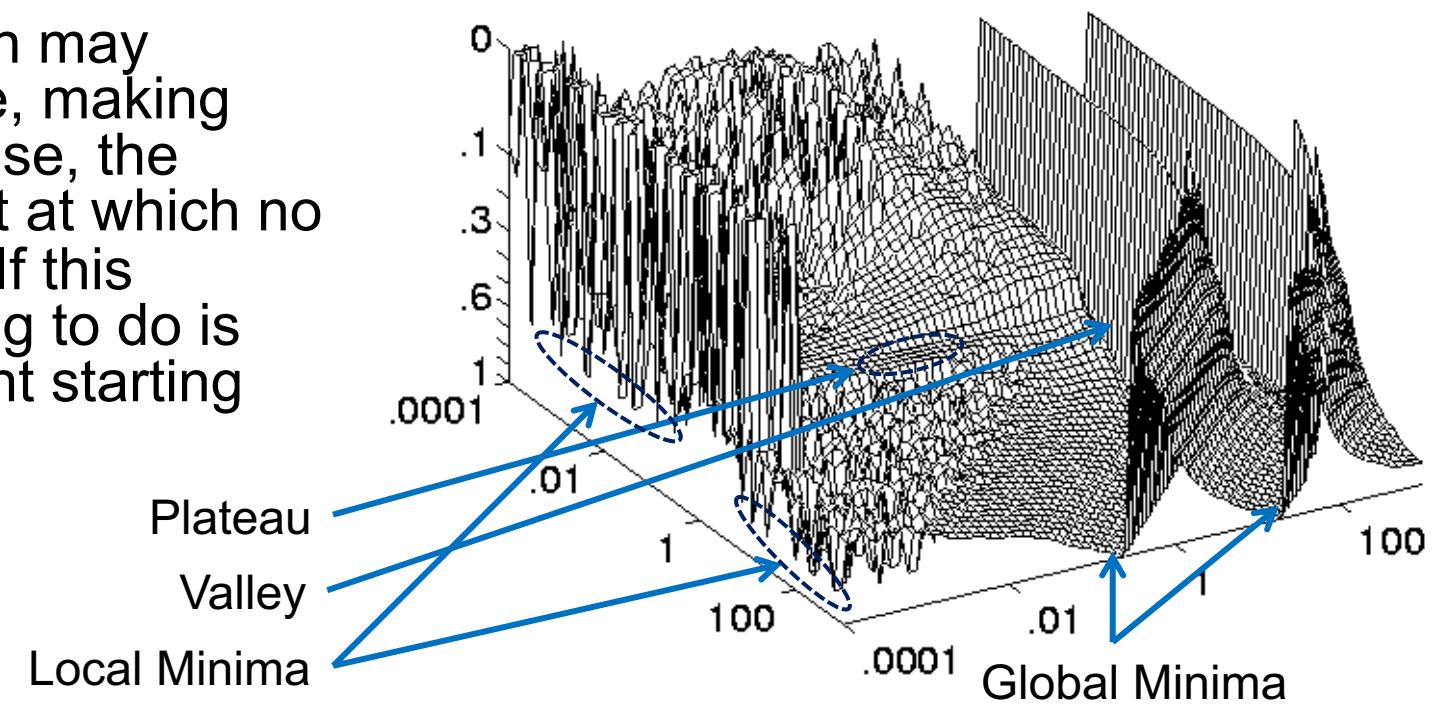
Strengths of Hill Climbing

- 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

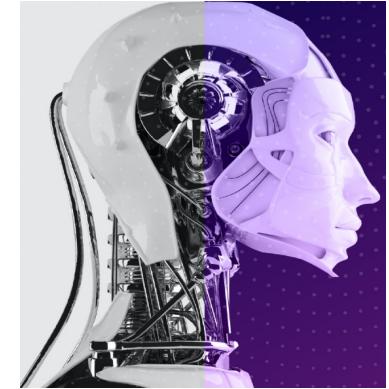
Home Exercise (Forum Discussion is Open)



- 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





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

Instance	Algorithm A				Algorithm B				Algorithm C			
	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

run/trial	Algorithm A				Algorithm B				Algorithm C				
	Instance	avg.	std.	med.	min.	avg.	std.	med.	min.	avg.	std.	med.	min.
1	Inst1	0.9	0.7	1	0	5.7	3.3	6	1	11.6	4.9	12	3
2		1	8	20									
3		1	6	11									
4		0	2	15									
5		1	1	12									
6		0	3	11									
7		2	4	13									
8		0	8	3									
9		1	6	19									
10		1	1	15									
11		0	8	3									
12		1	8	11									
13		2	11	9									
14		1	3	14									
15		2	11	12									
		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.

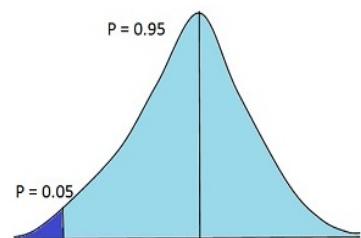


Which Stochastic Search Algorithm Performs Better for Solving Problem X?

- 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 $X \geq Y$ ($X \leq Y$) 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>)

run/trial	Algorithm A			Algorithm B			Algorithm C			
	Instance	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.
1	Inst1	0.9	0.7	1	0	>	5.7	3.3	6	1
2		1	8	20						
3		1	6	11						
4		0	2	15						
5		1	1	12						
6		0	3	11						
7		2	4	13						
8		0	8	3						
9		1	6	19						
10		1	1	15						
11		0	8	3						
12		1	8	11						
13		2	11	9						
14		1	3	14						
15		2	11	12						
		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



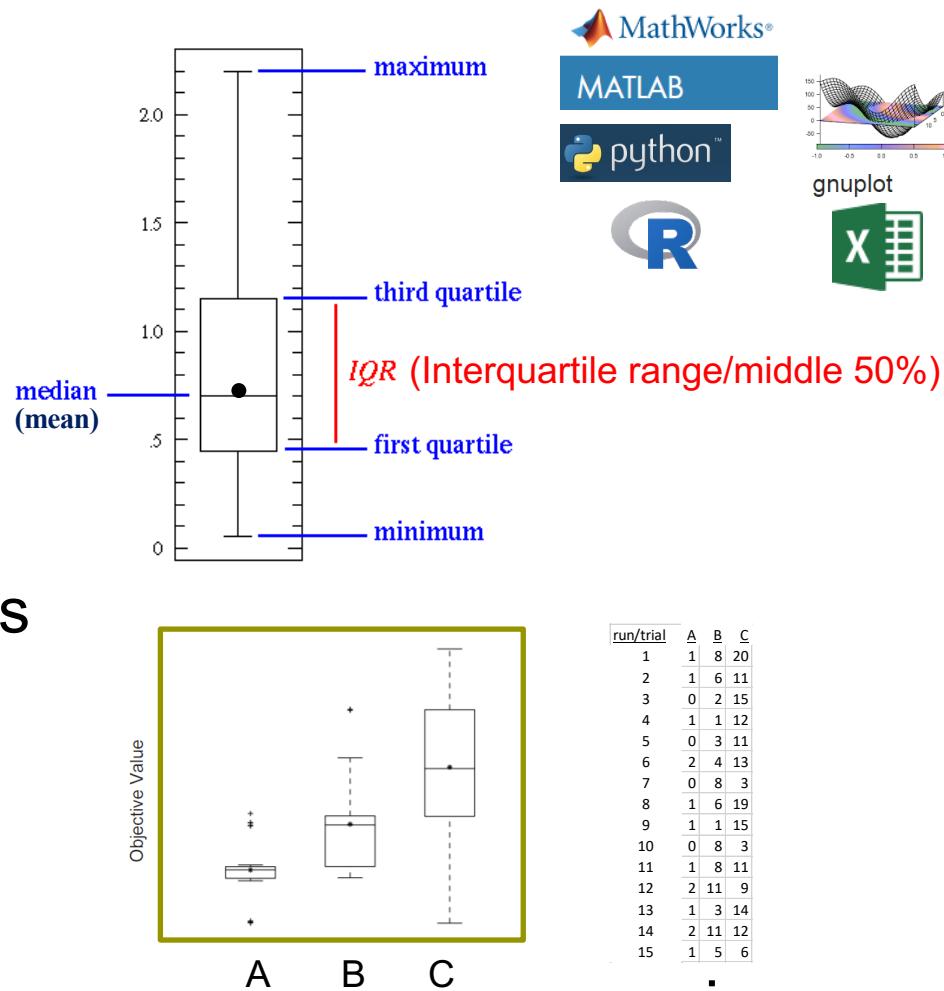
Which Stochastic Search Algorithm Performs Better for Solving Problem X?

Boxplots

- Boxplots illustrates groups of numerical data through their quartiles.

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

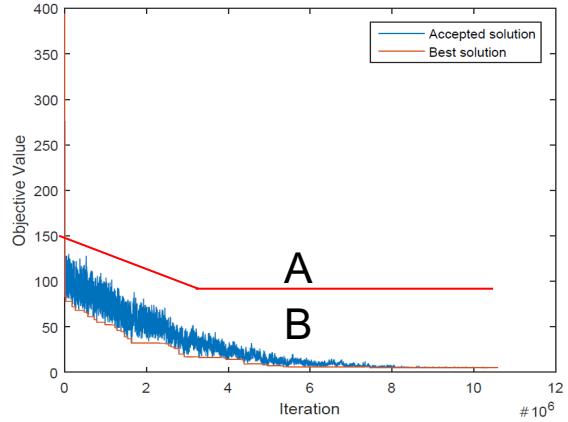
- Example: boxplot of objective values obtained from 30 runs on the instance Inst1 from 3 algorithms
 - ▶ Outliers may be plotted as individual points.
 - ▶ **A stronger conclusion for Inst1 – next slide**



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/



COM2001/2011 Artificial Intelligence Methods

Lecture 3 Metaheuristics

Prof Ender Özcan

Computer Science, Office: C86

Ender.Ozcan@nottingham.ac.uk

www.nottingham.ac.uk/~pszeo/



1. Performance Comparison of Stochastic Search Algorithms



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

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
⋮	⋮	⋮	⋮

Instance	Algorithm A				Algorithm B				Algorithm C			
	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

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

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

Instance	Algorithm A				Algorithm B				Algorithm C			
	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

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

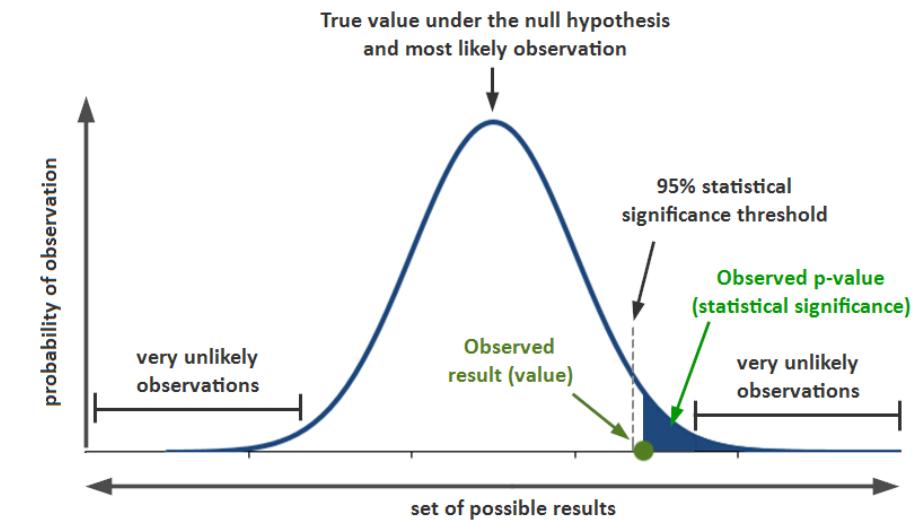
Any comment for 1 instance is valid for 1 problem instance.

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.

Statistical tests, null hypothesis and p-values



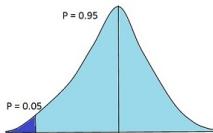
- The null hypothesis states the results are due to chance and are not significant in terms of supporting the idea being investigated.
- A p-value, or probability value, is a number describing how likely it is that your data would have occurred by random chance (i.e. that the null hypothesis is true).





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 $X \geq Y$ ($X \leq Y$) 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>)



Instance	Algorithm A				Algorithm B				Algorithm C					
	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



Statistical Tests

Nonparametric statistical tests

Type of comparison	Procedures
Pairwise comparisons	Sign test Wilcoxon test
Multiple comparisons ($1 \times N$)	Multiple sign test Friedman test Friedman Aligned ranks Quade test Contrast Estimation
Multiple comparisons ($N \times N$)	Friedman test

Nonparametric statistics **are not based on any assumptions**, such as the sample following a specific distribution, and measures the central tendency with the median value

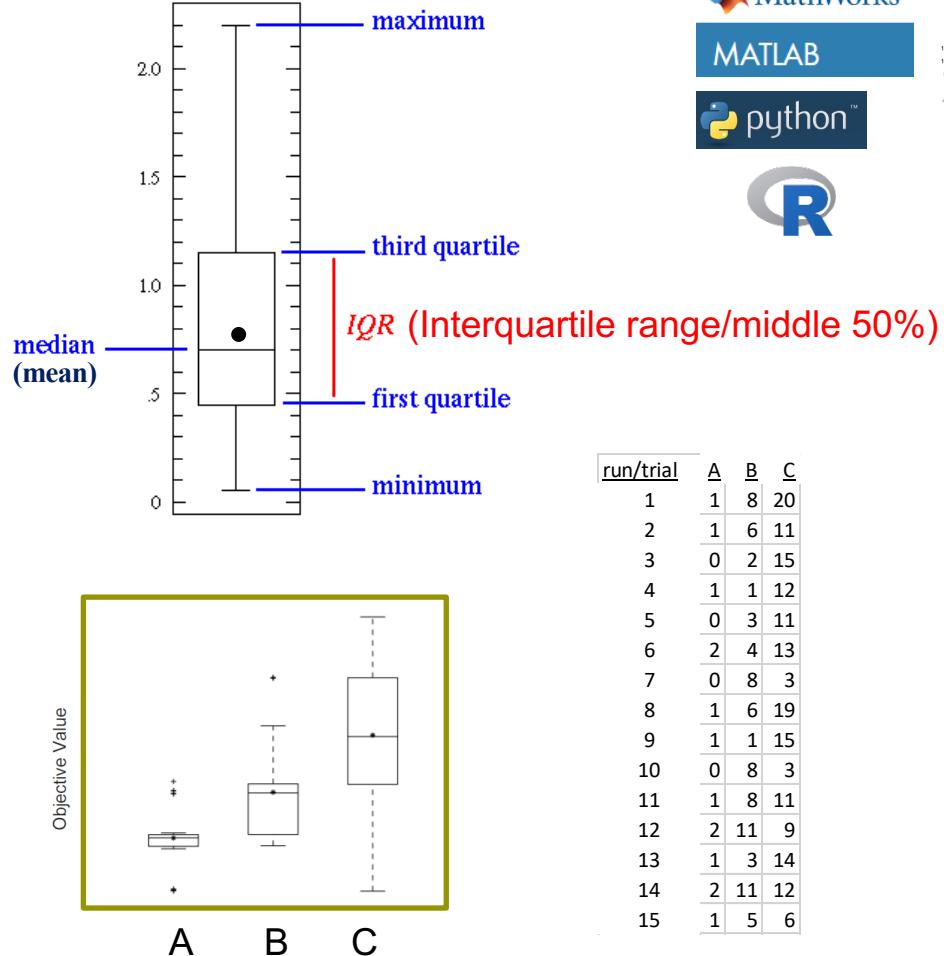
Post-hoc procedures

Type of comparison	Procedures
Multiple comparisons ($1 \times N$)	Bonferroni Holm Hochberg Hommel Holland Rom Finner Li
Multiple comparisons ($N \times N$)	Nemenyi Holm Shaffer Bergmann

J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, *Swarm and Evolutionary Computation*, vol. 1, issue 1, pp. 3-18, 2011. [[PDF](#)]

Which Stochastic Search Algorithm Performs Better for Solving Problem X? Boxplots

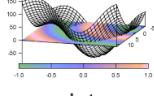
- Boxplots illustrates groups of numerical data through their quartiles.
BoxPlotR: <http://shiny.chemgrid.org/boxplotr/>
- Example: boxplot of objective values obtained from 30 runs on the instance Inst1 from 3 algorithms
 - ▶ Outliers may be plotted as individual points.
 - ▶ **A stronger conclusion for Inst1 – next slide**



MathWorks®

MATLAB

python™



gnuplot

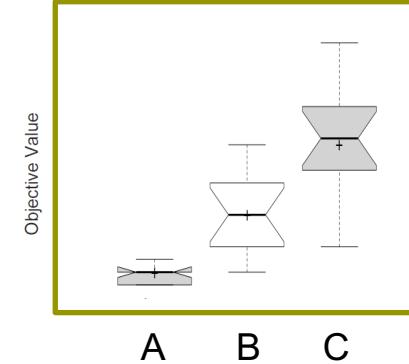
R



Which Stochastic Search Algorithm Performs Better for Solving Problem X? Notched Boxplots



- The notched boxplot allows you to evaluate confidence intervals (by default 95% confidence interval) for the medians of each boxplot.
- Since the notches in the boxplots A vs B, A vs C and B vs C do not overlap, you can conclude that with 95% confidence, that the true medians do differ between each pair of those algorithms on Inst1:
A performs significantly better than B as well as C, and B performs significantly better than C.





Which Stochastic Search Algorithm Performs Better for Solving Problem X?

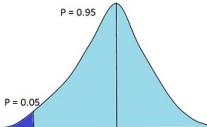
- Algorithms are run on multiple instances from Problem X – now even stronger conclusions can be made, e.g.,:
 - Algorithm A performs the best on (this benchmark/dataset) all problem X instances based on the mean/median objective values achieved over 30 trials.
 - Algorithm C is the worst performing approach based on all metrics.
 - Algorithm B provides the same best solution for 3 instances as Algorithm A.

Instance	Algorithm A				Algorithm B				Algorithm C			
	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
Inst2	3.1	3.9	2	1	21.3	13	12	3	44.9	9.8	30	18
Inst3	0.7	0.5	1	0	7.1	7.7	3	0	26.3	14	13	1
Inst4	1.7	1	1	1	5.7	4.3	3	1	20	4.6	15	12
Inst5	7.6	0.9	7	7	10.4	1.5	8	7	15.4	1.7	14	13



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 $X \geq Y$ ($X \leq Y$) 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>)



Instance	Algorithm A				Algorithm B				Algorithm C					
	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
Inst2	3.1	3.9	2	1	>	21.3	13	12	3	>	44.9	9.8	30	18
Inst3	0.7	0.5	1	0	>	7.1	7.7	3	0	>	26.3	14	13	1
Inst4	1.7	1	1	1	>	5.7	4.3	3	1	>	20	4.6	15	12
Inst5	7.6	0.9	7	7	>	10.4	1.5	8	7	>	15.4	1.7	14	13

- **Important:** Always repeat the experiments more than or equal to 30 times for all instances for a meaningful statistical comparison



Which Stochastic Search Algorithm Performs Better for Solving Problem X?

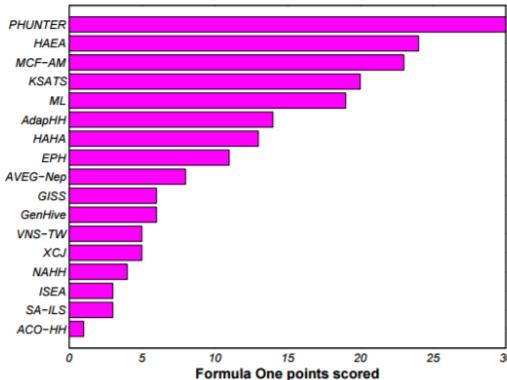
- Algorithms are run on multiple instances of Problem X –conclusions can be strengthen further, e.g.,:
 - Algorithm A is better than Algorithm B, while Algorithm B is better than C on (this benchmark/dataset or) all problem X instances on average and all those performance differences between the pair of algorithms are statistically significant within a confidence interval of 95%.

Instance	Algorithm A				Algorithm B				Algorithm C					
	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
Inst2	3.1	3.9	2	1	>	21.3	13	12	3	>	44.9	9.8	30	18
Inst3	0.7	0.5	1	0	>	7.1	7.7	3	0	>	26.3	14	13	1
Inst4	1.7	1	1	1	>	5.7	4.3	3	1	>	20	4.6	15	12
Inst5	7.6	0.9	7	7	>	10.4	1.5	8	7	>	15.4	1.7	14	13

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
- **Ranking**

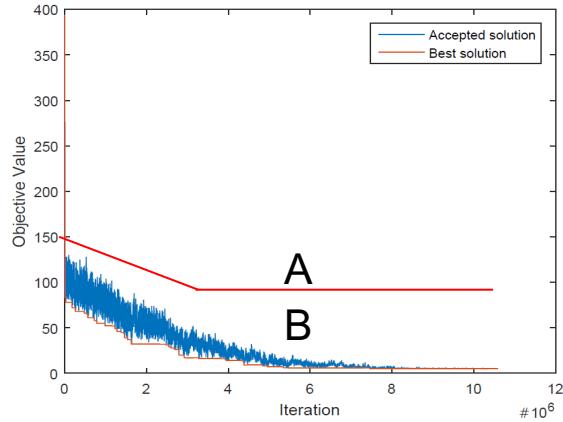


(higher score is better)

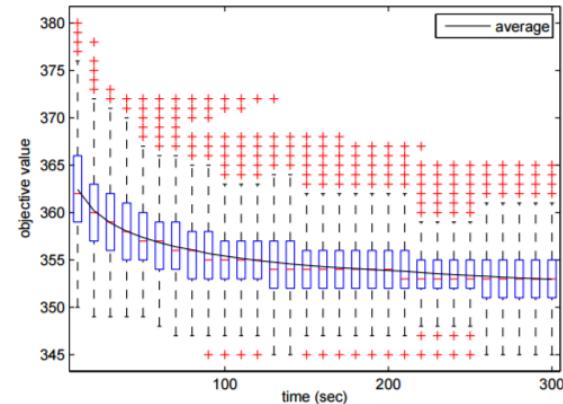
Formula 1 vs normalised objective value ranking of algorithms on multiple instances

$$f_{norm}(s) = \frac{f(s) - f(s_{best})}{f(s_{worst}) - f(s_{best})}$$

	Cross-domain
IE	17.67
AILLA	15.91
TA	20.04
GD	13.15
AILTA	19.05
NA	29.61
SA	10.10
SARH	10.22



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



Box plot of objective
values collected at each
time step (every 10 sec.)
from multiple runs of the
Algorithm A on an
instance



2. Metaheuristics



What is a Metaheuristic?

A **metaheuristic** is a high-level problem independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms

K. Sørensen and F. Glover. Metaheuristics. In S.I. Gass and M. Fu, editors, Encyclopedia of Operations Research and Management Science, pp 960–970. Springer, New York, 2013.

[\[PDF\]](#)

Metaheuristics



Local Search	<ul style="list-style-type: none">• [Kirkpatrick, 1983] Simulated Annealing (SA)• [Glover, 1986] Tabu Search (TS)• [Voudouris, 1997] Guided Local Search (GLS)• [Stutzle, 1999] Iterated Local Search (ILS)• [Mladenovic, 1999] Variable Neighborhood Search (VNS)
Population-based	<ul style="list-style-type: none">• [Holland, 1975] Genetic Algorithm (GA)• [Smith, 1980] Genetic Programming (GP)• [Goldberg, 1989] Genetic and Evolutionary Computation (EC)• [Moscato, 1989] Memetic Algorithm (MA)• [Kennedy and Eberhart, 1995] Particle Swarm Optimisation (PSO)
Constructive	<ul style="list-style-type: none">• [Dorigo, 1992] Ant Colony Optimisation (ACO)• [Resende, 1995] Greedy Randomized Adaptive Search Procedure (GRASP)

Main Components of a Metaheuristic Search Method (r.v.)



- ❑ Representation of candidate solutions
- ❑ Evaluation function
- ❑ Initialisation: E.g., initial candidate solution may be chosen
 - randomly □ use a constructive heuristic
 - according to some regular pattern
 - based on other information (e.g. results of a prior search), and more
- ❑ Neighbourhood relation (move operators)
 - Search process (guideline)
 - Stopping conditions
 - *Mechanism for escaping from local optima*

Mechanisms for Escaping from Local Optima I



- Search process (guideline)
- Stopping conditions
- Mechanism for escaping from local optima

- Iterate with different solutions, or restart (re-initialise search whenever a local optimum is encountered).
 - ▶ Initialisation could be costly
 - ▶ E.g., Iterated Local Search, GRASP
- Change the search landscape
 - ▶ Change the objective function (E.g., Guided Local Search)
 - ▶ Use (mix) different neighbourhoods (E.g., Variable Neighbourhood Search, Hyper-heuristics)

Mechanisms for Escaping from Local Optima II



- Search process (guideline)
- Stopping conditions
- Mechanism for escaping from local optima

- Use Memory (e.g., tabu search)
- Accept non-improving moves: allow search using candidate solutions with equal or worse evaluation function value than the one in hand
 - ▶ Could lead to long walks on plateaus (neutral regions) during the search process, potentially causing cycles – visiting of the same states
- None of the mechanisms is guaranteed to always escape effectively from local optima

Termination Criteria (Stopping Conditions) – Examples



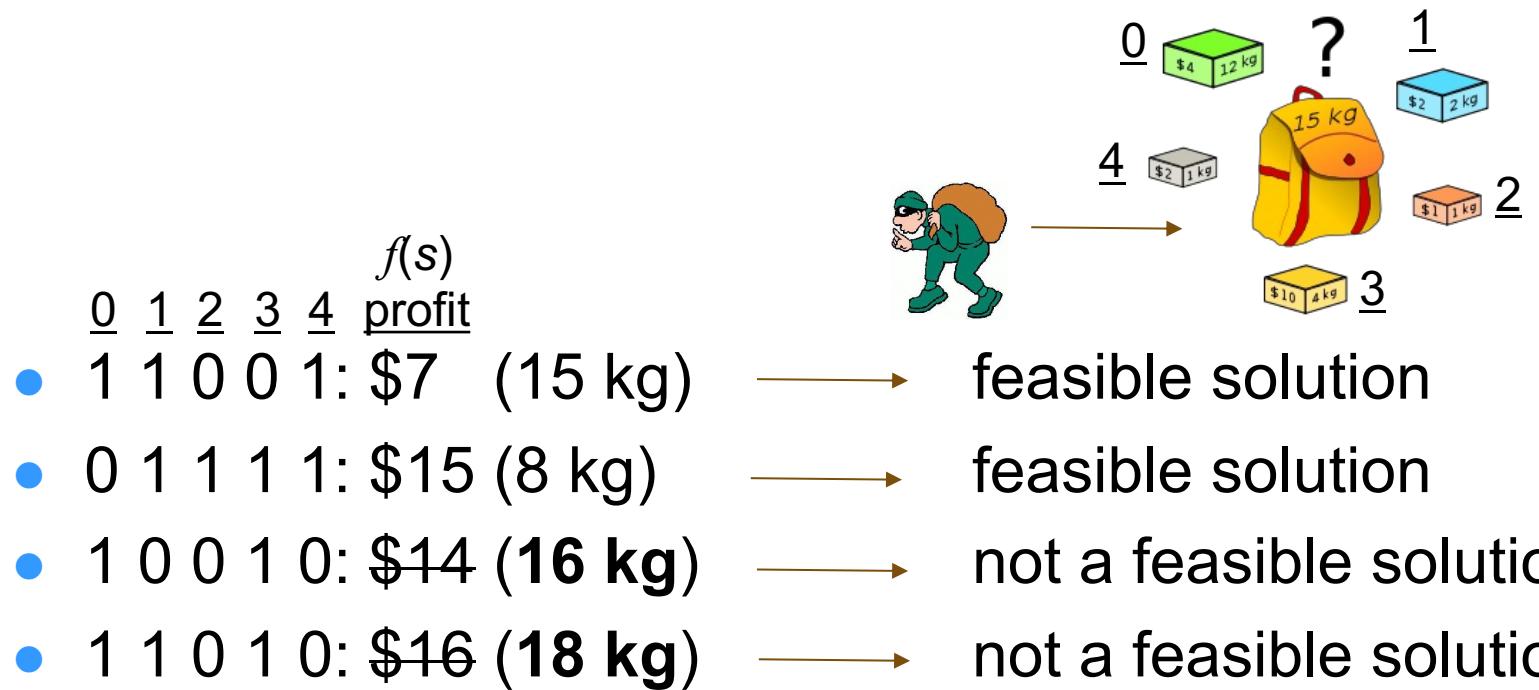
- Stop if
 - ▶ a fixed maximum number of iterations, or moves, objective function evaluations), or a fixed amount of CPU time is exceeded.
 - ▶ consecutive number of iterations since the last improvement in the best objective function value is larger than a specified number.
 - ▶ evidence can be given than an optimum solution has been obtained. (i.e. optimum objective value is known)
 - ▶ no feasible solution can be obtained for a fixed number of steps/time. (a solution is *feasible* if it satisfies all constraints in an optimisation problem)

- Search process (guideline)
- Stopping conditions
- *Mechanism for escaping from local optima*

Feasibility Example: 0/1 Knapsack Problem

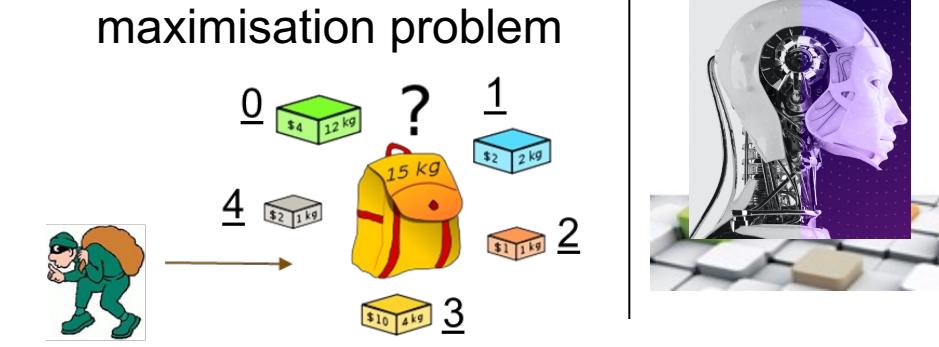


- Fill the knapsack with as much value in goods as possible (i.e., maximise “profit”) **without exceeding the capacity** (as a constraint) – which items to take?



How to Deal with Infeasible Solutions

- Use a problem domain specific repair operator
 - ▶ E.g. randomly flip a bit to 0 until the solution in hand feasible: 1 1 0 1 0: \$16 (**18 kg**) → 1 0 0 1 0: \$14 (**16 kg**) → 1 0 0 0 0 \$4 (12 kg)
- Penalise **each constraint violation** for the infeasible solutions such that they can't be better than the worst feasible solution for a given instance
 - ▶ Set a fixed (death) penalty value poorer than the worst, e.g.,
 $f'(s) = \text{if } s \text{ is infeasible, then } \min\{p_i, \forall i\}/2$ (that is \$1 for the example)
1 1 0 1 0: \$0.5 (**18 kg**), 1 0 0 1 0: \$0.5 (**16 kg**)
 - ▶ Distinguish the level of infeasibility of a solution with the penalty: e.g.,
 $f'(s) = \text{if } s \text{ is infeasible, then } \min\{p_i, \forall i\}/(2^*(\text{total_weight-capacity}))$
1 1 0 1 0: \$0.167 (**18 kg**), 1 0 0 1 0: \$0.5 (**16 kg**)



p_i is the profit from the i th item





3. Local Search Metaheuristics and Iterated Local Search

Stochastic Local Search – Single Point Based Iterative Search (Local Search Metaheuristics)



```
s0 ; // starting solution  
s* = initialise(s0); // e.g., improve s0 or use the same  
repeat  
    // generate a new solution  
    s' = makeMove(s*, memory); // choose a neighbour of s*  
    accept = moveAcceptance(s*, s', memory); // remember sbest  
    if (accept) s* = s'; // else reject new solution s'  
until (termination conditions are satisfied);
```

- Move Acceptance decides whether to accept or reject the new solution considering its evaluation/quality, $f(s')$
- Accepting non-improving moves could be used as a mechanism to escape from local optimum



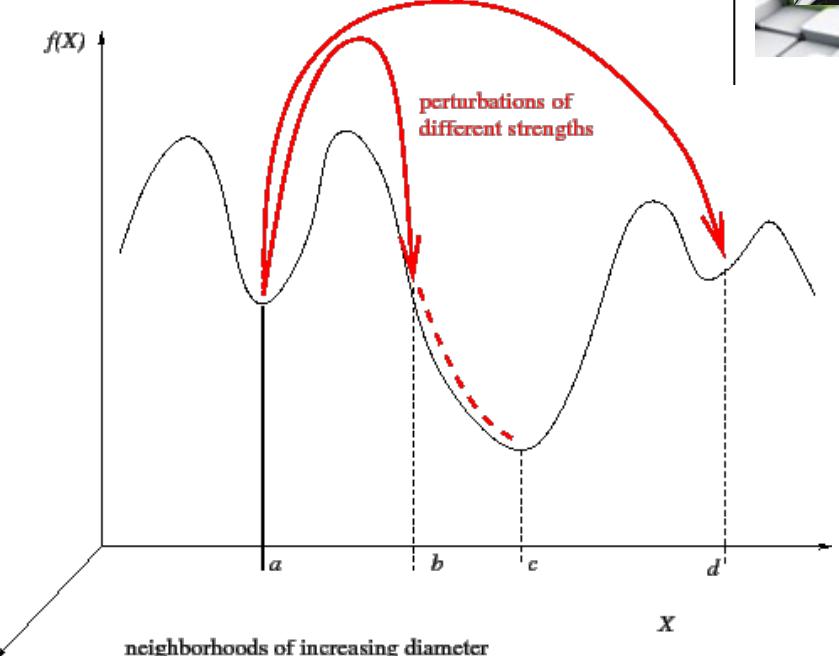
The Art of Searching

- Effective search techniques provide a mechanism to balance *exploration* and *exploitation*
 - ▶ Exploration aims to prevent stagnation of search process getting trapped at a local optimum
 - ▶ Exploitation aims to greedily increase solution quality or probability, e.g., by exploiting the evaluation function
- Aim is to design search algorithms/metaheuristics that can
 - ▶ escape local optima
 - ▶ balance exploration and exploitation
 - ▶ make the search independent from the initial configuration

Iterated Local Search (ILS)



```
s0 = GenerateInitialSolution()  
//random or construction heuristic  
s* = LocalSearch(s0) // hill climbing - not always used  
Repeat  
    s' = Perturbation(s*, memory)  
    // random move  
    s' = LocalSearch(s' )  
    // hill climbing  
    s* = AcceptanceCriterion(s*, s', memory) // remember sbest  
    // the conditions that the new local optimum  
    // must satisfy to replace the current solution  
Until (termination conditions are satisfied)  
return s*
```





Iterated Local Search II

- Based on visiting a sequence of locally optimal solutions by:
 - ▶ perturbing the current local optimum (exploration);
 - ▶ applying local search/hill climbing (exploitation) after starting from the modified solution.
- A perturbation phase might consist of one or more steps
- The perturbation strength is crucial
 - ▶ Too small (weak): may generate cycles
 - ▶ Too big (strong): good properties of the local optima are lost.



Iterated Local Search III

- Acceptance criteria
 - ▶ Extreme in terms of exploitation: accept only improving solutions
 - ▶ Extreme in terms of exploration: accept any solution
 - ▶ Other: deterministic (like threshold), probabilistic (like Simulated Annealing)
- Memory
 - ▶ Very simple use: restart search if for a number of iterations no improved solution is found

Exercise 1a – Designing an ILS Algorithm for MAX-SAT



- GenerateInitialSolution: Random
- Perturbation: A number of random bit flips (random walk)
- LocalSearch: Use 1-bit-flip neighbourhood, Steepest Descent Hill Climbing
- AcceptanceCriterion: accept *improving and equal moves* (non-worsening): accept s' if and only if $f(s') \leq f(s^*)$



Applying an Iterated Local Search Algorithm to a MAX-SAT Problem Instance – Exercise

- Problem:
 - ▶ $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Representation: Binary string
- Initialisation: random binary string
- Objective function: number of unsatisfied clauses
- Step 1: Initialise Solution
 - ▶ $s_0 \leftarrow \overset{a}{1} \overset{b}{0} \overset{c}{0} \overset{d}{1} \overset{e}{0} \overset{f}{0}$
 - ▶ $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
 - ▶ $c_0 \textcolor{red}{c}_1 \textcolor{red}{c}_2 c_3 c_4 \textcolor{red}{c}_5$
 - ▶ $f(s_0) = 3$
 - ▶ (No local search after initialisation)



$$s^* = s_0 = 100100; f(s_0) = 3$$

- Problem:
 - ▶ $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Representation: Binary string
- Initialisation: random binary string
- Objective function: number of unsatisfied clauses
- Perturbation Operator: Random bit flip
- Local Search Operator: Steepest (Gradient) Descent with IE (\leq) acceptance (SDHC)
- Move Acceptance: Improving or equal (\leq)
- Step 2: Iterated Local Search (Loop 1)
 - ▶ Random Bit Flip
 - $s' \leftarrow 1\textcolor{red}{1}0100$
 - Clause SAT: 012345
 - $f(s') = 4$
 - ▶ SDHC
 - $f(\overset{a}{\textcolor{red}{1}}\overset{b}{0}\overset{c}{1}\overset{d}{0}\overset{e}{1}\overset{f}{0}) = 012345 = 3$
 - $f(1\overset{a}{\textcolor{red}{0}}0100) = 012345 = 3$
 - $f(\underline{11\textcolor{red}{1}100}) = \underline{012345} = 1$
 - $f(1100\overset{a}{\textcolor{red}{1}}0) = 01\textcolor{red}{2345} = 3$
 - $f(1101\overset{a}{\textcolor{red}{1}}0) = 012\textcolor{red}{345} = 3$
 - $f(11010\overset{a}{\textcolor{red}{1}}) = 0123\textcolor{red}{45} = 3$
 - $s' \leftarrow 111100$
 - ▶ $1 < 3 (f(s') \leq f(s^*))$
 - $\therefore s^* \leftarrow s'$



$$s^* = s_1 = 111100; f(s_1) = 1$$

- Problem:
 - ▶ $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Step 2: Iterated Local Search (Loop 2)
 - ▶ Random Bit Flip
 - $s' \leftarrow 1\textcolor{red}{0}1100$
 - Clause SAT: 012345
 - $f(s') = 1$
 - ▶ SDHC
 - $\begin{matrix} a & b & c & d & e & f \\ \textcolor{red}{0} & 1 & 1 & 1 & 0 & 0 \end{matrix}$
 - $f(\textcolor{red}{0}01100) = \textcolor{red}{0}12345 = 2$
 - $f(1\textcolor{red}{1}1100) = 012345 = 1$
 - $f(10\textcolor{red}{0}100) = 012345 = 3$
 - $f(101\textcolor{red}{0}00) = 012345 = 0$
 - $f(1011\textcolor{red}{1}0) = 012345 = 1$
 - $f(10110\textcolor{red}{1}) = 012345 = 1$
 - $s' \leftarrow 101000$
 - ▶ $0 < 1 (f(s') \leq f(s^*))$
 - $\therefore s^* \leftarrow s'$
 - ▶ return $f(s^*) = 0;$

Example 1b – Designing Another ILS Algorithm for MAX-SAT



- GenerateInitialSolution: Random
- Perturbation: *intensityOfMutation* times random bit flips
- LocalSearch: Davis's Bit Hill Climbing for *depthOfSearch* (noOfSteps) times
- AcceptanceCriterion: accept *improving and equal moves* (non-worsening): accept s' if and only if $f(s') \leq f(s^*)$

Example 2 – Designing an ILS Algorithm for TSP



- GenerateInitialSolution: Nearest-neighbor
- Perturbation: a number of exchange moves
E.g.: $1 \text{ } \color{red}{2} \text{ } 3 \text{ } 4 \text{ } 5 \text{ } 6 \rightarrow 1 \text{ } \color{blue}{4} \text{ } 3 \text{ } \color{red}{2} \text{ } 5 \text{ } 6$, then $6 \text{ } \color{blue}{4} \text{ } 3 \text{ } \color{red}{2} \text{ } 5 \text{ } 1$
- LocalSearch: first improvement using insertion neighborhood moves
E.g.: $1 \text{ } \color{red}{2} \text{ } 3 \text{ } 4 \text{ } 5 \text{ } 6 \rightarrow 1 \text{ } 3 \text{ } \color{red}{2} \text{ } 4 \text{ } 5 \text{ } 6$, $1 \text{ } \color{red}{2} \text{ } 3 \text{ } 4 \text{ } 5 \text{ } 6$, $1 \text{ } 3 \text{ } 4 \text{ } \color{red}{2} \text{ } 5 \text{ } 6$, $1 \text{ } 3 \text{ } 4 \text{ } 5 \text{ } \color{red}{6}$,
 $1 \text{ } 3 \text{ } 4 \text{ } 5 \text{ } 6 \text{ } \color{red}{2}$, $2 \text{ } 1 \text{ } 3 \text{ } 4 \text{ } 5 \text{ } 6$
- AcceptanceCriterion: accept *improving moves only*: accept s' if and only if $f(s') < f(s^*)$



ILS – Some Guidelines I

- Initial solution should be to a large extent irrelevant for longer runs.
- The interactions among perturbation strength and acceptance criterion can be particularly important
 - ▶ it determines the relative balance of exploration and exploitation
 - ▶ large perturbations are only useful if they can be accepted.



ILS – Some Guidelines II

- Advanced acceptance criteria may take into account search history, e.g., by occasionally reverting to incumbent solution.
- Advanced ILS algorithms may change nature and/or **strength of perturbation** (e.g., number of bit flips) adaptively during search.
- Local search should be as effective and as fast as possible. Better local search generally leads to better ILS performance
- Choose a perturbation operator whose steps cannot be easily undone by the local search



4. Tabu Search



Tabu Search

- Proposed by Fred W. Glover in 1986 and formalised in 1989
- Uses history (memory structures) to escape from local minima, inspired by ideas from artificial intelligence in the late 1970s.
- Applies hill climbing/local search
 - ▶ Some solution elements/moves are regarded as tabu (forbidden)
 - ▶ Proceeds according to the assumption that there is no point in accepting a new (poor) solution unless it is to avoid a path already investigated

Glover F 1986 Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research. Vol. 13, pp. 533-549. [[PDF](#)]



Tabu Search Algorithm

Tabu Search (TS):

determine initial candidate solution s

While *termination criterion* is not satisfied:

determine set N' of non-tabu neighbours of s

choose a best improving candidate solution s' in N'

update tabu attributes based on s'

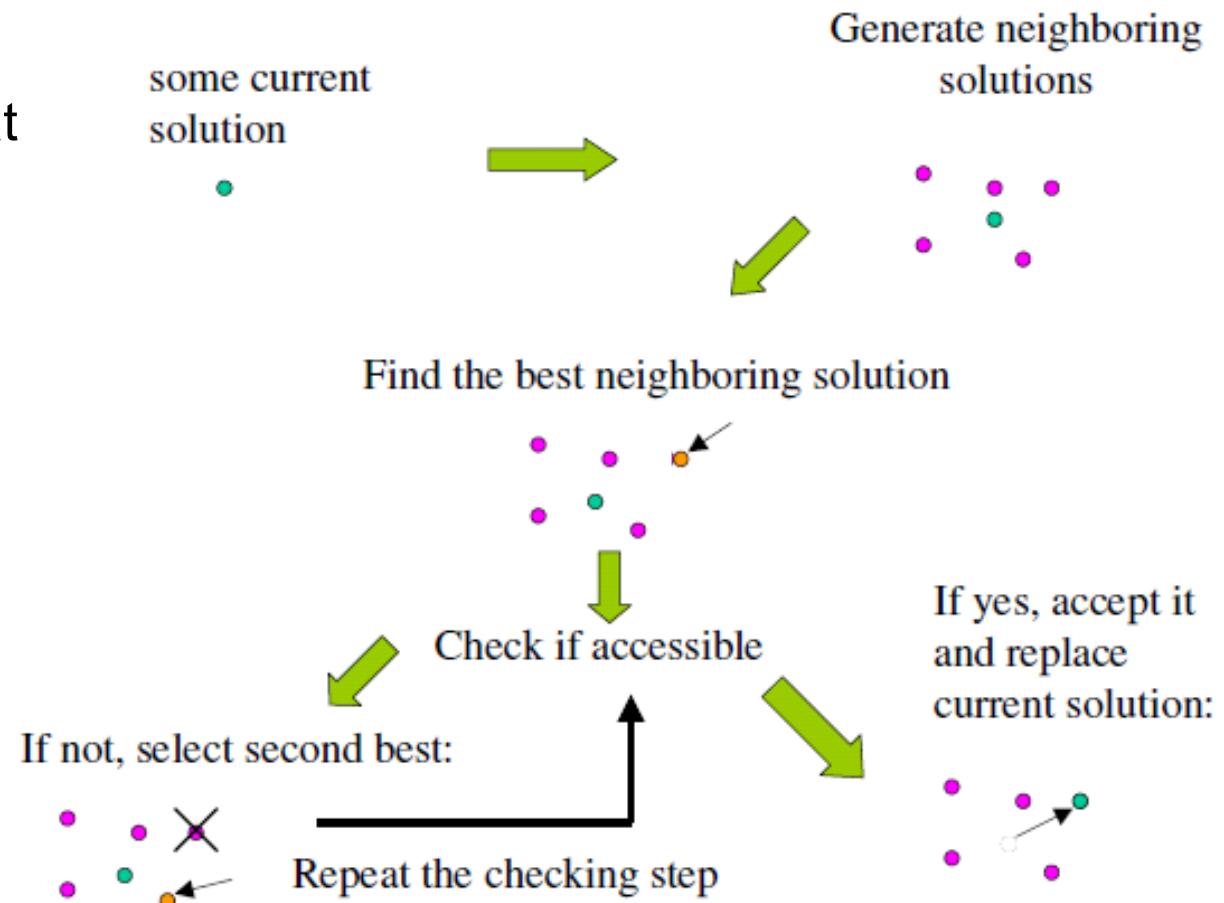
$s := s'$



Tabu Search – Overview

3 main components:

- Forbidding strategy: control what enters the tabu list
- Freeing strategy: control what exits the tabu list and when
- Short-term strategy: manage interplay between the forbidding strategy and freeing strategy to select trial solutions





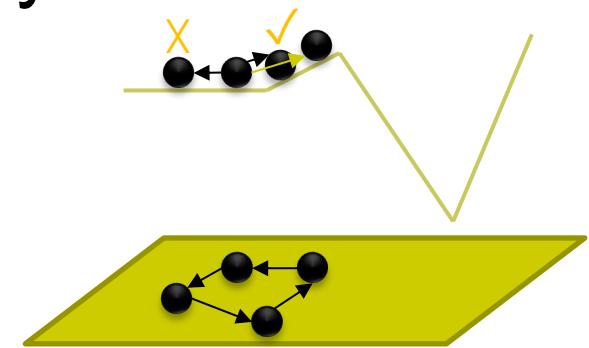
Tabu Search – Fundamentals I

- A stochastic local search algorithm which heavily relies on the use of an explicit memory of the search process
 - ▶ systematic use of memory to guide search process
 - ▶ memory typically contains only specific attributes of previously seen solutions
 - ▶ simple tabu search strategies exploit only short term memory
 - ▶ more complex tabu search strategies exploit long term memory



Tabu Search – Fundamentals II

- In each step, move to ‘non-tabu’ best neighbouring solution (*admissible neighbours*) although it may be worse than current one
- To avoid cycles, tabu search tries to avoid revisiting previously seen solutions
- Avoid storing complete solutions by basing the memory on attributes of recently seen solutions





Tabu Search – Fundamentals III

- Tabu solution attributes are often defined via local search moves
- *Tabu-list* contains moves which have been made in the recent past
 - ▶ *tabu list length* or *tabu tenure*
- Solutions which contain tabu attributes are forbidden for a certain number of iterations.
- Often, an additional *aspiration criterion* is used: this specifies conditions under which tabu status may be overridden (e.g., if considered step leads to improvement in incumbent solution).

Designing a Tabu Search Algorithm for MAX-SAT



- Initialisation: random – select an assignment randomly from set of all truth assignments
- Neighbourhood: 1-bit flip neighbourhood
- Memory: Associate tabu status (Boolean value) with each variable in given CNF.
 - ▶ variables are tabu iff they have been changed in the last T steps
 - for each variable x_i store the search step when its value was last changed denoted as t_i ; x_i is tabu iff $ci - t_i \leq T$, where ci is the current iteration/search step number.
 - use a queue of length T indicating the bit that was flipped in which step



An iteration of Tabu Search for MAX-SAT

- Problem:
 - ▶ $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Neighbourhood: 1-bit flip
- Assume $T = 2$
- Current iteration (ci) is 1207
- Current list content:
[1205, 59, 1206, 11, 0, 928]
 - ▶ This means that 1st bit/variable (a) was flipped in the 1205th step, second bit/variable (b) was flipped in the 59th step, and so on...

- $s^* = 101100; f(s^*) = 1$
- Consider all 1-bit flip neighbours which are not in the tabu list
 - (001100) X tabu (1207-1205) ≤ 2
 - $f(1\textcolor{red}{1}1100) = 0\textcolor{red}{1}2345 = 1$
 - (100100) X tabu (1207-1206) ≤ 2
 - $f(101\textcolor{red}{0}00) = 012345 = 0$
 - $f(1011\textcolor{red}{1}0) = 0\textcolor{red}{1}2345 = 1$
 - $f(10110\textcolor{red}{1}) = 012\textcolor{red}{3}45 = 1$
 - $s' \leftarrow 101000$
- Update list:
[1205, 59, 1206, 1207, 0, 928]



An iteration of Tabu Search for MAX-SAT

- Problem:
 - ▶ $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Neighbourhood: 1-bit flip
- Tabu tenure: 2
- Current iteration (ci) is 1207
- Current tabu list content: <1,3> (tail)
 - ▶ This means that 1st bit/variable (a) was flipped two steps ago (i.e., at step 1205), while 3rd bit/variable (c) was flipped in the immediately previous step (i.e., at step 1206).

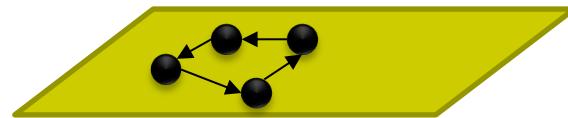
$$s^* = 101100; f(s^*) = 1$$

- Consider all 1-bit flip neighbours which are not in the tabu list
 - (001100) X tabu
 - $f(111100) = 012345 = 1$
 - (100100) X tabu
 - $f(101000) = 012345 = 0$
 - $f(101110) = 012345 = 1$
 - $f(101101) = 012345 = 1$
 - $s' \leftarrow 101000$
- Update tabu list: <3,4> (tail)



Practical Considerations

- Appropriate choice of tabu tenure critical for performance
- Tabu tenure: the length of time/number of steps t for which a move is forbidden
 - ▶ t too low- risk of cycling
 - ▶ t too high - may restrict the search too much
 - ▶ $t = 7$ has often been found sufficient to prevent cycling
 - ▶ $t = \sqrt{n}$
 - ▶ number of tabu moves: 5 – 9
- If a tabu move is smaller than the aspiration level then we accept the move (use of aspiration criteria to override tabu status)





5. Introduction to Scheduling



Scheduling

- Scheduling deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives. The resources and tasks in an organization can take many different forms.
- It is a decision-making process that is used on a regular basis in many manufacturing and services industries. Many real-world applications including
 - ▶ Project planning
 - ▶ Semiconductor manufacturing
 - ▶ Gate assignment at airports
 - ▶ Scheduling tasks in CPUs/parallel computers, ...



Scheduling – Framework and Notation

- In all the scheduling problems, considered number of jobs and machines are assumed to be finite.

jobs $j = 1, \dots, n$ **machines** $i = 1, \dots, m$

(i, j) processing step, or operation of job j on machine i

Classification of Scheduling Problems – Notation



- **Scheduling problem:** $\alpha \mid \beta \mid \gamma$
 - ▶ α machine characteristics (environments)
 - ▶ β processing/job characteristics
 - ▶ γ optimality criteria (objective to be minimised)



Sample Machine Characteristics (α)

Single-stage problem (vs. Multi-stage problem: O, F, J)

$\alpha \mid \beta \mid \gamma$

1 Single machine

P_m Identical machines in parallel

- m machines in parallel
- Job j requires a single operation and may be processed on any of the m machines

Q_m Machines in parallel with different speeds

R_m Unrelated machines in parallel machines have different speeds for different jobs



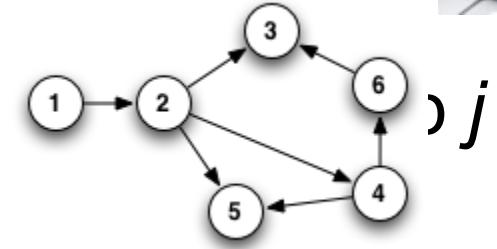
Sample Job Characteristics (β)

- The processing time may be different, or equal for all jobs, or even of unit length. All processing times are assumed to be integers.
- **Processing time** p_{ij} - processing time of job j on machine i (if a single machine then p_j)
- **Due date** d_j - committed shipping or completion (due) date of job j
- **Weight** w_j - importance of job j relative to the other jobs in the system



Sample Job Characteristics (β) II

- **Release date** r_j - earliest time at which can start its processing
- **Precedence** $prec$ – Precedence relations might be given for the jobs. If k precedes l , then starting time of l should be not earlier than completion time of k .
- **Sequence dependent setup times** s_{jk} - setup time between jobs j and k
- **Breakdowns** $brkdwn$ - machines are not continuously available





Sample Optimality Criteria (γ)

We define for each job j :

- ▶ C_{ij} **completion time** of the operation of job j on machine i
- ▶ C_j **time** when job j exits the system
- ▶ C_{\max} **makespan** is the time difference from the start (often, $t=0$) to finish when the last job exits the system
- ▶ $L_j = C_j - d_j$ **lateness** of job j
- ▶ $T_j = \max(C_j - d_j, 0)$ **tardiness** of job j
- ▶ $U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases}$ **unit penalty** of job j

See Lecture 4

Scheduling Notation – Examples



- $1 \mid \text{prec} \mid C_{\max}$
A single machine, general precedence constraints, minimising makespan (maximum completion time).
- $P3 \mid d_j, s_{jk} \mid \sum L_j$
3 identical machines, each job has a due date and sequence dependent setup times between jobs, minimising total lateness of jobs.
- $R \parallel \sum C_j$
variable number of unrelated parallel machines, no constraints, minimising total completion time.

See Lecture 4

A Single Machine Scheduling Problem



$$1 \mid d_j \mid \sum w_j T_j$$

- Given n jobs to be processed by a single machine, each job (j) with a *due date* (d_j) (i.e. hard deadline), processing time (p_j), and a weight (w_j) (i.e., job with the highest weight, say more important and so needs to finish on time), **find the optimal sequencing of jobs** producing the minimal weighted tardiness (T_j).
- Tardiness is 0 if a job completes on time ($C_j \leq d_j$), otherwise it is the time spent after the due date to completion ($C_j - d_j$)

$$T_j = \max(C_j - d_j, 0)$$

tardiness of job j

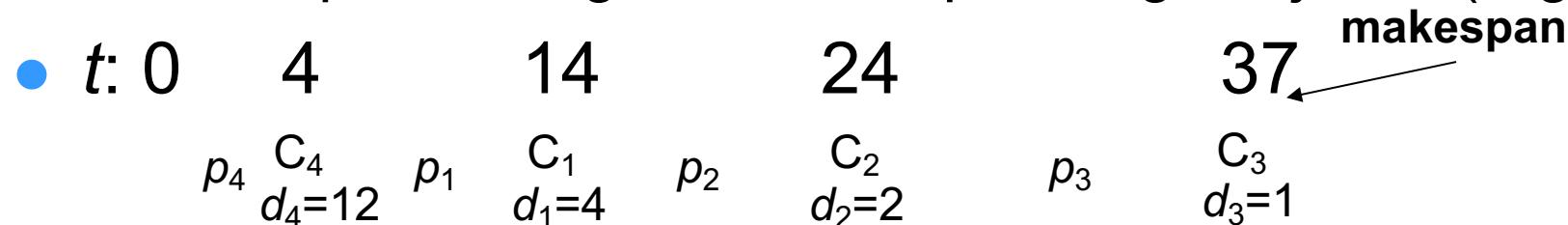
See Lecture 4

Example: Computing Weighted Tardiness



	jobs	1	2	3	4
1 d_j $\sum w_j T_j$	p_j	10	10	13	4
	d_j	4	2	1	12
	w_j	14	12	1	12

- What would be weighted tardiness of the solution which uses the shortest processing time for sequencing the jobs? (E.g., $<4, 1, 2, 3>$)



- $\sum w_j T_j = w_4 \max(C_4 - d_4, 0) + w_1 \max(C_1 - d_1, 0) + w_2 \max(C_2 - d_2, 0) + w_3 \max(C_3 - d_3, 0)$

$$\begin{aligned}
 &= 12 \max(-8, 0) + 14 \max(10, 0) + 12 \max(22, 0) + 1 \max(36, 0) \\
 &= 0 + 140 + 264 + 36 = 440
 \end{aligned}$$

See Lecture 4

Overall Summary



- Each metaheuristic has a mechanism for escaping from local optima
 - ▶ Balance between exploration and exploitation is important
 - ▶ Iterated Local Search enforces exploration and exploitation explicitly
 - ▶ Tabu Search uses flexible memory structures in conjunction with strategic restrictions and aspiration levels
- There are different types of metaheuristics embedding different components
 - ▶ The performance of a metaheuristic often varies based on the chosen design components – hence the need for empirical studies
- Scheduling contains many crucial NP hard real-world optimisation problems often dealt with in manufacturing/ production using heuristics/hyper-/metaheuristics.



University of
Nottingham

UK | CHINA | MALAYSIA



Q&A

ender.ozcan@nottingham.ac.uk

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

Move Acceptance in Local Search Metaheuristics and Parameter Setting Issues



Ender Özcan



Lecture 4



Computational
Optimisation &
Learning Lab



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA

Scheduling Notation – Examples



- $1 \mid \text{prec} \mid C_{\max}$
A single machine, general precedence constraints, minimising makespan (maximum completion time).
- $P3 \mid d_j, s_{jk} \mid \sum L_j$
3 identical machines, each job has a due date and sequence dependent setup times between jobs, minimising total lateness of jobs.
- $R \parallel \sum C_j$
variable number of unrelated parallel machines, no constraints, minimising total completion time.

Lecture 3

A Single Machine Scheduling Problem



$$1 \mid d_j \mid \sum w_j T_j$$

- Given n jobs to be processed by a single machine, each job (j) with a *due date* (d_j) (i.e. hard deadline), processing time (p_j), and a weight (w_j) (i.e., job with the highest weight, say more important and so needs to finish on time), **find the optimal sequencing of jobs** producing the minimal weighted tardiness (T_j).
- Tardiness is 0 if a job completes on time ($C_j \leq d_j$), otherwise it is the time spent after the due date to completion ($C_j - d_j$)

$$T_j = \max(C_j - d_j, 0)$$

tardiness of job j

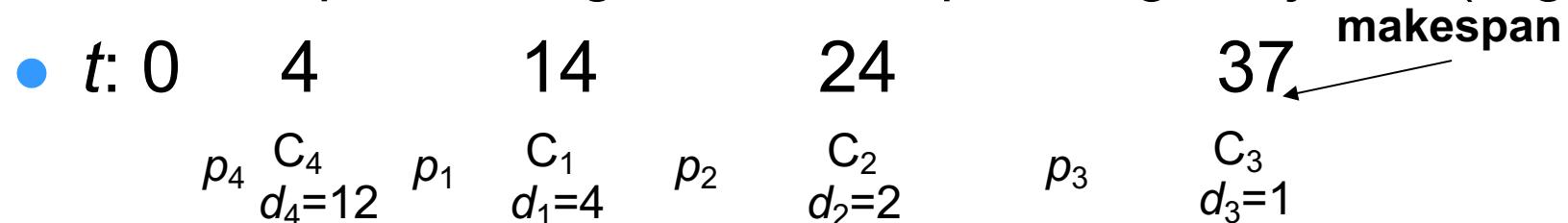
Lecture 3

Example: Computing Weighted Tardiness



1 d_j $\sum w_j T_j$	jobs	1	2	3	4
	p_j	10	10	13	4
	d_j	4	2	1	12
	w_j	14	12	1	12

- What would be weighted tardiness of the solution which uses the shortest processing time for sequencing the jobs? (E.g., $<4, 1, 2, 3>$)



- $\sum w_j T_j = w_4 \max(C_4 - d_4, 0) + w_1 \max(C_1 - d_1, 0) + w_2 \max(C_2 - d_2, 0) + w_3 \max(C_3 - d_3, 0)$
$$= 12 \max(-8, 0) + 14 \max(10, 0) + 12 \max(22, 0) + 1 \max(36, 0)$$
$$= 0 + 140 + 264 + 36 = 440$$



Lecture 3 – Overall Summary

- Each metaheuristic has a mechanism for escaping from local optima
 - ▶ Balance between exploration and exploitation is important
 - ▶ Iterated Local Search enforces exploration and exploitation explicitly
 - ▶ Tabu Search uses flexible memory structures in conjunction with strategic restrictions and aspiration levels
- There are different types of metaheuristics embedding different components
 - ▶ The performance of a metaheuristic often varies based on the chosen design components – hence the need for empirical studies
- Scheduling contains many crucial NP hard real-world optimisation problems ($\alpha \mid \beta \mid \gamma$) often dealt with in manufacturing/ production using heuristics/hyper-/metaheuristics.

1. Local Search Metaheuristics and Move Acceptance Methods



COM2001/2011: Artificial Intelligence Methods

Ender Özcan

Lecture 4



UNITED KINGDOM • CHINA • MALAYSIA

Stochastic Local Search – Single Point Based Iterative Search (Local Search Metaheuristics) - revisited

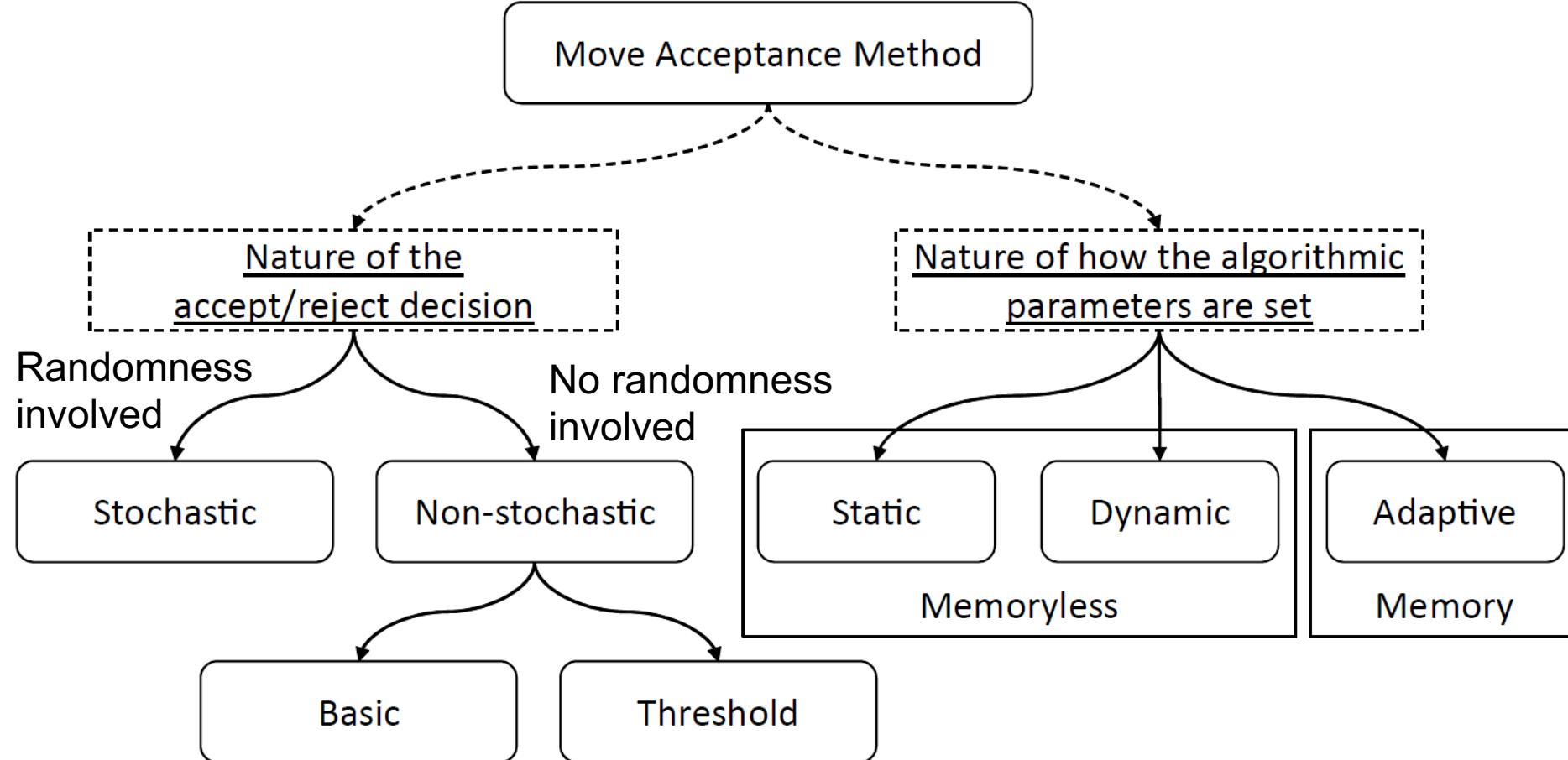


Algorithm 1: Outline of a Local Search Metaheuristic.

```
1  $s \leftarrow generateInitialSolution();$ 
2  $s_{best} \leftarrow s;$ 
3 while termination criteria not met do
4    $s' \leftarrow apply(h, s);$ 
5    $process_1();$ 
6    $s \leftarrow acceptRejectDecision(s, s');$ 
7   if  $f(s').isBetterThan(f(s_{best}))$  then
8      $| s_{best} \leftarrow s';$ 
9   end
10   $process_2();$ 
11 end
12 return  $s_{best};$ 
```

- Move Acceptance decides whether to accept or reject the new solution considering its evaluation/quality, $f(s')$
- Accepting non-improving moves could be used as a mechanism to escape from local optimum

Move Acceptance Methods – A Taxonomy



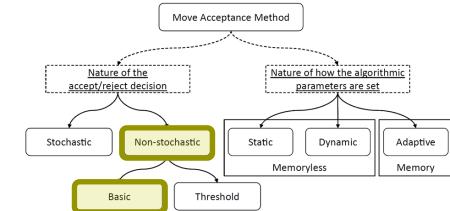
W. G. Jackson, E. Özcan and R. I. John, Move Acceptance in Local Search Metaheuristics for Cross-domain Search, Expert Systems with Applications, 109: 131-151, 2018 [[original PDF](#)]. [[PDF](#)]

Parameter Setting Mechanisms in Move Acceptance



- **Static**, either there is no parameter to set or parameters are set to a fixed value. E.g., $\text{IoM}=5$;
- **Dynamic**, parameter values vary with respect to time/iteration count. Given the same candidate and current solutions at the same current elapsed time or iteration count, the acceptance threshold or acceptance probability would be the same irrespective of search history.
E.g., $\text{IoM} = \text{round}(1 + (\text{iter}_{\text{current}} / \text{iter}_{\text{max}}) * 4)$;
- **Adaptive**, Given the same candidate and current solutions at the same current elapsed time or iteration count, the acceptance threshold or acceptance probability is not guaranteed to be the same as one or more components depend on search history. E.g., if for 100 steps best solution found in the stage cannot be improved, then $\text{IoM}++$, and after any improvement, reset $\text{IoM}=1$;

Non-stochastic Basic Move Acceptance Methods



- Reuse the objective values of previously encountered solutions for the accept/reject decisions
- Static**
 - All Moves: returns true regardless of $f(s')$
 - Improving Moves Only: $f(s') < f(s)$
 - Improving and Equal: $f(s') \leq f(s)$
- Dynamic:** None
- Adaptive**
 - Late Acceptance: compares the quality of the solution with that of the solution accepted/visited L iterations previously s_{t-L} , and accepts the move if and only if $f(s') \leq f(s_{t-L})$

Late Acceptance Algorithm – Pseudocode



```
Produce an initial solution  $s$ 
Calculate initial cost function  $C(s) \leftarrow f(s)$ 
Specify  $L_h$ 
For all  $k \in \{0 \dots L_h - 1\}$   $f_k := C(s)$ 
First iteration  $I := 0$ ;  $I_{idle} := 0$ 
Do until ( $I > 100000$ ) and ( $I_{idle} > I * 0.02$ )
    Construct a candidate solution  $s^*$ 
    Calculate a candidate cost function  $C(s^*)$ 
    If  $C(s^*) \geq C(s)$ 
        Then increment the idle iterations number  $I_{idle} := I_{idle} + 1$ 
        Else reset the idle iterations number  $I_{idle} := 0$ 
    Calculate the virtual beginning  $v := I \bmod L_h$ 
    If  $C(s^*) < f_v$  or  $C(s^*) \leq C(s)$ 
        Then accept the candidate  $s := s^*$ 
        Else reject the candidate  $s := s$ 
    If  $C(s) < f_v$ 
        Then update the fitness array  $f_v := C(s)$ 
    Increment the iteration number  $I := I + 1$ 
```

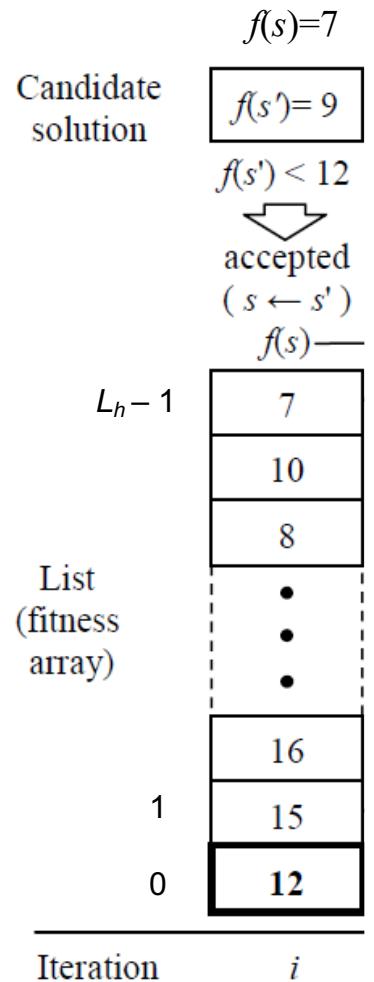
- **Initialisation:** Assign all elements of the list to be equal to the initial cost (objective value)
- List for the history of the objective values of the recent solutions is implemented as a circular queue

Edmund K. Burke, Yuri Bykov, [The late acceptance Hill-Climbing heuristic](#). EJOR 258(1): 70-78 (2017)

A Sample Run – Late Acceptance Algorithm

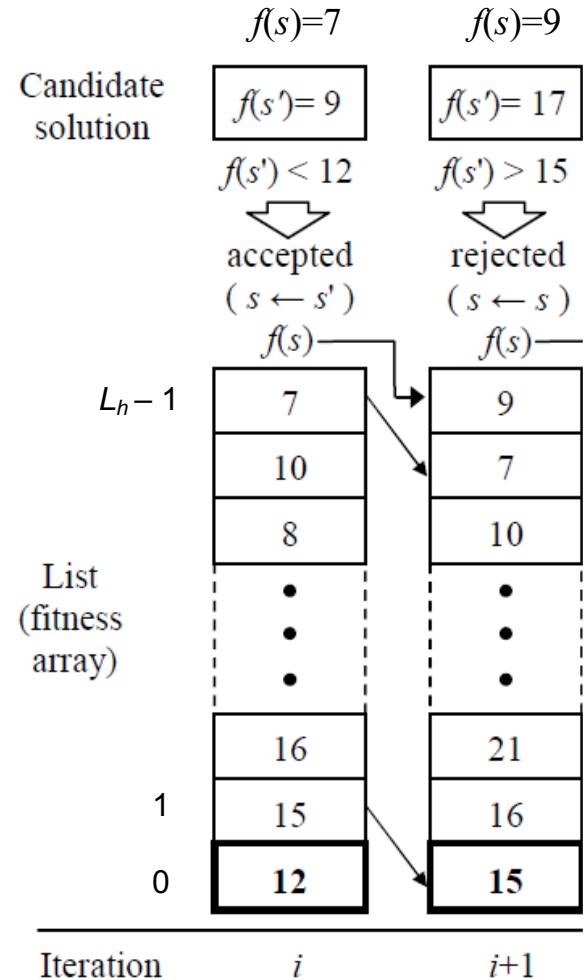


linear queue implementation



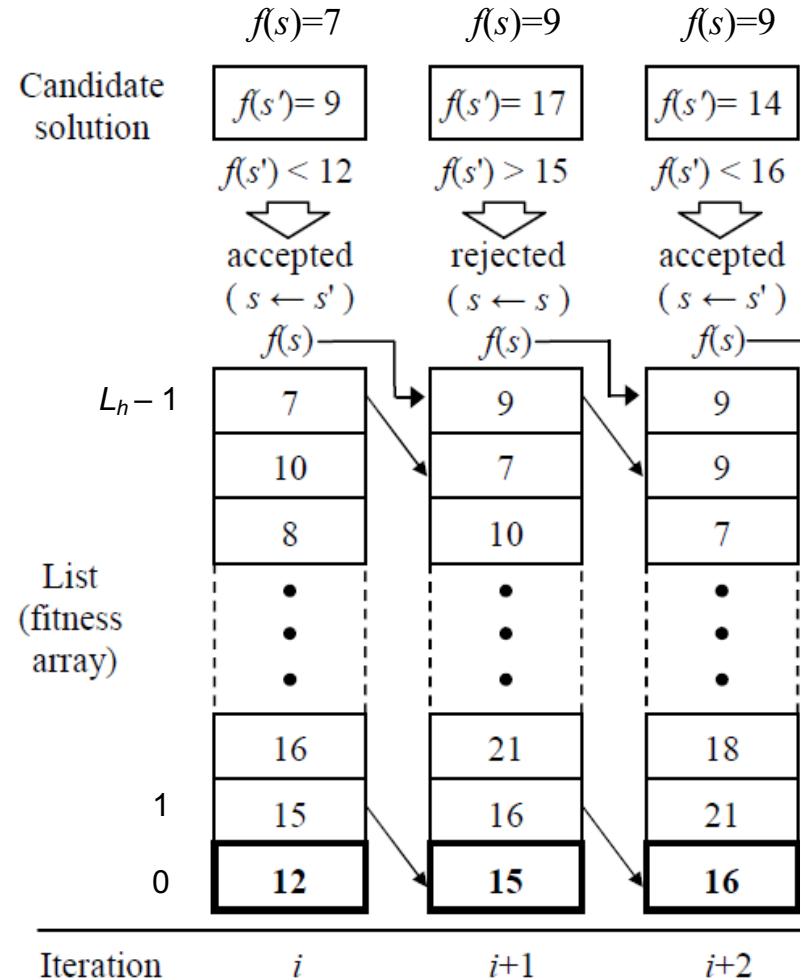
A Sample Run – Late Acceptance Algorithm

linear queue implementation



A Sample Run – Late Acceptance Algorithm

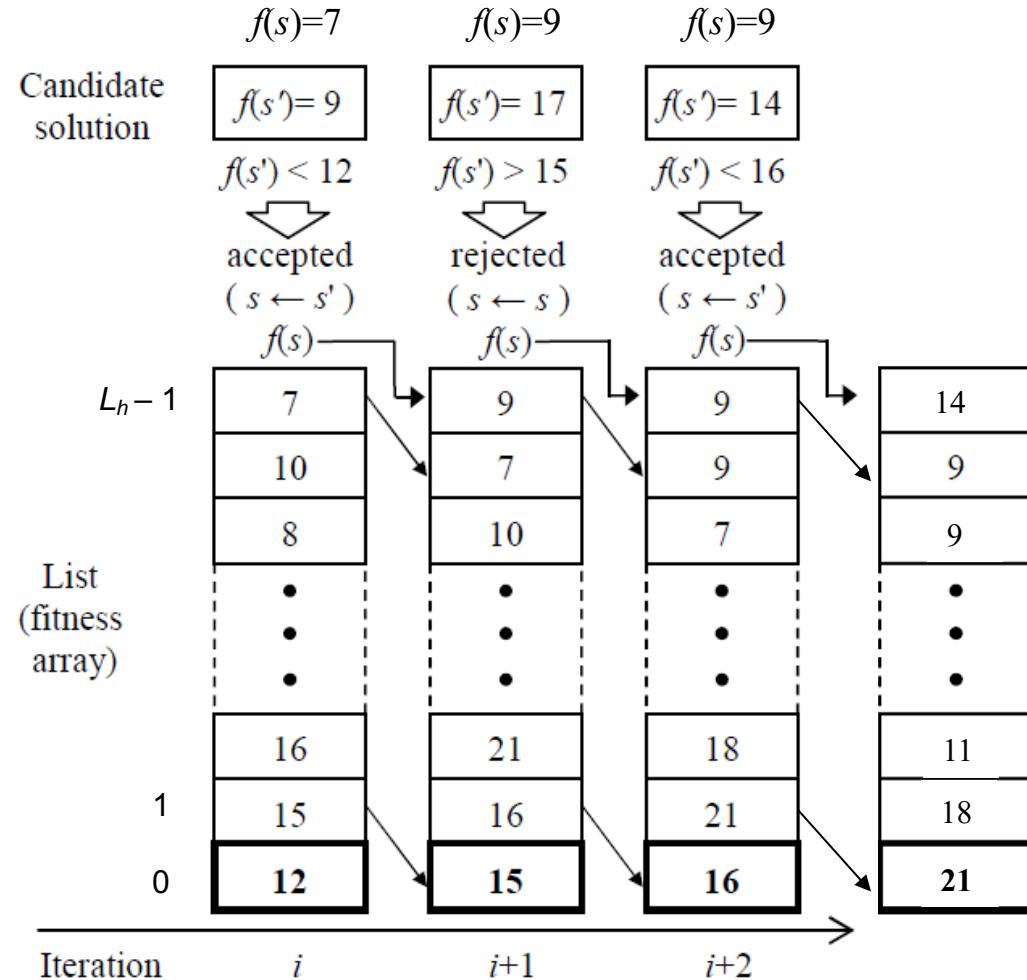
linear queue implementation



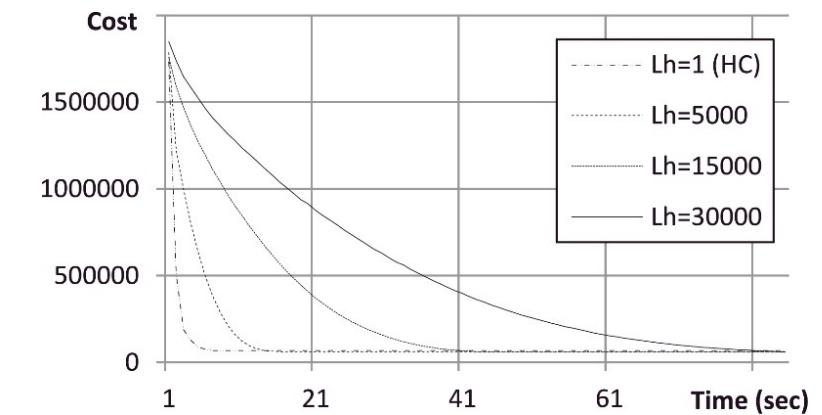
A Sample Run – Late Acceptance Algorithm



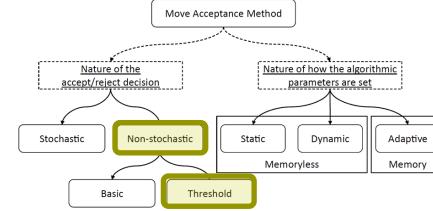
linear queue implementation



- Simple approach with a single parameter, yet the setting of the list length parameter influences the overall performance of the algorithm



Non-stochastic Threshold Move Acceptance – minimising F



```
s0 = generateInitialSolution();  
s, sbest = s0;  
// initialise other relevant parameters if there is any  
REPEAT  
    s' = makeMove(s, memory); // choose a neighbour of s*  
    threshold-ε = moveAcceptance-getThreshold(s, s', memory);  
    if ( f(s') ≤ threshold-ε) s = s'; // else reject new solution s'  
    sbest = updateBest(s, s'); // keep track of sbest  
UNTIL(termination conditions are satisfied);  
RETURN sbest
```

- Determine a threshold which is in the vicinity of a chosen solution quality, e.g. the quality of the best solution found so far or current solution, and accept all solutions below that threshold

Non-stochastic Threshold Move Acceptance – Examples



- **Static**
 - ▶ Accept a worsening solution if the worsening of the objective value is no worse than a fixed value
- **Dynamic**
 - ▶ Great Deluge
 - ▶ Flex Deluge

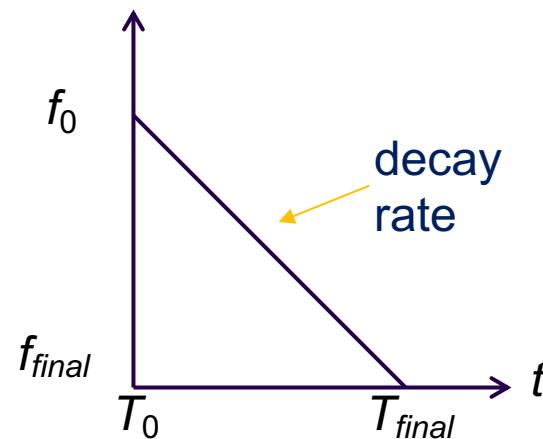
New Optimization Heuristics. [The Great Deluge Algorithm and the Record-to-Record Travel](#). Dueck, Gunter, Journal of Computational Physics, Volume 104, Issue 1, January 1993, Pages 86-92
- **Adaptive**
 - ▶ Record to record travel (RRT)
 - ▶ Extended Great Deluge
 - ▶ Modified Great Deluge



Great Deluge – minimisation

THE GREAT DELUGE ALGORITHM FOR MAXIMIZATION.

Example:



Choose an initial configuration
choose the “rain speed” DOWN>0
choose the initial WATER-LEVEL > 0
Opt: choose a new configuration which is a stochastic
small
perturbation of the old configuration
compute $E := \text{quality}(\text{new configuration})$ $\rightarrow (E \text{ is } f(\cdot))$
IF $E < \text{WATER-LEVEL}$
THEN old configuration := new configuration
WATER-LEVEL := WATER-LEVEL – DOWN \rightarrow (decay rate)
IF a long time no increase in quality or too many
iterations
THEN stop
GOTO Opt

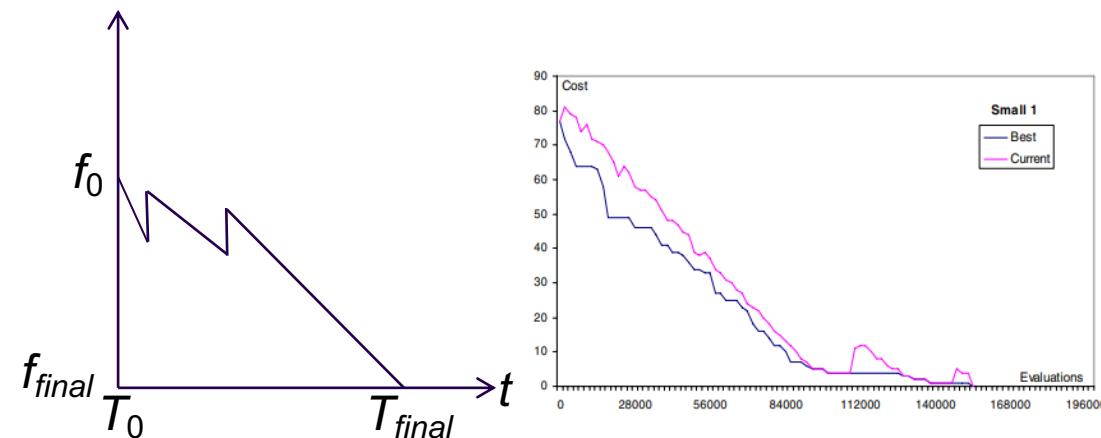
Gunter Dueck, Tobias Scheuer, [Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing](#), Journal of Computational Physics, Volume 90, Issue 1, September 1990, Pages 161-175



Extended Great Deluge – minimisation

- Feedback is received during the search and decay-rate is updated/reset accordingly whenever there is no improvement for a long time.

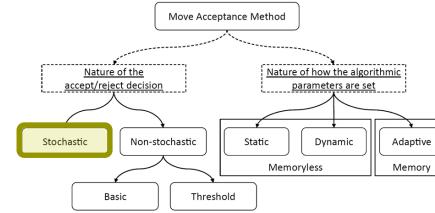
```
Set the initial solution  $s$  using a construction heuristic;  
Calculate initial cost function  $f(s)$   
Set Initial Boundary Level  $B_0 = f(s)$   
Set initial decay Rate  $\Delta B$  based on Cooling Parameter  
While stopping criteria not met do  
  Apply neighbourhood Heuristic  $S^*$  on  $S$   
  Calculate  $f(s^*)$   
  If  $f(s^*) \leq f(s)$  or  $(f(s^*) - f(s)) \leq \Delta B$  Then  
    Accept  $s = s^*$   
  Lower Boundary  $B = B - \Delta B$   
  If no improvement in given time  $T$  Then  
    Reset Boundary Level  $B_0 = f(s)$   
    Set new decay rate  $\Delta B$  based on Secondary  
    Cooling Parameter
```



[1] An Extended Great Deluge Approach to the Examination Timetabling Problem, B. McCollum, P.J. McMullan, A. J. Parkes, E.K. Burke, S. Abdullah [[PDF](#)]

[2] An Extended Implementation of the Great Deluge Algorithm for Course Timetabling, Paul McMullan [[PDF](#)]

Stochastic Move Acceptance



```
s0 = generateInitialSolution();  
s, sbest = s0;  
REPEAT  
    s' = makeMove(s, memory); // choose a neighbour of s  
    P = moveAcceptance-getAcceptanceProbability(s, s', memory);  
    r = getRandomValue(); // e.g., a uniform random value ∈ [0,1)  
    if ( f(s').isBetterThan( f(s) ) || (r < P) )  
        s = s'; // else reject new solution s'  
    sbest ← updateBest(s, s'); // keep track of sbest  
UNTIL (termination conditions are satisfied);  
RETURN sbest;
```



Stochastic Move Acceptance – Examples

- **Static**
 - ▶ E.g., Naive Acceptance: P is fixed, e.g. if improving $P=1.0$, else $P=0.5$
- **Dynamic**
 - ▶ E.g., Simulated Annealing: P changes in time with respect to the difference in the quality of current and previous solutions (see the next slides). Temperature parameter is changes dynamically.
- **Adaptive**
 - ▶ E.g., Simulated Annealing with reheating: P is modified via increasing temperature time to time causing partial restart – increasing the probability of acceptance of non-improving solutions



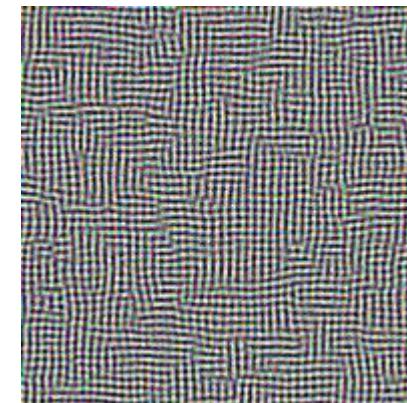
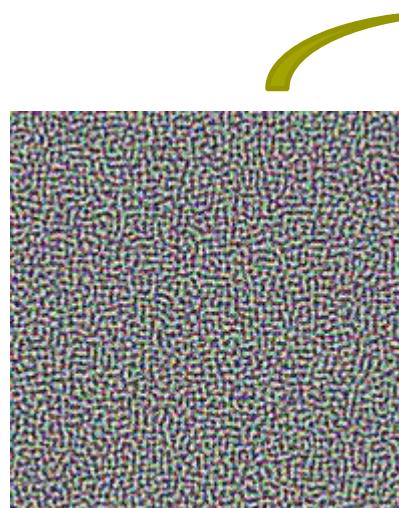
Simulated Annealing

- A stochastic local search algorithm inspired by the physical process of annealing (Kirkpatrick et al. 1983)
- Easy to implement
- Achieves good performance given sufficient running time
- Requires a good parameter setting for improved performance
- Has interesting theoretical properties (convergence), but these are of very limited practical relevance



Simulated Annealing – Analogy

- If a liquid material **cools and anneals** too quickly, then the material will solidify into a **sub-optimal** configuration.
- If the liquid material **cools slowly**, the crystals within the material will solidify **optimally into a state of minimum energy** (i.e. ground state). – This ground state corresponds to the minimum of the cost function in an optimisation problem.



Simulated Annealing Local Search Metaheuristic



```
INPUT:  $T_0, T_{final}$ 
 $s_0 \leftarrow generateInitialSolution();$ 
 $T \leftarrow T_0;$  // initialise temperature to  $T_0$ 
 $s_{best} \leftarrow s_0; s \leftarrow s_0;$  // set  $s$  and  $s_{best}$  to initial solution
REPEAT
     $s' \leftarrow perturbation(s);$  // choose a neighbouring solution of  $s$ 
     $\Delta = f(s') - f(s);$ 
     $r \leftarrow random \in [0,1];$  // get a uniform random number in the range [0,1)
    if( $\Delta < 0 \mid\mid r < P(\Delta, T)$ ) { // accept  $s'$  if solution is non-worsening or with Boltzmann probability
         $s \leftarrow s';$ 
    }
     $s_{best} \leftarrow updateBest();$  // keep track of best solution
     $T \leftarrow coolTemperature();$  // decrease the temperature according to cooling schedule
UNTIL (Termination conditions are satisfied);
Return  $s_{best};$ 
```



Accepting Moves using SA

- Improving moves are accepted
- Worsening moves are accepted using the Metropolis criterion at a given temperature T

$$\Delta = F(S_{new}) - F(S_{old}) \quad \text{minimise}\{ F \}$$

An inferior solution S_{new} would yield $\Delta > 0$

Accept it with a Boltzman probability of

$$P(\Delta, T) = e^{\frac{-\Delta}{T}}$$

$U(0,1)$ generates a random number in $[0,1]$

Accept if $U(0,1) < P(\Delta, T)$



Cooling/Annealing

$$P(\Delta, T) = e^{\frac{-\Delta}{T}}$$

- Temperature T is slowly decreased
 - ▶ T is initially high - many inferior moves are accepted
 - ▶ T is decreasing - inferior moves are nearly always rejected
- As the temperature T decreases, the probability of accepting worsening moves decreases.

$\Delta > 0$ inferior solution

$$-\Delta < 0 \quad T \searrow \quad -\frac{\Delta}{T} \searrow$$

$$e^{\frac{-\Delta}{T}} \searrow$$

T	Probability of acceptance $P(\Delta, T)$		
	$\Delta=1$	$\Delta=2$	$\Delta=3$
10	0.9	0.82	0.74
1	0.37	0.14	0.05
0.25	0.018	0.0003	0.000006
0.1	0.00005	2×10^{-9}	9×10^{-14}



SA – Cooling Schedule

- Starting Temperature
- Final Temperature
- Temperature Decrement
- Iterations at each temperature



SA – Cooling Schedule II

- Starting Temperature (T_0)
 - ▶ *hot enough*: to allow almost all neighbours
 - ▶ *not so hot*: random search for sometime
 - ▶ Estimate a suitable starting temperature:
 - Reduce quickly to 60% of worse moves are accepted
 - Use this as the starting temperature
- Final Temperature
 - ▶ Usually 0, however in practise, not necessary
 - ▶ T is low: accepting a worse move is almost the same as $T=0$
 - ▶ The stopping criteria: either be a suitably low T , or “frozen” at the current T (i.e. no worse moves are accepted)



SA – Cooling Schedule III

- Temperature Decrement

- ▶ **Linear:** $T = T - x$
 - ▶ **Geometric:** $T = T * \alpha$

- Experience: α is typically in the interval [0.9, 0.99]

- ▶ **Lundy Mees:**

$$T = \frac{T}{1 + \beta T}$$

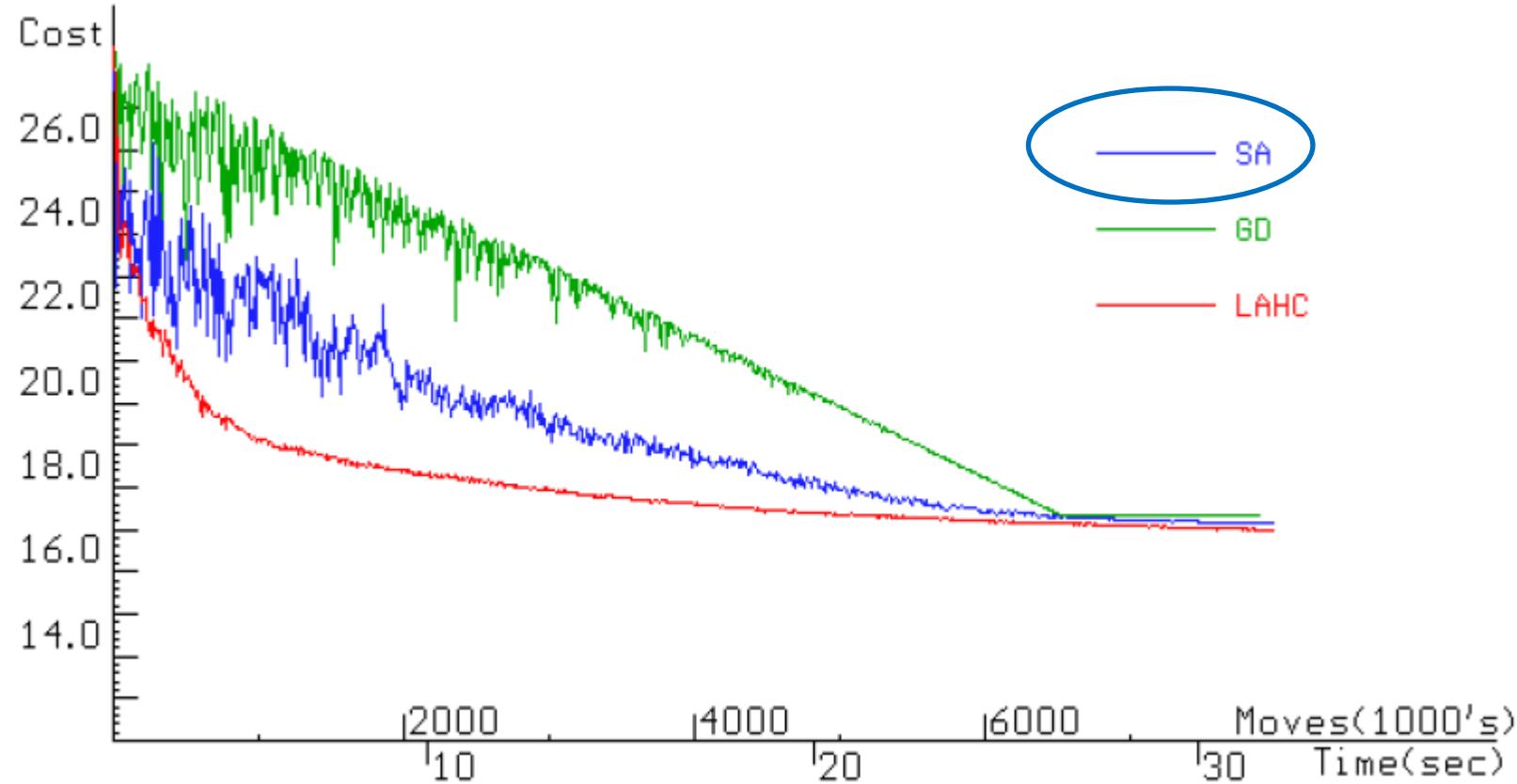
- One iteration at each T , but decrease T very slowly. Experience: β is typically a very small value, that is close to 0 (e.g., 0.0001)



SA – Cooling Schedule IV

- **Iterations at each temperature**
 - ▶ One iteration at each T
 - ▶ A constant number of iterations at each T
 - ▶ Compromise
 - Either: a large number of iterations at a few T s, or
 - A small number of iterations at many T s, or
 - A balance between the two
 - ▶ Dynamically change the no. of iterations
 - At higher T s: less no. of iterations
 - At lower T s: large no. of iterations, local optimum fully exploited
- **Reheating**, if stuck at a local optimum for a while, increase the current temperature with a certain rate

Behaviour of Simulated Annealing – An Example



Simulated Annealing with Geometric Cooling



INPUT: $T_0 (> 0)$, α (cooling rate < 1.0)

Generate an initial solution S_0 using some heuristics

Set $S_k = S_{best} = S_0$, $k=0$;

REPEAT

Select $S_{new} \in \mathcal{N}(S_k)$ // Make a move from S_k to S_{new} based on \mathcal{N}

If $F(S_{new}) < F(S_k)$ **then** $S_{k+1} = S_{new}$ // an improving move is made

else // A worsening solution is obtained

generate a random uniform number in $(0,1]$, $U(0,1)$

$$-\frac{F(S_{new}) - F(S_k)}{T_k}$$

If $U(0,1) < e^{-\frac{F(S_{new}) - F(S_k)}{T_k}}$ **then** $S_{k+1} = S_{new}$ // Accept worsening move

else $S_{k+1} = S_k$ // Reject worsening move

// Keep track of the best solution found so far

If $F(S_{new}) < F(S_{best})$ **then** $S_{best} = S_{new}$

$T_{k+1} = \alpha T_k$; // geometric cooling: multiply previous temp with α (value < 1.0)

$k = k+1$; // one iteration at each T

UNTIL (stopping condition = true)



Exercise

- Consider the MAX-SAT problem instance:
 - ▶ $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Objective function: number of unsatisfied clauses
- Apply the simulated annealing to the instance starting out with the 100100 as an initial solution for 3 SA steps/iterations.
- **Neighbourhood operator:** perform random 1-bit flip
- Choose $\alpha = 0.9$ and $T_0 = 0.9$
- Use the following numbers in the given order as **random numbers where appropriate**
 - ▶ for choosing a random literal i in $[1..6]$: <2, 3, 1, 6, ...> to apply 1-bit flip of (i)th literal's truth assignment in the candidate solution
 - ▶ for $U(0,1)$: <0.47, 0.089, ...>



Running of SA

$$\begin{array}{ccccc}
 0 & 1 & 2 \\
 (a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \\
 \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e) \\
 3 & 4 & 5
 \end{array}$$

$$S_{best} = S_0 = \overset{a}{1} \overset{b}{0} \overset{c}{1} \overset{d}{0} \overset{e}{0} \overset{f}{0}, T_0 = 0.9$$

$F(S_0) = F(100100) = 3$ (Clause SAT: 012345) = $F(S_{best})$

k=0

$$S_{new} \leftarrow \mathcal{N}(S_0): 110100 \quad \triangleright \text{Rand}[1..6]: <2, 3, 1, 6, \dots>$$

$$S_{new} = 110100, \text{ Clause SAT: 012345: } F(S_{new}) = 4 > F(S_0) = 3$$

$$U(0,1) = 0.47 > e^{-\frac{4-3}{0.9}} = 0.33 \quad \triangleright U(0,1) : <0.47, 0.089, \dots>$$

$$S_1 = 100100$$

$$F(S_{new}) = 4 > F(S_{best}) = 3 \quad \triangleright S_{best} \text{ does not change}$$

$$T_1 = \alpha T_0 = 0.9 \cdot 0.9 = 0.81 \quad \triangleright \text{update temperature}$$

k=1

$$S_{new} \leftarrow \mathcal{N}(S_1): 101100 \quad \triangleright \text{Rand}[1..6]: <2, 3, 1, 6, \dots>$$

$$S_{new} = 101100, \text{ Clause SAT: 012345: } F(S_{new}) = 1 < F(S_1) = 3$$

$$S_2 = 101100 \quad \triangleright \text{accept } S_{new}: S_2 = S_{new}$$

$$F(S_{new}) = 1 < F(S_{best})$$

$$S_{best} = 101100 \quad \triangleright \text{update best solution}$$

$$\textcolor{red}{F(S_{best}) = 1}$$

$$T_2 = \alpha T_1 = 0.9 \cdot 0.81 = 0.729 \quad \triangleright \text{update temperature}$$

Running of SA (cont.)



k=2

$S_{new} \leftarrow \mathcal{N}(S_2)$: **001100** ➤ Rand[1..6]: <2, 3, **1**, 6, ...>

$S_{new} = 001100$, **01**2345, $F(S_{new}) = 2 > F(S_2) = 1$

$$U(0,1) = 0.089 < e^{-\frac{2-1}{0.729}} = 0.253 \quad \triangleright U(0,1) : <\!0.17, \mathbf{0.089}, \!\dots\!>$$

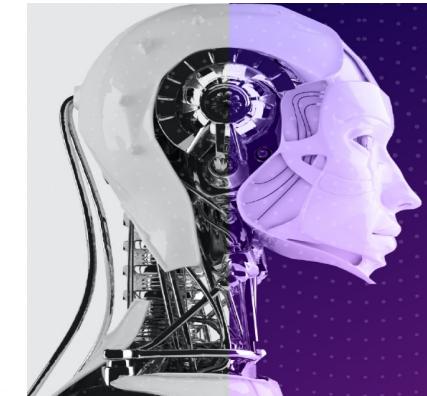
$S_3 = 001100$ ➤ accept S_{new} : $S_3 = S_{new}$

$F(S_{new}) = 2 > F(S_{best}) = 1$ ➤ S_{best} does not change (101100)

$$T_3 = \alpha T_2 = 0.9 \cdot 0.729 = 0.6561$$

...

2. Parameter Setting Issues and Tuning Methods



COM2001/2011: Artificial Intelligence Methods

Ender Özcan

Lecture 4



UNITED KINGDOM • CHINA • MALAYSIA

Metaheuristics



Local Search	• [Kirkpatrick, 1983]	Simulated Annealing (SA)
	• [Glover, 1986]	Tabu Search (TS)
	• [Voudouris, 1997]	Guided Local Search (GLS)
	• [Stutzle, 1999]	Iterated Local Search (ILS)
	• [Mladenovic, 1999]	Variable Neighborhood Search (VNS)
Population-based	• [Holland, 1975]	Genetic Algorithm (GA)
	• [Smith, 1980]	Genetic Programming (GP)
	• [Goldberg, 1989]	Genetic and Evolutionary Computation (EC)
	• [Moscato, 1989]	Memetic Algorithm (MA)
	• [Kennedy and Eberhart, 1995]	Particle Swarm Optimisation (PSO)
Constructive	• [Dorigo, 1992]	Ant Colony Optimisation (ACO)
	• [Resende, 1995]	Greedy Randomized Adaptive Search Procedure (GRASP)

Metaheuristics



Examples of Parameters

Local Search	<ul style="list-style-type: none">• SA• TS• GLS• ILS• VNS	T_0 (initial temperature), α (cooling rate) Tabu list size (tabu tenure) λ (intensification control), a (coefficient) <u>perturbation, perturbation strength</u> k_{min}, k_{max} (smallest, largest neighbourhood size)
Population-based	<ul style="list-style-type: none">• GA• GP• EC• MA• PSO	population size, mutation probability, <i>mutation</i> number of particles, V_{max} (maximum velocity)
Constructive	<ul style="list-style-type: none">• ACO• GRASP	weight of pheromone, evaporation rate restricted candidate list parameter



Parameter Types

- Categorical/symbolic/structural parameters
 - ▶ Choice of initialisation method, choice of mutation,...
- Ordinal parameters
 - ▶ Neighbourhoods (e.g., small, medium, large),...
- Numerical/behavioural parameters
 - ▶ integer, real-valued, ...
 - ▶ population sizes, evaporation rates,...
 - ▶ values may depend on the setting of categorical or ordinal parameters



Parameter Setting Methods

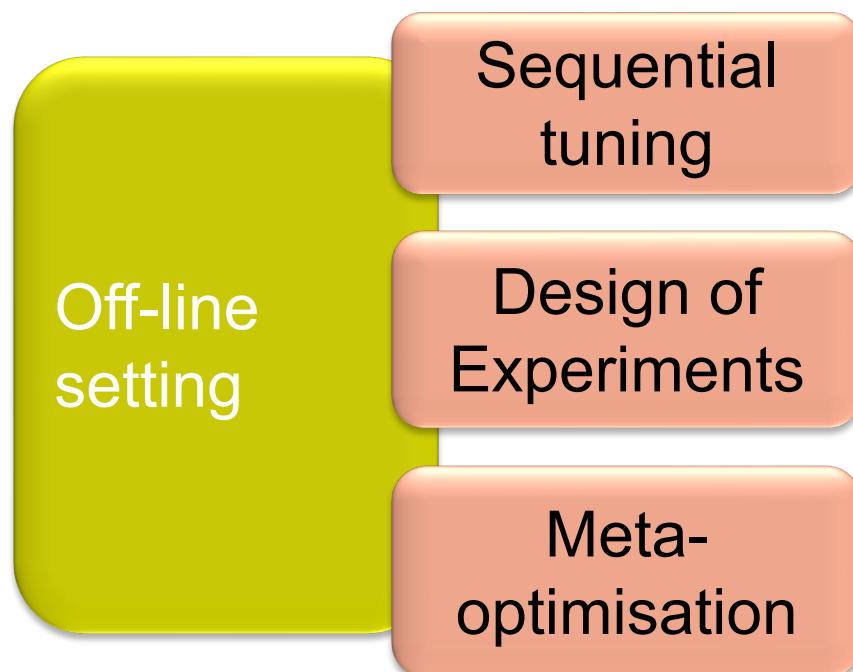
- **Parameter tuning:** Finding the best initial settings for a set of parameters before the search process starts (*off-line*). E.g., fixing the mutation strength in ILS, mutation probability in genetic algorithms, etc.
 - ▶ The initial parameter setting influences the performance of a metaheuristic
- **Parameter control:** Managing the settings of parameters during the search process (*online*) (dynamic, adaptive, self-adaptive). E.g., changing the mutation strength in ILS, changing the mutation probability in genetic algorithms during the search process
 - ▶ Controlling parameter setting could yield a system which is not sensitive to its initial setting

A Classification

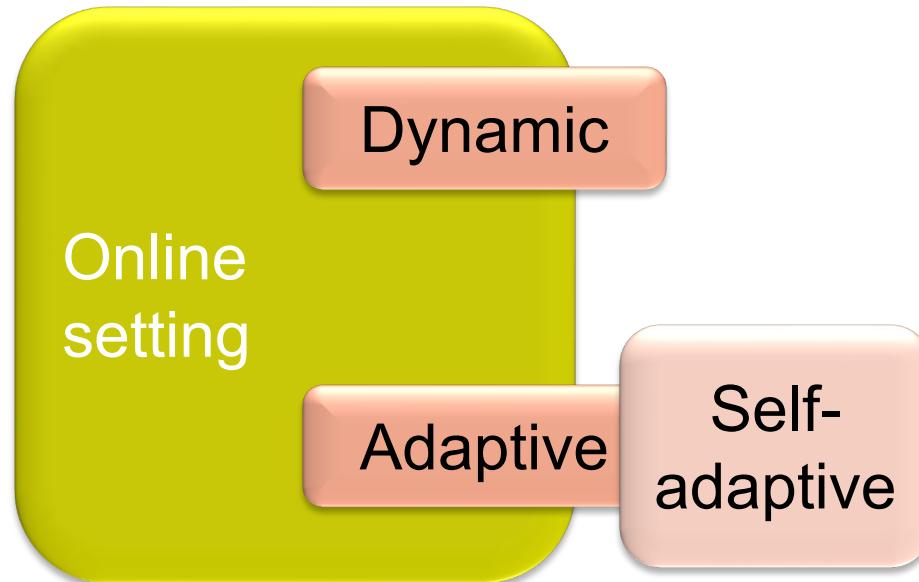


Parameter Setting

Parameter Tuning



Parameter Control





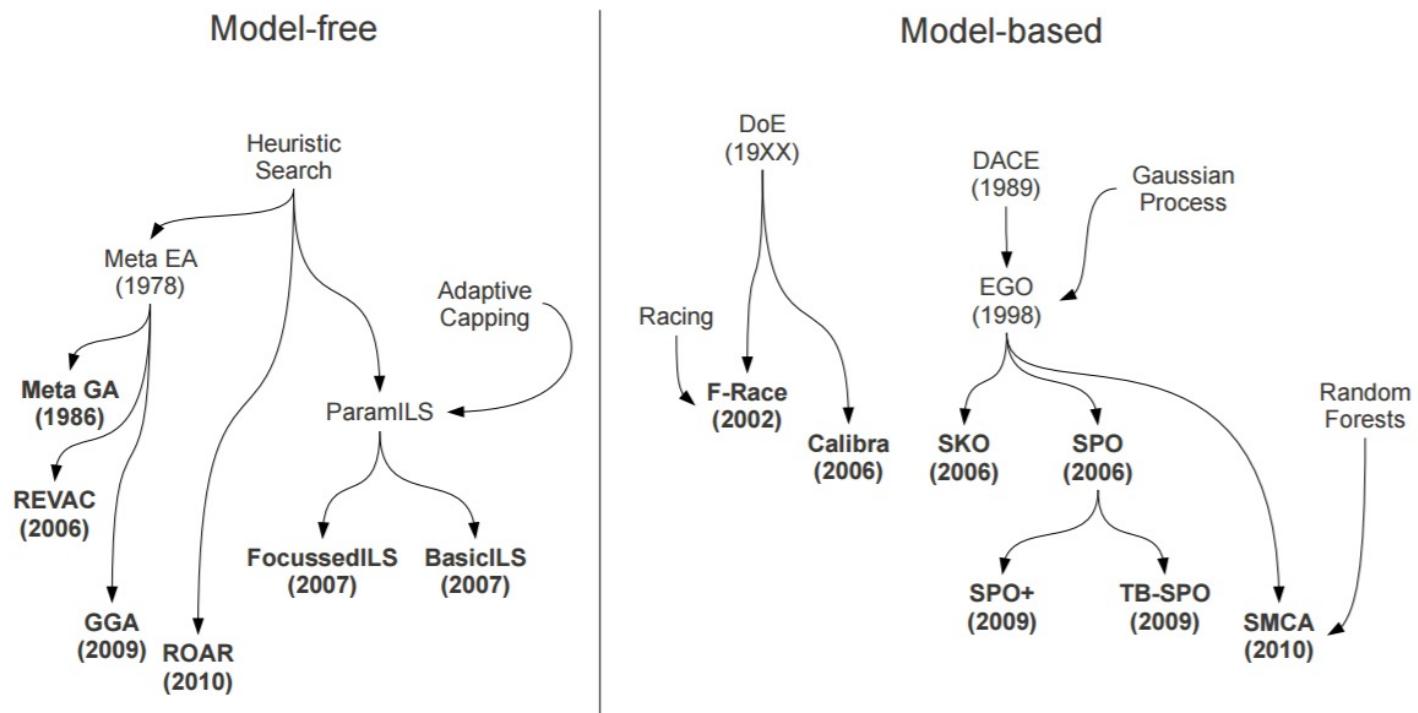
Parameter Tuning Methods

- Traditional approaches
 - ▶ Use of an arbitrary setting (e.g., $\phi = 0$ and $\delta = 80$)
 - ▶ Trial&error with settings based on intuition
 - ▶ Use of theoretical studies
 - ▶ A mixture of above
- Sequential tuning: fix parameter values successively (e.g., fix fixing $\delta = 20$ and tune ϕ that is try {0, 0.3, 0.5, 0.8, 1.0}, then fixing the best setting for ϕ from the previous trials and tune δ that is try {20, 40, 50, 60, 80})
- Design of experiments
- Meta-optimisation: use a metaheuristic to obtain “optimal” parameter settings

Example: Assume
 $\phi \in \{0, 0.3, 0.5, 0.8, 1.0\}$ and
 $\delta \in \{20, 40, 50, 60, 80\}$



Automated Parameter Tuning



Source: http://www.mff.cuni.cz/veda/konference/wds/proc/pdf10/WDS10_109_i1_Dobslaw.pdf

I-race (download: <http://iridia.ulb.ac.be/irace/>)

ParamILS (download: <http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/>)

SPOT (download: <https://cran.r-project.org/web/packages/SPOT/index.html>)



Design of Experiments (DoE)

- A systematic method (controlled experiments) to determine the relationship between controllable and uncontrollable *factors* (inputs to the process, variables) affecting a process (e.g., running of an algorithm), their *levels* (settings) and the *response* (output) of that process (e.g., quality of solutions obtained – performance of an algorithm). (Fisher 1926, 1935)
- Important outcomes are measured and analysed to determine the factors and their settings that will provide the best overall outcome
 - ▶ E.g., Two factors; $\phi \in \{0, 0.3, 0.5, 0.8, 1.0\}$ and $\delta \in \{20, 40, 50, 60, 80\}$, each with 5 levels – at least 5^2 runs required



Fractional Factorial Designs

- Assuming the number of factors is k in an n level factorial design, then number of runs for even a single replicate of the n^k design becomes very large.
 - ▶ E.g, a replicate of an 8 factor two level experiment would require $2^8 = 256$ runs. If a run consists of 31 trials each taking 5 min, such an experiment would take ~ 28 days (1 instance)
- Fractional factorial designs can be used in these cases to draw out valuable conclusions from fewer runs.
- Key observation: Responses are often affected by a small number of main effects and lower order interactions, while higher order interactions are relatively unimportant.



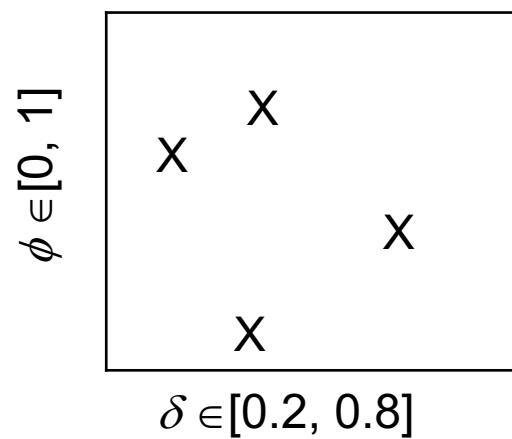
Design of Experiments – Sampling

- Whenever factorial design is not possible, sampling is performed:
 - ▶ **Random**
 - ▶ **Latin Hyper-cube**
 - ▶ **Orthogonal**
- Example: Assume that we have two parameters,
 $\phi \in [0, 1]$ and $\delta \in [0.2, 0.8]$



Random Sampling

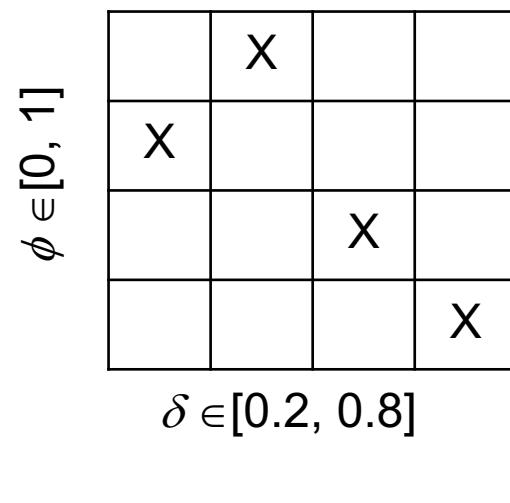
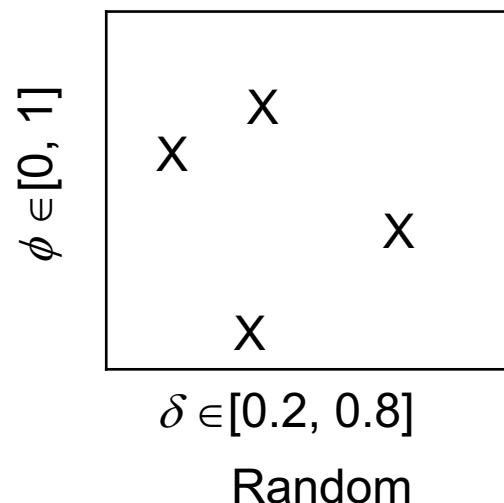
- Generate each sample point independently (M)
- Example:





Latin Hypercube Sampling

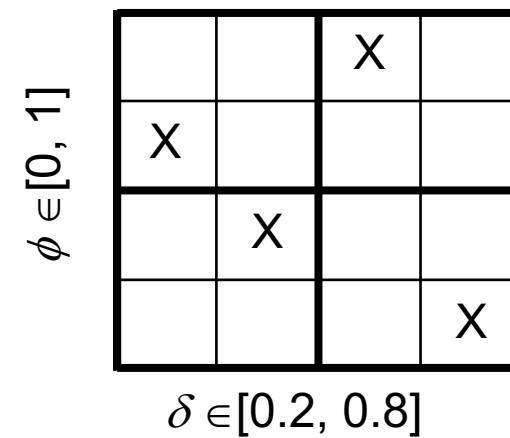
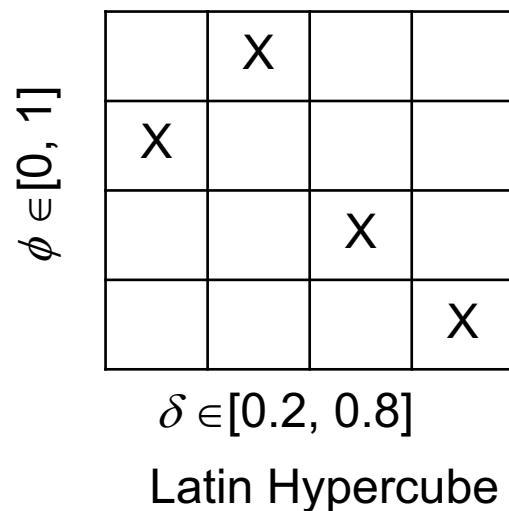
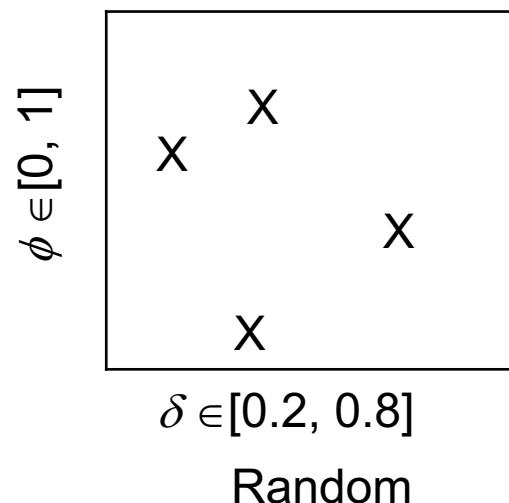
- Decide the number of sample points (M) for N variables and for each sample point remember in which row and column the sample point was taken
- Example:





Orthogonal Sampling

- The sample space is divided into equally probable subspaces. Sample points simultaneously, ensuring they form an ensemble of Latin Hypercube sample
- Example:



Taguchi Orthogonal Arrays Method for Parameter Tuning



- Developed by Genichi Taguchi to improve the quality of manufactured goods initially, then applied to problems from the other fields
- Aim: make a “product” or “process” less variable (more *robust*) in the face of variation over which we have little or no control.
- Taguchi method is a structured statistical (experimental design) method for determining the *best* combination of parameter settings to achieve certain objective(s)

Taguchi Orthogonal Arrays Method for Parameter Tuning II



- Best when there are an intermediate number of parameters/variables/ factors (3 to 50), few interactions between variables, and when only a few variables contribute significantly.
 - ▶ E.g., 3 factors and 2 levels (settings) per factor: 2^3 combinations
- Taguchi's orthogonal arrays are highly fractional orthogonal designs, which can be used to estimate main effects using only a few experimental runs (which can consist of multiple trials). E.g.,

L4 (2^3)

Run	Columns		
	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

L8 (2^7)

Run	Columns						
	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

L9 (3^4)

Run	Columns			
	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1



Taguchi Orthogonal Arrays Method for Parameter Tuning – Main Steps

1. Selection of control parameters
(independent variables/factors)
 2. Selection of number of level settings for each parameter
 3. Select a suitable orthogonal array based on the number of parameters and levels
 4. Conduct the experiments using the algorithm on the selected subset of test instances
 5. Analyse the results
 6. Determine the optimum levels for the individual parameters
 7. Confirmation experiment
Use the same configuration for the rest of the experiments
- Planning Conduct Analysis Validation

Taguchi Orthogonal Arrays Method – Software Packages



Minitab

- MATLAB, Minitab, R, Octave, JMP, DOE++,

...

Planning

- Assume that we have a metaheuristic with 4 parameters (factors):

- param1 $\in [0, 1.0]$ (\mathbb{R})
- param2 $\in [0, 1.0]$ (\mathbb{R})
- param3 $\in [1..80]$ (\mathbb{Z}^+)
- param4 $\in [1..5]$ (\mathbb{Z}^+)

- Choosing a suitable Taguchi orthogonal array design:

- We have 4 factors with a maximum of 5 levels for each factor

- L4: Three two-level factors
- L8: Seven two-level factors
- L9: Four three-level factors
- L12: Eleven two-level factors
- L16: Fifteen two-level factors
- L16b: Five four-level factors
- L18: One two-level and seven three-level factors
- L25: Six five-level factors
- L27: Thirteen three-level factors
- L32: Thirty-two two-level factors

- Selection of control parameters (independent variables)
- Selection of number of level settings for each parameter
- Select a suitable orthogonal array based on the number of parameters and levels

Parameter levels

		Value Options				
[5 levels]	param1	{0.2, 0.4, 0.6, 0.8, 1.0}				
[5 levels]	param2	{0.2, 0.4, 0.6, 0.8, 1.0}				
[5 levels]	param3	{5, 10, 20, 40, 80}				
[4 levels]	param4	{2, 3, 4, 5}				

Taguchi orthogonal array

X1 X2 X3 X4 X5 X6	Experiment number	param1	param2	param3	param4
1 1 1 1 1 1	1	1	1	1	1
1 2 1 2 2 2	2	1	2	2	2
1 3 3 3 3 3	3	1	3	3	3
1 4 4 4 4 4	4	1	4	4	4
1 5 5 5 5 5	5	1	5	5	2
2 1 2 3 4 5	6	2	1	2	3
2 2 3 4 5 1	7	2	2	3	4
2 3 4 5 1 2	8	2	3	4	1
2 4 5 1 2 3	9	2	4	5	1
2 5 1 2 3 4	10	2	5	1	2
3 1 3 5 2 4	11	3	1	3	4
3 2 4 1 3 5	12	3	2	4	1
3 3 5 2 4 1	13	3	3	5	2
3 4 1 3 5 2	14	3	4	1	3
3 5 2 4 1 3	15	3	5	2	4
4 1 4 2 5 3	16	4	1	4	2
4 2 5 3 1 4	17	4	2	5	3
4 3 1 4 2 5	18	4	3	1	4
4 4 2 5 3 1	19	4	4	2	2
4 5 3 1 4 2	20	4	5	3	1
5 1 5 4 3 2	21	5	1	5	4
5 2 1 5 4 3	22	5	2	1	3
5 3 2 1 5 4	23	5	3	2	1
5 4 3 2 1 5	24	5	4	3	2
5 5 4 3 2 1	25	5	5	4	3





Planning

Parameter levels

	Value Options			
[5 levels]	param1	{0.2, 0.4, 0.6, 0.8, 1.0}		
[5 levels]	param2	{0.2, 0.4, 0.6, 0.8, 1.0}		
[5 levels]	param3	{5, 10, 20, 40, 80}		
[4 levels]	param4	{2, 3, 4, 5}		
		1	2	3
		4		

Configuration ID	e.g.:	param1	param2	param3	param4
1	e.g.:	1 (0.2)	1 (0.2)	1 (5)	1(2)
2		1	2	2	2
3		1	3	3	3
4		1	4	4	4
5		1	5	5	2
6		2	1	2	3
7		2	2	3	4
8		2	3	4	1
9		2	4	5	1
10		2	5	1	2
11		3	1	3	4
12		3	2	4	1
13		3	3	5	2
14		3	4	1	3
15		3	5	2	4
16		4	1	4	2
17		4	2	5	3
18		4	3	1	4
19		4	4	2	2
20		4	5	3	1
21		5	1	5	4
22		5	2	1	3
23		5	3	2	1
24		5	4	3	2
25	e.g.:	5 (1.0)	5 (1.0)	4(40)	3(4)

Conduct Experiments

- Run experiments, say for 30 times using the algorithm with each setting (potentially on multiple ‘training’ instances)
- Use a performance metric, e.g., record Formula 1 score for each run/trial (higher the better), where the top 8 algorithms score **10, 8, 6, 5, 4, 3, 2 and 1 point(s).**

Configuration ID	param1	param2	param3	param4
1	0.2	0.2	5	2
2	0.2	0.4	10	3
3	0.2	0.6	20	4
4	0.2	0.8	40	5
5	0.2	1.0	80	3
6	0.4	0.2	10	4
7	0.4	0.4	20	5
8	0.4	0.6	40	2
9	0.4	0.8	80	2
10	0.4	1.0	5	3
11	0.6	0.2	20	5
12	0.6	0.4	40	2
13	0.6	0.6	80	3
14	0.6	0.8	5	4
15	0.6	1.0	10	5
16	0.8	0.2	40	3
17	0.8	0.4	80	4
18	0.8	0.6	5	5
19	0.8	0.8	10	3
20	0.8	1.0	20	2
21	1.0	0.2	80	5
22	1.0	0.4	5	4
23	1.0	0.6	10	2
24	1.0	0.8	20	3
25	1.0	1.0	40	4





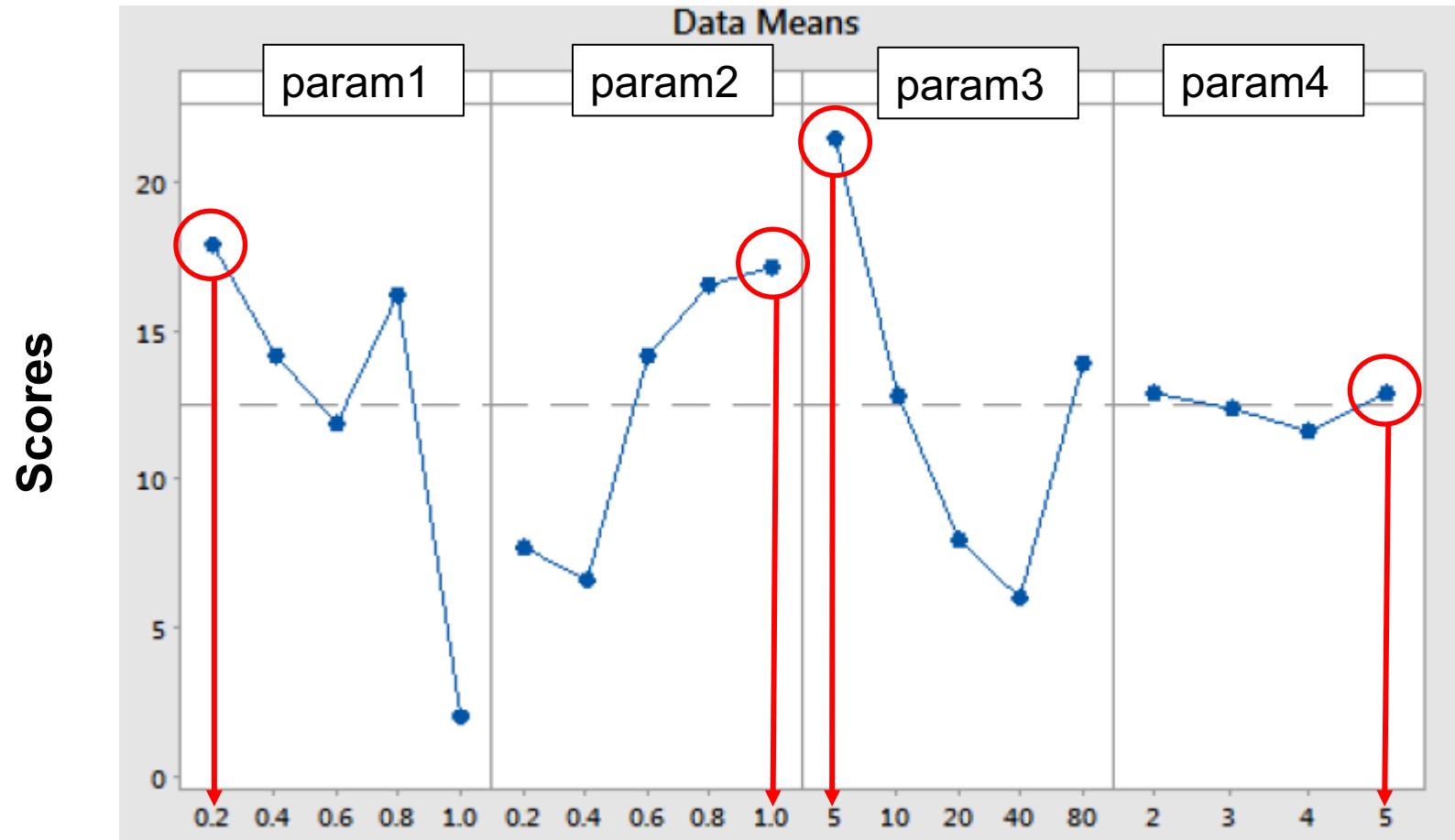
Conduct Experiments

- Collect results using an appropriate performance indicator: e.g., obtain mean F1 score per run for each algorithm with a specific parameter combination setting per instance, and sum those scores up from all instances
- For analysis: Main effect of param3 for the setting of 20 is the *mean score* for all experiments with that setting:
 $(17.5+2.5+0+18.38+1.88)/5 = 8.052$

Configuration ID	param1	param2	param3	param4	Performance Indicator/Scores
1	0.2	0.2	5	2	24.88
2	0.2	0.4	10	3	14
3	0.2	0.6	20	4	17.50
4	0.2	0.8	40	5	17.88
5	0.2	1.0	80	3	15.50
6	0.4	0.2	10	4	9.88
7	0.4	0.4	20	5	2.50
8	0.4	0.6	40	2	8.88
9	0.4	0.8	80	2	22.50
10	0.4	1.0	5	3	27.25
11	0.6	0.2	20	5	0
12	0.6	0.4	40	2	1
13	0.6	0.6	80	3	12
14	0.6	0.8	5	4	24.38
15	0.6	1.0	10	5	22.25
16	0.8	0.2	40	3	0
17	0.8	0.4	80	4	25.88
18	0.8	0.6	5	5	30.88
19	0.8	0.8	10	3	16.25
20	0.8	1.0	20	2	18.38
21	1.0	0.2	80	5	4
22	1.0	0.4	5	4	0
23	1.0	0.6	10	2	1.88
24	1.0	0.8	20	3	1.88
25	1.0	1.0	40	4	2.50



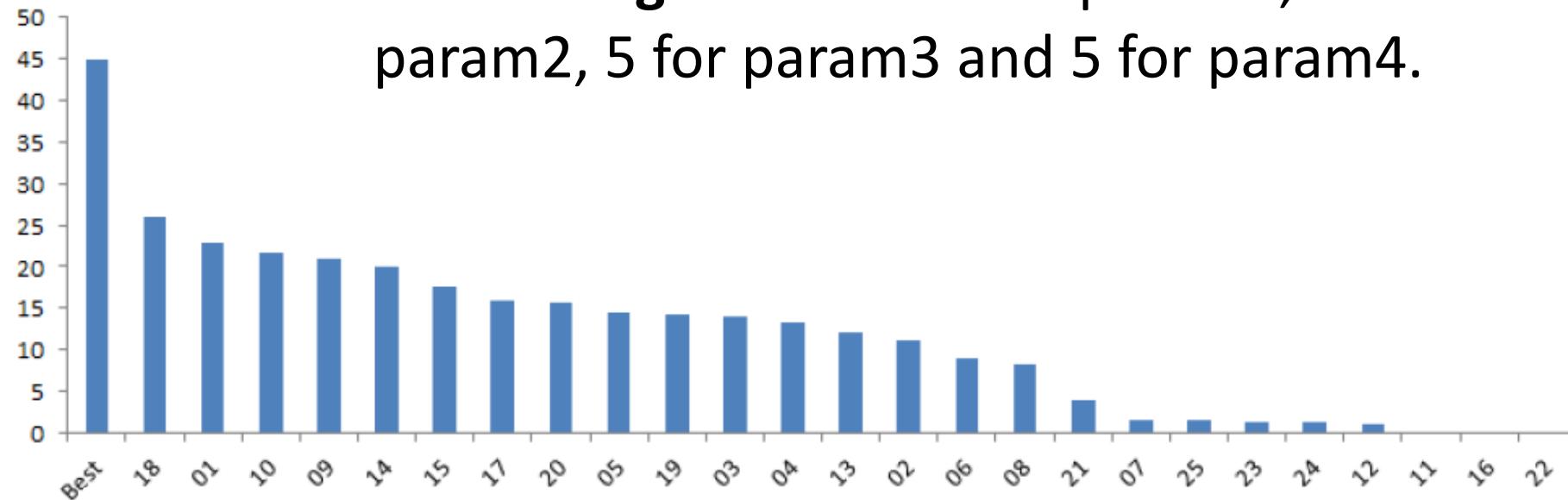
Analysis – Main Effects Plot





Validation: Confirmation Run

Best configuration: 0.2 for param1, 1.0 for param2, 5 for param3 and 5 for param4.



If failure, then chosen levels could be problematic needs to be reviewed.



Summary I

- It is not trivial which (meta)heuristic optimisation/search algorithm will perform better than the other one on a given problem domain
 - ▶ Experiments using a single or only small instances might not be realistic or true performance indicator of an algorithm
- There is a variety of statistical tools for the performance analyses of algorithms.
- Move acceptance methods as a part of local search metaheuristics can be used for escaping from the local optima, enabling acceptance of non-improving solutions.
 - ▶ Great Deluge and Simulated Annealing are well-known and well-studied such local search metaheuristics, and recently Late Acceptance. These are easy to implement methods and there are more elaborate variants.
 - ▶ Move acceptance methods vary depending on the mechanism and components they have for accepting a non-improving solution.



Summary II

- Many (meta)heuristic optimisation/search algorithms come with parameters which often require an initial setting (tuning)
 - ▶ Parameter tuning is possible, however it is time consuming.
 - ▶ There is a range of different techniques varying from manual/semi-automated experimental design methods to automated tuning, such as, Taguchi method, I-race.
 - ▶ Parameter control as an alternative to parameter tuning changes parameter values during the run of the algorithm
 - ▶ There is no guidance indicating which method is the best, however many studies show that parameter tuning/control often does improve the performance of an algorithm as compared to the variant where it is not used



Summary III

- Move acceptance methods can be hybridised:
 - ▶ Single stage strategy:
 - Use a decision mechanism to choose which move acceptance to employ at each step.
 - Use group decision making. E.g, combine simulated annealing, great deluge and late acceptance and apply majority vote.
 - ▶ Multi-stage
 - Use a different move acceptance at different stages of the search process. For example, use improving and equal until the search process gets stuck, then switch to simulated annealing.



HOME EXERCISE (SEE THE FORUM)



Soft Drink Bottling

- How would you represent this scheduling problem using the standard notation?
- single machine
- 4 flavours
- each flavour has its own filling time
- cleaning and changeover time between the bottling of successive flavours

aim: to minimise cycle time,

sufficient: to minimise the total changeover time

	f_1	f_2	f_3	f_4
f_1	—	2	70	50
f_2	6	—	3	4
f_3	8	3	—	2
f_4	50	5	6	—

$$f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_4 \rightarrow f_1 \\ 2+3+2+50 = 57$$

$$f_3 \rightarrow f_4 \rightarrow f_2 \rightarrow f_1 \rightarrow f_3 \\ 2+5+6+70 = 83$$

$$f_2 \rightarrow f_3 \rightarrow f_4 \rightarrow f_1 \rightarrow f_2 \\ 3+2+50+2 = 57$$

$$f_4 \rightarrow f_2 \rightarrow f_3 \rightarrow f_1 \rightarrow f_4 \\ 5+3+8+50 = 66$$

optimal: $f_1 \rightarrow f_2 \rightarrow f_4 \rightarrow f_3 \rightarrow f_1 \\ 2+4+6+8 = 20$

Illustration of a Run of Tabu Search on a Scheduling Problem



Example:

jobs	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
$1 \mid d_j \mid \sum w_j T_j$	14	12	1	12

Schedule four jobs on a machine

$$T_j = \max(C_j - d_j, 0)$$

tardiness of job j

Neighbourhood operator: go through all schedules that can be obtained through adjacent pairwise interchanges, choose the best.

Tabu-list: pairs of jobs (j, k) that were swapped within the last two moves

Run the algorithm on the following slide for 2 while loop iterations, starting with the initial solution: $S_0 = <2, 1, 4, 3>$

Applying Tabu Search Algorithm to a Scheduling Problem – Example



```
1 I=0; si← initialize, maxTabuSize = 2;
2 Sbest ← s0
3 tabuList ← []
4 while (not stoppingCondition())
5     candidateList ← []
6     bestCandidate ← null
7     for (sCandidate in sNeighborhood) // any configuration in the neighbourhood of s: sCandidate ∈ N(s)
8         if ( (not tabuList.contains(sCandidate) and ( F(sCandidate) < F(bestCandidate) ) )
9             bestCandidate ← sCandidate; bestMove ← (i, j) // swapped adjacent jobs
10            end
11        end
12        I++; si← bestCandidate
13        if ( F(bestCandidate) < F(Sbest) )
14            Sbest ← bestCandidate
15        end
16        tabuList.push( reverse(bestMove) ); // save (j, i) into tabu list
17        if (tabuList.size > maxTabuSize)
18            tabuList.removeFirst()
19        end
20    end
21 return Sbest
```

Q&A



**Thank you.
Ender Özcan**

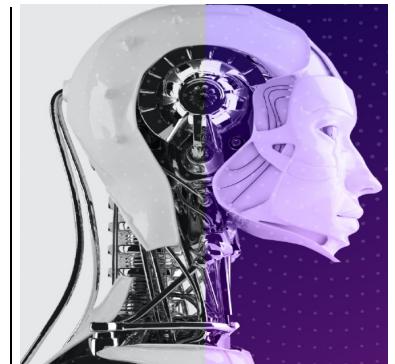
ender.ozcan@nottingham.ac.uk

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
<http://www.cs.nott.ac.uk/~pszeo>

Evolutionary Algorithms I

Lecture 5

Ender Özcan



The University of
Nottingham

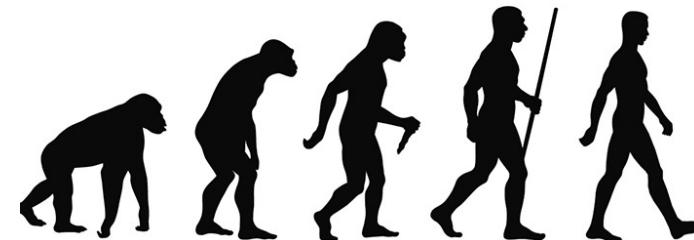
UNITED KINGDOM • CHINA • MALAYSIA

264



Evolutionary is Revolutionary

- Nature as a Problem Solver: 4.55 Billion years of evolution can't be wrong.



Evolution: Gradual change in the inherited characteristics of a population of animals or plants over successive generations

- Beauty-of-nature argument: Complexity achieved in *short* time in nature.
- Can we solve complex problems as quickly and reliably on a computer?



Evolution at Work

- Heritable characteristics or heritable traits, e.g., the colour of your eyes are passed from one generation to the next via DNA (a molecule that encodes genetic information)
- Change or genetic variation comes from:
 - ▶ Mutations: changes in the DNA sequence,
 - ▶ Crossover: reshuffling of genes through sexual reproduction and migration between populations
- Evolution is driven by natural selection – survival of the fittest
- Genetic variations that enhance survival and reproduction become and remain more common in successive generations of a population.

A famous example – peppered moth evolution

white/black-bodied peppered moth

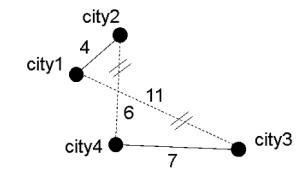


- Before the Industrial Revolution, the black peppered moth was rare.
 - ▶ The frequency of the dark allele was about 0.01%
- The rapid industrialization and rampant coal use coated the British trees in a layer of soot
- By the mid-19th century, the number of dark-coloured moths had risen noticeably, and by 1895, the frequency was 98% in Manchester



Evolutionary Algorithms (EAs)

- EAs simulate natural evolution (Darwinian Evolution) of **individual** structures at the genetic level using the idea of ***survival of the fittest*** via processes of **selection, mutation, and reproduction (recombination)**
- An ***individual (chromosome)*** represents a candidate solution for the problem at hand. (e.g., $<2\ 1\ 3\ 4>$)
- A collection of individuals currently “alive”, called **population** (set of individuals/chromosomes) is evolved from one **generation (iteration)** to another depending on the **fitness** of individuals in a given **environment**, indicating how fit an individual is, (how close it is to the *optimal* solution) – **objective value**. (e.g., $f(<2\ 1\ 3\ 4>) = 28$)
- **Hope:** Last generation will contain the best solution





Evolutionary Algorithms (EAs) II

- **Genetic Algorithms** (evolves (bit) strings) ⇒ Nils Aall Barricelli 1954, Holland 1975
 - ▶ Memetic Algorithms ⇒ Moscato 1989
- **Evolutionary Programming** (evolves parameters of a program with a fixed structure) ⇒ Fogel, Owens, Walsh 1966
- **Evolution Strategies** (vectors of real numbers) ⇒ Rechenberg 1973
- **Genetic Programming** (evolves computer programs in tree form) ⇒ Koza 1992
 - ▶ **Gene Expression Programming** (computer programs of different sizes are encoded in linear chromosomes of fixed length)
 - ▶ **Grammatical Evolution** (evolves solutions wrt a specified grammar) ⇒ Ryan, Collins and O'Neill 1998

Genetic Algorithms (GAs)



Pseudocode of a Generic Genetic Algorithm



```
begin
    generate initial population; // initialise
    calculate fitness values; // evaluate population
    do
    {
        perform reproduction; // select parents
        recombine pairs with  $p_c$ ; // apply crossover
        apply mutation with  $p_m$ ; // mutate
        offspring/children
        calculate fitness values; // eval. population
        replace current population;
    } while termination criteria not satisfied;
end
```



Basic Components of GAs

- A genetic **representation (encoding)** for candidate solutions (individuals) to the problem at hand
- An **initialisation** scheme to generate the first population (set) of candidate solutions (individuals)
- A **fitness (evaluation) function** that plays the role of the environment, rating the solutions in terms of their fitness
- A scheme for **selecting mates (parents)** for recombination
- **Crossover (recombination)** exchanges genetic material between mates producing **offspring (children)**
- **Mutation** perturbs an individual creating a new one
- **Replacement strategy** to select the surviving individuals for the next generation
- **Termination Criteria**
- Values for various parameters that GA uses (population size, probabilities of applying genetic operators, etc)



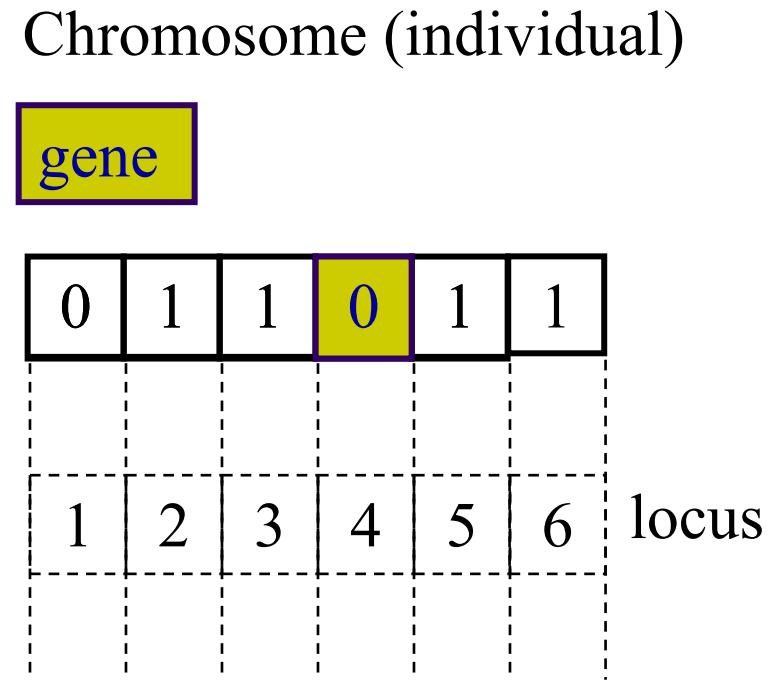
GA Components: Representation

- **Haploid structure** is used: Each individual contains one chromosome
- Each individual is evaluated and has an associated **fitness** value
- Chromosomes contain a fixed number of genes: **chromosome length**
- Traditionally **binary encoding** is used for each gene: **Allele** value $\in \{0,1\}$
- A population contains a fixed number of individuals: **population size**
- Each iteration is referred as **generation**

GA Components: Initialisation



- Random Initialisation
- **Population size** number of individuals are created randomly
- Each gene at a locus of an individual is assigned an **allele** value 0 or 1 randomly, decided by flipping a coin (E.g., if the random value is <0.5, then allele is assigned to 0, otherwise to 1).



Example – MAX-SAT: Initialisation



```
begin
  generate initial population; // initialise
  calculate fitness values; // evaluate population
  do
  {
    perform reproduction; // select parents
    recombine pairs with pc; // apply crossover
    apply mutation with pm; // mutate
    offspring/children
    calculate fitness values; // eval. population
    replace current population;
  } while termination criteria not satisfied;
end
```

0 1 2 3 4 5
$$(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$$

- 1: <0.98, 0.85, 0.13, 0.04, 0.47, 0.44>
- 2: <0.09, 0.63, 0.22, 0.54, 0.07, 0.37>
- 3: <0.21, 0.03, 0.72, 0.84, 0.19, 0.49>
- 4: <0.61, 0.43, 0.87, 0.53, 0.97, 0.14>

101110

- Assume *population size* is 4
- The individual size/*chromosome length* is 6 (since we have 6 literals: *abcdef*)
- So, create 4 individuals with 6 genes within their chromosomes, where each allele at a locus is determined **randomly** (by throwing a random number in [0,1]).

Example – MAX-SAT: Initialisation



<i>i</i>	Chromosome <i>abcdef</i>
1:	110000
2:	010100
3:	001100
4:	101110

0 1 2 3 4 5

$$(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$$

<i>i</i>	Chromosome <i>abcdef</i>
1:	110000
2:	010100
3:	001100
4:	101110

- Assume *population size* is 4
- The individual size/*chromosome length* is 6 (since we have 6 literals: *abcdef*)
- So, create 4 individuals with 6 genes within their chromosomes, where each allele at a locus is determined **randomly** (by throwing a random number in [0,1]).

GA Components – Fitness Calculation

```
begin
  generate initial population; // initialise
  calculate fitness values; // evaluate population
  do
  {
    perform reproduction; // select parents
    recombine pairs with pc; // apply crossover
    apply mutation with pm; // mutate
    offspring/children
    calculate fitness values; // eval. population
    replace current population;
  } while termination criteria not satisfied;
end
```



- **Fitness value** indicates
 - ▶ how fit the individual is to survive and reproduce under the current conditions
 - ▶ how much the current solution meets the requirements of the objective function
- **Fitness value** is obtained by applying the fitness function to the individual's chromosome (candidate solution) – *genotype* (e.g., 101110) to *phenotype* (e.g., 1) mapping

Example – MAX-SAT: Fitness Calculation



```
begin
  generate initial population; // initialise
  calculate fitness values; // evaluate population
  do
  {
    perform reproduction; // select parents
    recombine pairs with pc; // apply crossover
    apply mutation with pm; // mutate
    offspring/children
    calculate fitness values; // eval. population
    replace current population;
  } while termination criteria not satisfied;
end
```

0 1 2 3 4 5
 $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$

<i>i</i>	Chromosome <i>abcdef</i>	Unsatisfied clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

GA Components – Reproduction

```
begin
    generate initial population; // initialise
    calculate fitness values; // evaluate population
    do
    {
        perform reproduction; // select parents
        recombine pairs with p_c; // apply crossover
        apply mutation with p_m; // mutate
        offspring/children
        calculate fitness values; // eval. population
        replace current population;
    } while termination criteria not satisfied;
end
```



- Reproduction (Mate Selection) consists of
 - ▶ **selecting** individuals: apply selection pressure considering the fitness of individuals in the population ⇒ e.g., *roulette wheel selection*, *tournament selection*, rank selection, truncation selection, Boltzmann selection, etc.
 - Selection pressure means the individuals with better fitness have higher chance for being selected
 - ▶ usually **2 parents** (individuals/candidate solutions) are selected using the same method, which will go under the crossover operation

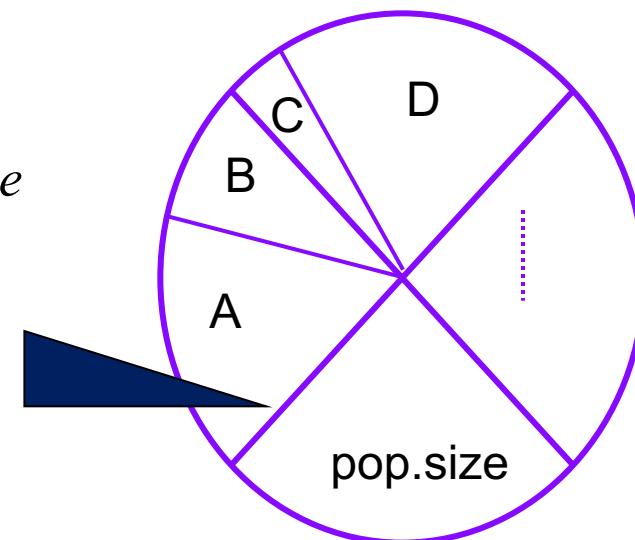
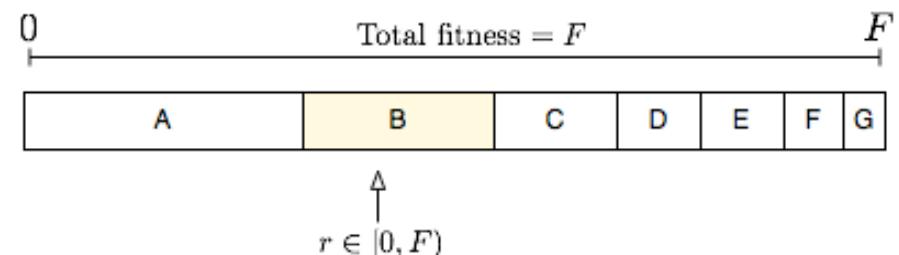
Fitness Proportionate Selection – Roulette Wheel Selection



- Fitness level is used to associate a probability ($prob_i$) of selection with each individual chromosome (i)
- While candidate solutions with a higher fitness will be less likely to be eliminated, there is still a chance that they may be (maximisation problem)
- Expected number of representatives of each individual in the pool is proportional to its fitness (maximisation problem)

maximisation
problem

$$prob_i = \frac{fitness_i}{\sum_j fitness_j} , j = 1 \dots pop.size$$



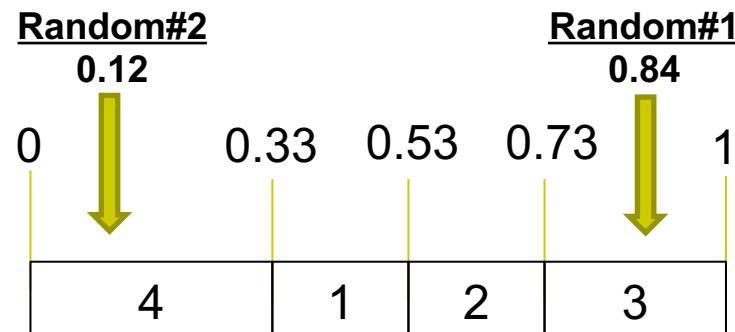
Example: MAX-SAT – Roulette Wheel Selection



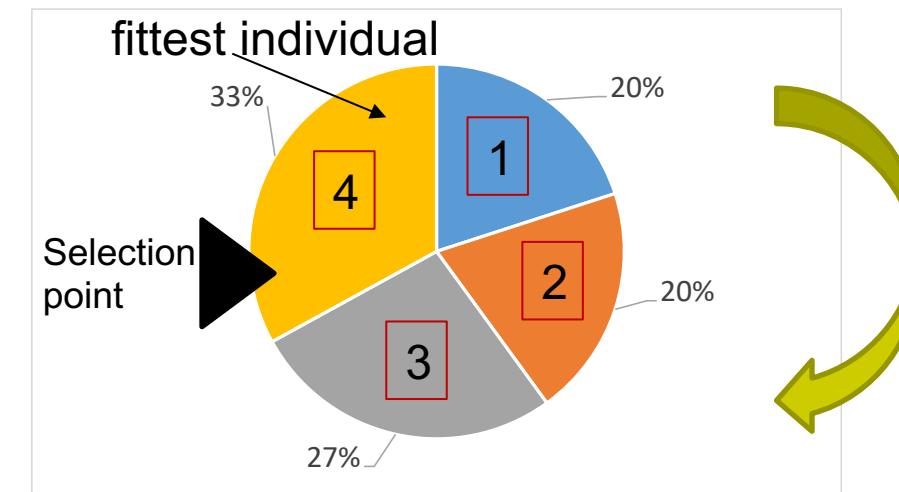
$$0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \\ (a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$$

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness (f _{max} -f)	prob _{<i>i</i>}
1	110000	012345	3 (3)	20%
2	010100	012345	3 (3)	20%
3	001100	012345	2 (4)	27%
4	101110	012345	1 (5)	33%

total:(15)



- Rotate the wheel



Random#1: 0.84 → Parent#1: 3

Random#2: 0.12 → Parent#2: 4



Tournament Selection

- This method involves running a number of "tournaments" among randomly chosen individuals (of tour size) selecting the one with best fitness at the end
 - ▶ This process is repeated for selecting each parent to be recombined

Example – MAX-SAT: Tournament Selection

tour size = 3, first parent

```
begin
    generate initial population; // initialise
    calculate fitness values; // evaluate population
    do {
        perform reproduction; // select parents
        recombine pairs with p_c; // apply crossover
        apply mutation with p_m; // mutate
        offspring/children
        calculate fitness values; // eval. population
        replace current population;
    } while termination criteria not satisfied;
end
```



<i>i</i>	Chromosome <i>abcdef</i>	Violated clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

- Throw a random number between 1 and 4 (population size) for 3 times:
 - ▶ [3, 1, 2]
- Tournament selection chooses 3 individuals: #1, #2 and #3 at random, then individual #3 with the fitness of 2 is returned as the first parent

Example – MAX-SAT: Tournament Selection

tour size = 3, second parent

```
begin
    generate initial population; // initialise
    calculate fitness values; // evaluate population
    do
    {
        perform reproduction; // select parents
        recombine pairs with p_c; // apply crossover
        apply mutation with p_m; // mutate
        offspring/children
        calculate fitness values; // eval. population
        replace current population;
    } while termination criteria not satisfied;
end
```



<i>i</i>	Chromosome <i>abcdef</i>	Violated clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1



- Throw a random number between 1 and 4 (population size) for 3 times:
 - ▶ [4, 1, 3]
- Tournament selection chooses 3 individuals: #1, #3 and #4 at random, then individual #4 with the fitness of 1 is returned as the second parent

Recombination – Crossover

```
begin
    generate initial population; // initialise
    calculate fitness values; // evaluate population
do
{
    perform reproduction; // select parents
    recombine pairs with  $p_c$ ; // apply crossover
    apply mutation with  $P_m$ ; // mutate
    offspring/children
    calculate fitness values; // eval. population
    replace current population;
} while termination criteria not satisfied;
end
```



- Selected pairs/mates (*parents*) are recombined to form new individuals (candidate solutions/children/offspring) – exchange of genetic material
- Crossover is applied with a **crossover probability** p_c which in general is chosen close to 1.0

One Point Crossover (1PTX)

- Generate a random number in $[0,1)$, if it is smaller than a **crossover probability** p_c Then
 - ▶ Select a random crossover site in $[1, \text{chromosome length}]$
 - ▶ Split individuals at the selected site
 - ▶ Exchange segments between pairs to form two new individuals
- Else
 - ▶ Copy the individuals as new individuals

Example – MAX-SAT: 1PTX

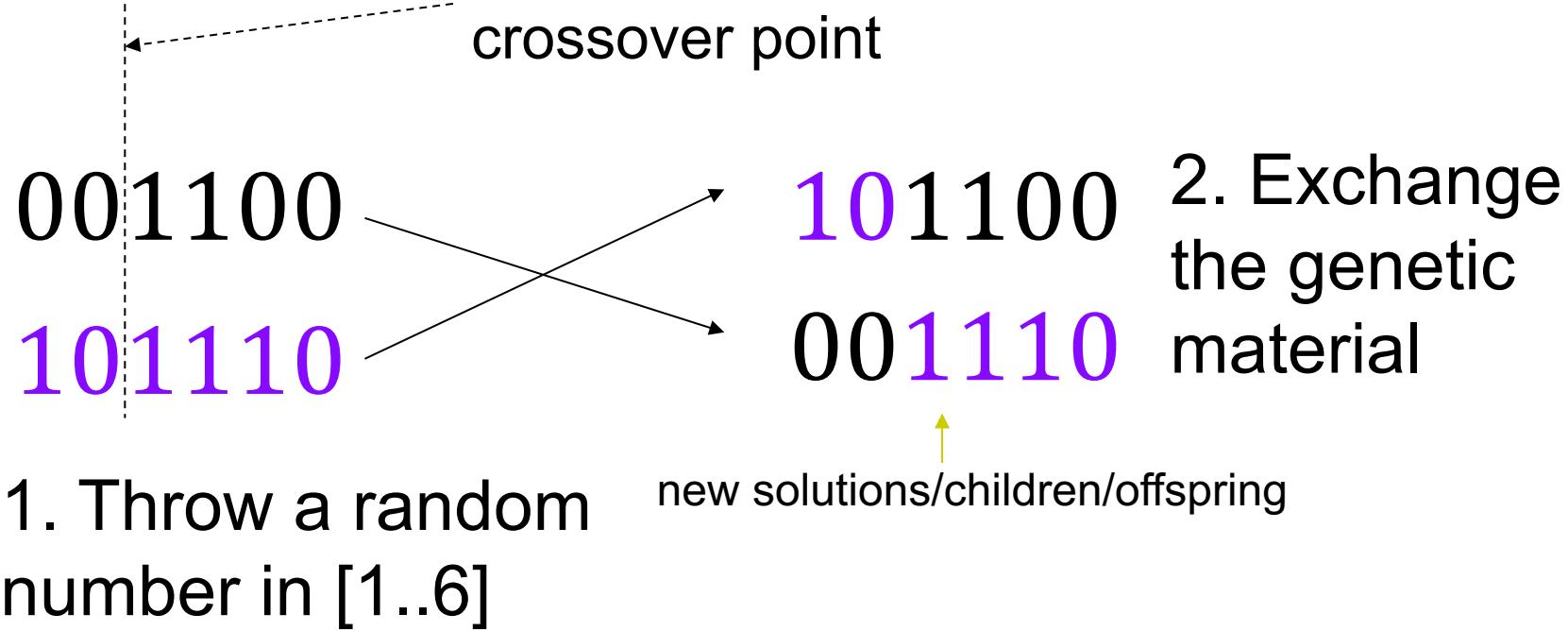
```
begin
  generate initial population; // initialise
  calculate fitness values; // evaluate population
  do
  {
    perform reproduction; // select parents
    recombine pairs with p_c; // apply crossover
    apply mutation with p_m; // mutate
    offspring/children
    calculate fitness values; // eval. population
    replace current population;
  } while termination criteria not satisfied;
end
```



0 1 2 3 4 5
 $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$

Random
number: 2

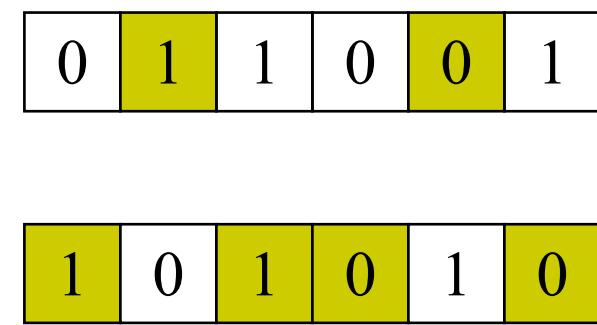
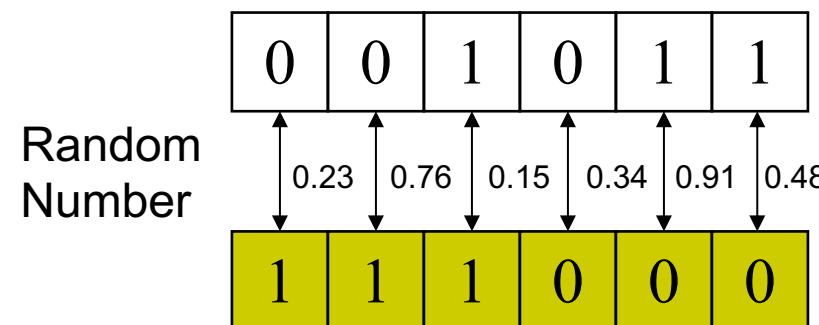
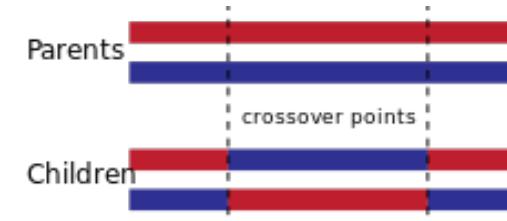
Randomly determined
crossover point





Other Crossover Operators

- 2 Point Crossover (2PTX)
- K-point Crossover
- Uniform Crossover (UX)
 - ▶ The uniform crossover considers each bit in the parent strings for exchange with a probability of 0.5.





Mutation

- Any offspring might be exposed to mutation
- Loop through all the alleles of all the individuals one by one, and if that allele is selected for mutation with a given probability p_m , you can either change it by a small amount or replace it with a new value
 - ▶ For binary representation mutation corresponds to flipping a selected gene value ($0 \rightarrow 1$, $1 \rightarrow 0$)
- Mutation provides diversity and allows GA to explore different regions of the search space (escaping)
- Mutation rate is typically chosen to be very small (0.001, 0.001). Choosing p_m as $(1/\text{chromosome length})$ implies on average a single gene will be mutated for an individual.

Example – Mutation



1	0	1	1	0	0
---	---	---	---	---	---

0	0	1	1	1	0
---	---	---	---	---	---

```
begin
    generate initial_population; // initialise
    calculate fitness values; // evaluate population
    do
    {
        perform reproduction; // select parents
        recombine pairs with p_c; // apply crossover
        apply mutation with p_m; // mutate
        offspring/children
        calculate fitness values; // eval. population
        replace current population;
    } while termination criteria not satisfied;
end
```

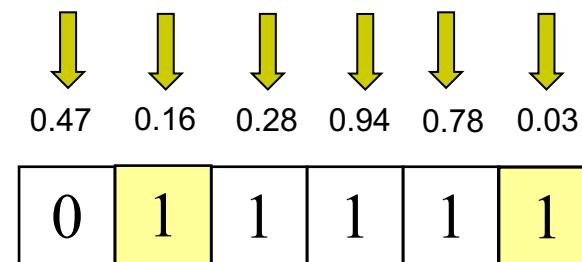
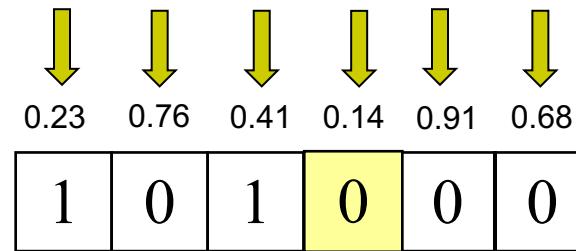
- A loop is performed on each individual
 - If a random value in $[0,1)$ is $< p_m = 0.17$ ($1/6$), then the allele value is flipped, otherwise kept same.



Example – Mutation

Random numbers:

<0.23 0.76 0.41 0.14 0.91 0.68 0.47 0.16 0.28 0.94 0.78 0.03 ...>



- A loop is performed on each individual

- If a random value in $[0,1)$ is $< p_m = 0.17$ ($1/6$), then the allele value is flipped, otherwise kept same.

Replacement Strategy



```
begin
    generate initial population; // initialise
    calculate fitness values; // evaluate population
    do
    {
        perform reproduction; // select parents
        recombine pairs with p_c; // apply crossover
        apply mutation with p_m; // mutate
        offspring/children
        calculate fitness values; // eval. population
        replace current population;
    } while termination criteria not satisfied;
end
```

- There are variety of strategies for replacing the old population (generation) by the new (offspring) population to form the next generation
- **Generation gap (α)** controls the fraction of the population to be replaced in each generation, where $\alpha \in [1/N, 1.0]$
 - ▶ Number of offspring produced at each generation is $g = \alpha^* N$
- **(Trans-)Generational GA** ($g > 2$, that is $\alpha > 2/N$)
 N individuals produce αN offspring, so $(N + \alpha N) \rightarrow N$
 - ▶ αN replaces worst αN of N
 - largest *generation gap* where $\alpha=1.0$ yields $g=N$.
 - GA relies on improvement of average objective values from one population to another
 - It is always a good idea not to lose the best solution found so far.
 - ▶ sort $(N + \alpha N)$ and choose the N best (elitism)

Replacement Strategy

- ***Steady-State GA (g=2, that is $\alpha=2/N$)***

Two offspring replace two individuals from the old generation.

- ▶ Method#1: two offspring replace two parents
- ▶ Method#2: two offspring replace worst two of the population
- ▶ Method#3: best two of (parents and offspring) replace two parents (elitism)
- ▶ Method#4: best two of (parents and offspring) replace worst two of the population (strong elitism)

```
begin
    generate initial_population; // initialise
    calculate fitness values; // evaluate population
do
{
    perform reproduction; // select parents
    recombine pairs with pc; // apply crossover
    apply mutation with pm; // mutate
    offspring/children
    calculate fitness values; // eval. population
    replace current population;
} while termination criteria not satisfied;
end
```



Example – Transgenerational GA Replacement (no elitism)



$$\begin{array}{ccccccc}
 & 0 & & 1 & & 2 & \\
 & (a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) & \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e) & & & & 3 \\
 & & & & & & 4 \\
 & & & & & & 5
 \end{array}$$

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	100100	012345	3
2	010100	012345	3
3	101000	012345	0
4	011111	012345	0

New Population/
Generation

Form the new
generation by
copying
offspring
onto the old
population,
forming the
new
population
($g=N=4$)

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	100100	012345	3
2	010100	012345	3
3	101000	012345	0
4	011111	012345	0

Offspring

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

Old Population

Example – TGA Replacement (with elitism)



$$\begin{array}{ccccccc}
 & 0 & & 1 & & 2 & \\
 & (a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) & \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e) & & & & 5
 \end{array}$$

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	001100	012345	2
2	101110	012345	1
3	101000	012345	0
4	011111	012345	0

New Population/
Generation

Form the new generation by

Selecting the best 4 individuals among both old population and offspring ($g=N=4$)

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	100100	012345	3
2	010100	012345	3
3	101000	012345	0
4	011111	012345	0

Offspring

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

Old Population

Example – A Steady State GA (with elitism)



$$\begin{array}{ccccccc}
 & 0 & & 1 & & 2 & \\
 & (a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e) & & & & & 5
 \end{array}$$

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	010100	012345	3
2	101000	012345	0
3	001100	012345	2
4	101110	012345	1

New Population/
Generation

Next
generation

Using
Method#2:
Replace the
worst 2 of the
population
with 2
offspring
($g=2$)

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	010100	012345	3
2	101000	012345	0

Offspring

<i>i</i>	Chromosome <i>abcdef</i>	Unsat. clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

Old Population

```

begin
generate initial population; // initialise
calculate fitness values; // evaluate population
do
{
    perform reproduction; // select parents
    recombine pairs with pc; // apply crossover
    apply mutation with pm; // mutate
    offspring/children
    calculate fitness values; // eval. population
    replace current population;
} while termination criteria not satisfied;
end

```



Termination Criteria

- The evolution (main loop) continues until a termination criteria is met, possibly until:
 - ▶ A predefined maximum number of generations is exceeded
 - ▶ A goal is reached, for example:
 - Expected fitness is achieved
 - Population converges
 - ▶ Best fitness does not change for a while
 - ▶ A condition is satisfied depending on a combination of above



Convergence

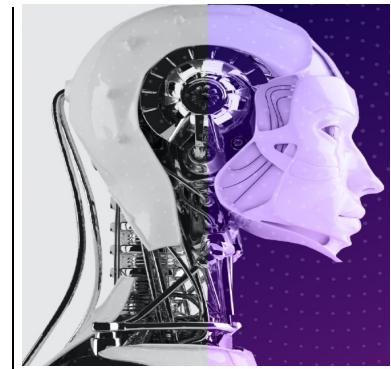
- Defined as the progression towards uniformity (individuals become alike)
 - ▶ **Gene convergence:** a location on a chromosome is converged when 95% of the individuals have the same gene value for that location
 - ▶ **Population (Genotypic) convergence:** a population is converged when all the genes have converged (all individuals are alike – they might have different fitness)
 - ▶ **Phenotypic Convergence:** average fitness of the population approaches to the best individual in the population (all individuals have the same fitness)



Key Features of EAs

- Population based search approaches
 - ▶ Be independent of initial starting point(s)
 - Start search from many points in the search space
 - ▶ Conduct search in parallel over the search space
 - implicit parallelism
- Avoid converging to local optima
- Balances exploration and exploitation?
- May be used together with other approaches (hybrids)

Memetic Algorithms





Memetic Algorithms (MAs)

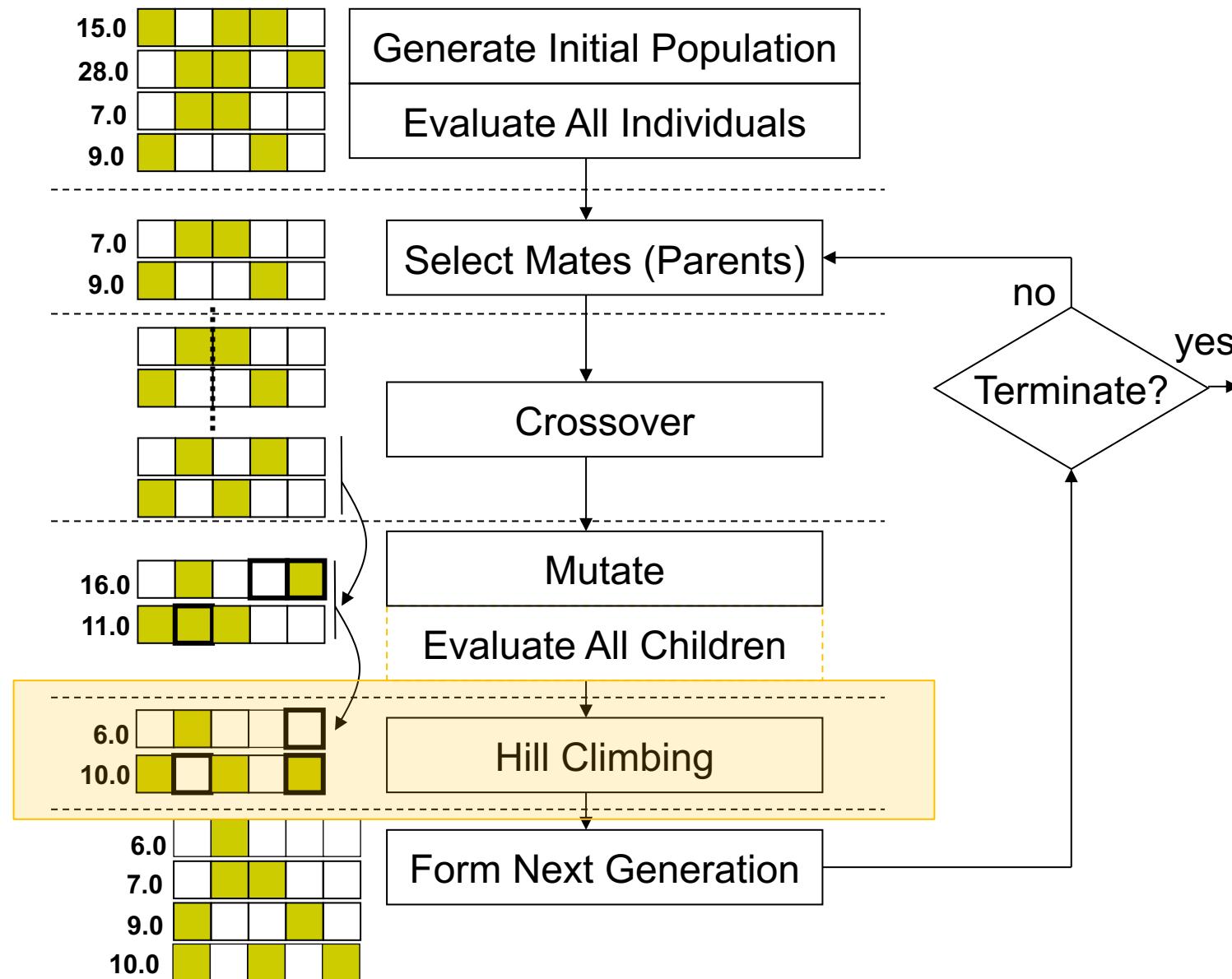
- Meme (Richard Dawkins): contagious piece of information
- Memes are similar to local refinement/local search
- Gene vs meme: Memes can change, evolve using rules and time scales other than the traditional genetic ones
- MAs aim to improve GAs by embedding local search



Memetic Algorithms (MAs) II

- MAs make use of exploration capabilities of GAs and exploitation capabilities of local search
 - ▶ MAs have an explicit mechanism to balance exploitation and exploration
- Memetic Algorithms shown to be much faster and more accurate than GAs on some problems, and are the “state of the art” on many problems

A Generic Memetic Algorithm (MA)





Memetic Algorithms (MAs)

Pseudocode of memetic algorithm

```
→ CreateInitialSolutions(); // create initial population of solutions

repeat
    → SelectParents(); // select solutions from the population to breed
    → Crossover(); // apply crossover operator with a given probability
    → Mutate(); // apply mutation operator with a given probability
    → LocalSearch();
    → Evaluate();
    → ReplaceSolutions(); // generate new population of solutions
until TerminationCriteriaSatisfied();
```

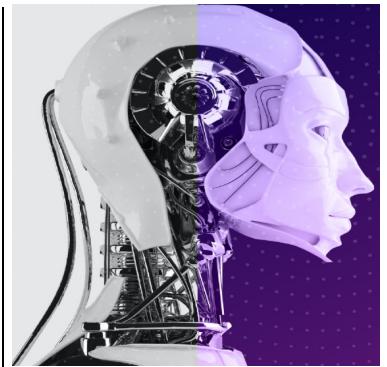
Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, Caltech Concurrent Computation Program Report 826, California Institute of Technology (1989)

Example – Designing a Genetic/Memetic Algorithm for MAX-SAT



- *Representation:* Bit string of length N (truth assignment for each variable)
- *Initialisation:* Randomly generate initial population, population size= N
- *Fitness function:* (<# of clauses – C); # of unsatisfied clauses
- *Mate selection:* Tournament selection with a tour size of 2
- *Crossover:* 1PTX, crossover probability = 0.99
- *Mutation:* random bit-flip, mutation probability = $1/N$
- **Hill Climbing: Davis's Bit Hill Climbing**
- *Replacement:* Steady State GA, best two of (parents and offspring) replaces the worst two individuals in the population (strong elitism)

Early Module Feedback



Q&A



Thank you.

Ender Özcan ender.ozcan@nottingham.ac.uk

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
<http://www.nottingham.ac.uk/~pszeo/>

Evolutionary Algorithms II



Ender Özcan

Lecture 6



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA



Questions

- When does a genetic algorithm perform better than a memetic algorithm?
- Is the choice of **meme** (hill climbing/local search algorithm) important?
- Which meme does have a better performance in a memetic algorithm?
- Can we somehow combine several memes to obtain a synergy leading to improved performance?

Benchmark Functions



Why to use benchmark (test) functions for optimisation



- Benchmark functions serve as a testbed for performance comparison of (meta/hyper)heuristic optimisation algorithms
 - ▶ Their **global minimum** are **known**
 - ▶ They can be **easily computed**
 - ▶ Each function is recognised to have certain characteristics potentially representing a different real-world problem
 - E.g., separable vs non-separable
- CComparing Continuous Optimisers (COCO) provide benchmark function testbeds: <http://coco.gforge.inria.fr/>

Classification of benchmark (test) functions



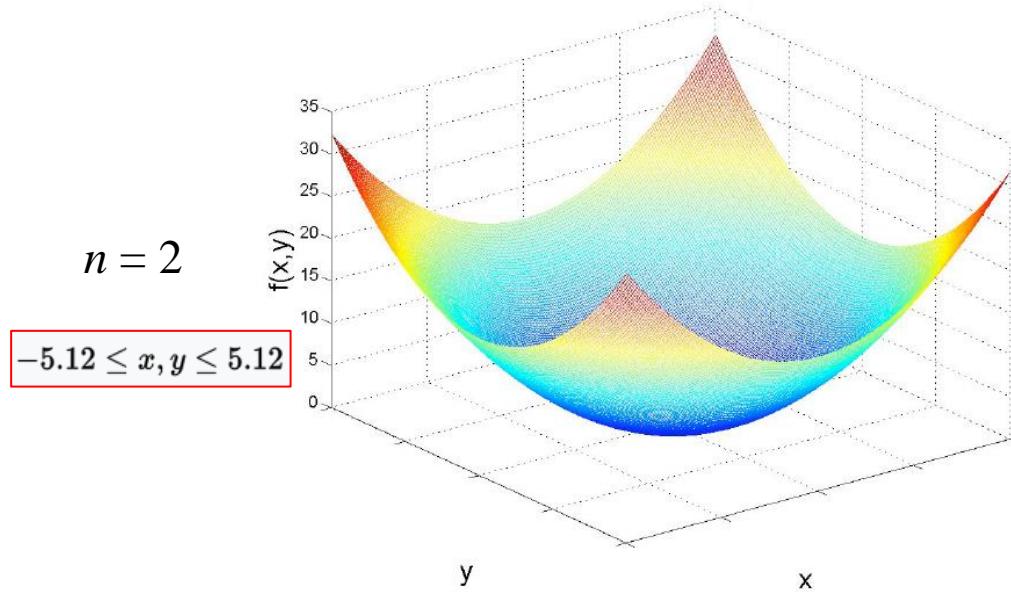
- A common classification is:
 - ▶ **Continuity (Differentiability)**
 - Discontinuous vs continuous
 - ▶ **Dimensionality**
 - Scalability
 - ▶ **Separability**
$$\arg \underset{x_1, \dots, x_p}{\text{minimize}} f(x_1, \dots, x_p) = (\arg \underset{x_1}{\text{minimize}} f(x_1, \dots), \dots, \arg \underset{x_p}{\text{minimize}} f(\dots, x_p))$$
 - ▶ **Modality**
 - Unimodal
 - Multimodal with few local minima
 - Multimodal with exponential number of local minima

Example – unimodal function: Sphere function



$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$$

$$f(x_1, \dots, x_n) = f(0, \dots, 0) = 0$$



- continuous
- differentiable
- separable
- scalable

Delta Evaluation in Function Optimisation



- Separable functions allows delta evaluation
- Example, sphere function

$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$$

$$\underbrace{x_1 \dots 6 \dots x_n}_{\underline{s}} \Rightarrow \underbrace{x_1 \dots 5 \dots x_n}_{\underline{s'}}$$

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

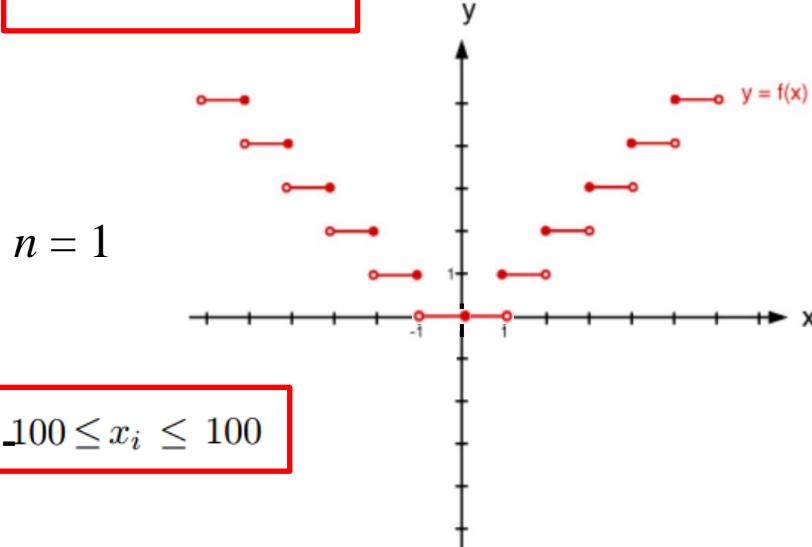
$$(5^2 - 6^2)$$

Example – unimodal function: Step function



$$f(\mathbf{x}) = \sum_{i=1}^n (\lfloor |x_i| \rfloor)$$

$$f(x_1, \dots, x_n) = f(0, \dots, 0) = 0$$



- discontinuous
- non-differentiable
- separable
- scalable

Example – multimodal function

Rastrigin's function



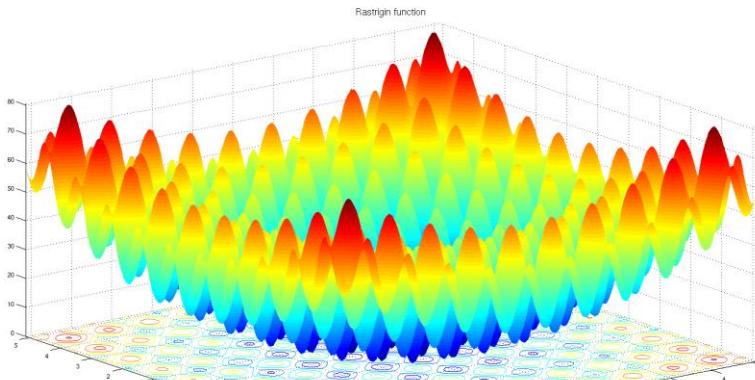
$$f(\mathbf{x}) = An + \sum_{i=1}^n [x_i^2 - A \cos(2\pi x_i)]$$

where: $A = 10$

$n = 2$

$$f(0, 0) = 0$$

$$-5.12 \leq x, y \leq 5.12$$



- continuous
- differentiable
- separable
- scalable

Example – multimodal function Ackley's function

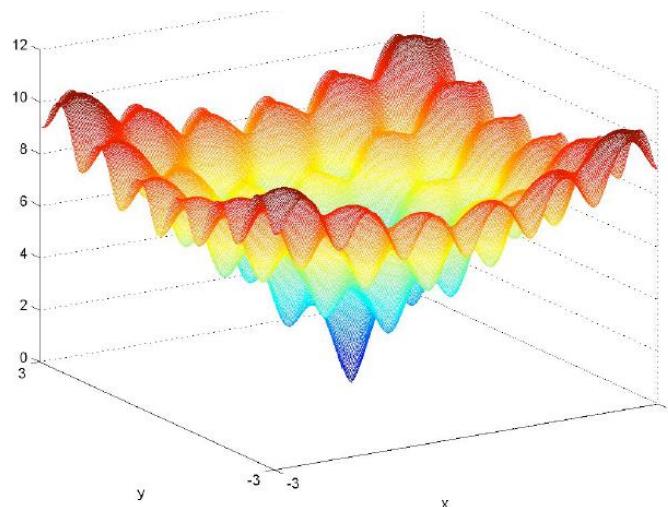


$$f(x, y) = -20 \exp\left(-0.2\sqrt{0.5(x^2 + y^2)}\right)$$
$$- \exp(0.5(\cos(2\pi x) + \cos(2\pi y))) + e + 20$$

$n = 2$

$$f(0, 0) = 0$$

$$-5 \leq x, y \leq 5$$



- continuous
- differentiable
- non-separable
- non-scalable

Example – Sphere Function Optimisation



- **Objective:**
- Find the set of integers x_i which maximize the function f given below.

$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2 \quad \Rightarrow \text{for } n=3, f(x) = x_1^2 + x_2^2 + x_3^2$$

↑
Fitness
function

$-512 < x_i \leq 512$

Sphere Function Optimisation – Representation I



- There are 1024 integers in the given interval
 $-512 < x_i \leq 512$
- In binary representation, 1024 different integers can be represented using 10 bits

Encoding of x_i

0	:	0000000000	(-511)
1	:	0000000001	(-510)
...	...		
1023	:	1111111111	(512)

Decoding of x_i

- Convert binary # into decimal,
e.g., $x_i = (0000000011) = 3$
- Subtract 511 from that value,
e.g., $3-511 = -509$

Sphere Function Optimisation – Representation II



- Function has 3 parameters: x_1, x_2, x_3
- Each parameter can be represented by 10 bits
- Chromosome consists of 30 bits
- Example

11001010101100000000000000000000110

x_1 x_2 x_3

Sphere Function Optimisation – Fitness Evaluation (Decoding)



11001010101100000000000000000000110
 x_1 x_2 x_3

$$f(x) = x_1^2 + x_2^2 + x_3^2$$

$$x_1 = (\textcolor{red}{810} - 511) = 299$$

$$x_2 = (\textcolor{purple}{768} - 511) = 257$$

$$x_3 = (\textcolor{green}{6} - 511) = -505$$

Fitness: $f(x) = (299)^2 + (257)^2 + (-505)^2$
 $= 410425$



More on Function Optimization

- What if x_i were real numbers in the interval
 $-5.12 < x_i \leq 5.12$
- Solution:
 - ▶ Use binary representation
 - with precision of 2 digits after decimal point
 - use 1024 different numbers
 - divide number by 100
 - ▶ Use other representation (e.g. real)
- What if the representation has redundancy?
e.g. $-5.40 < x_i \leq 5.40$

Case Studies:

Benchmark Function Optimisation

Travelling Salesman Problem





Ender Özcan, Burak Bilgin, Emin Erkan Korkmaz, A Comprehensive Analysis of Hyper-heuristics, Intelligent Data Analysis, 12:1, pp. 3-23, 2008. [\[PDF\]](#)

BENCHMARK FUNCTION OPTIMISATION

Benchmark Function

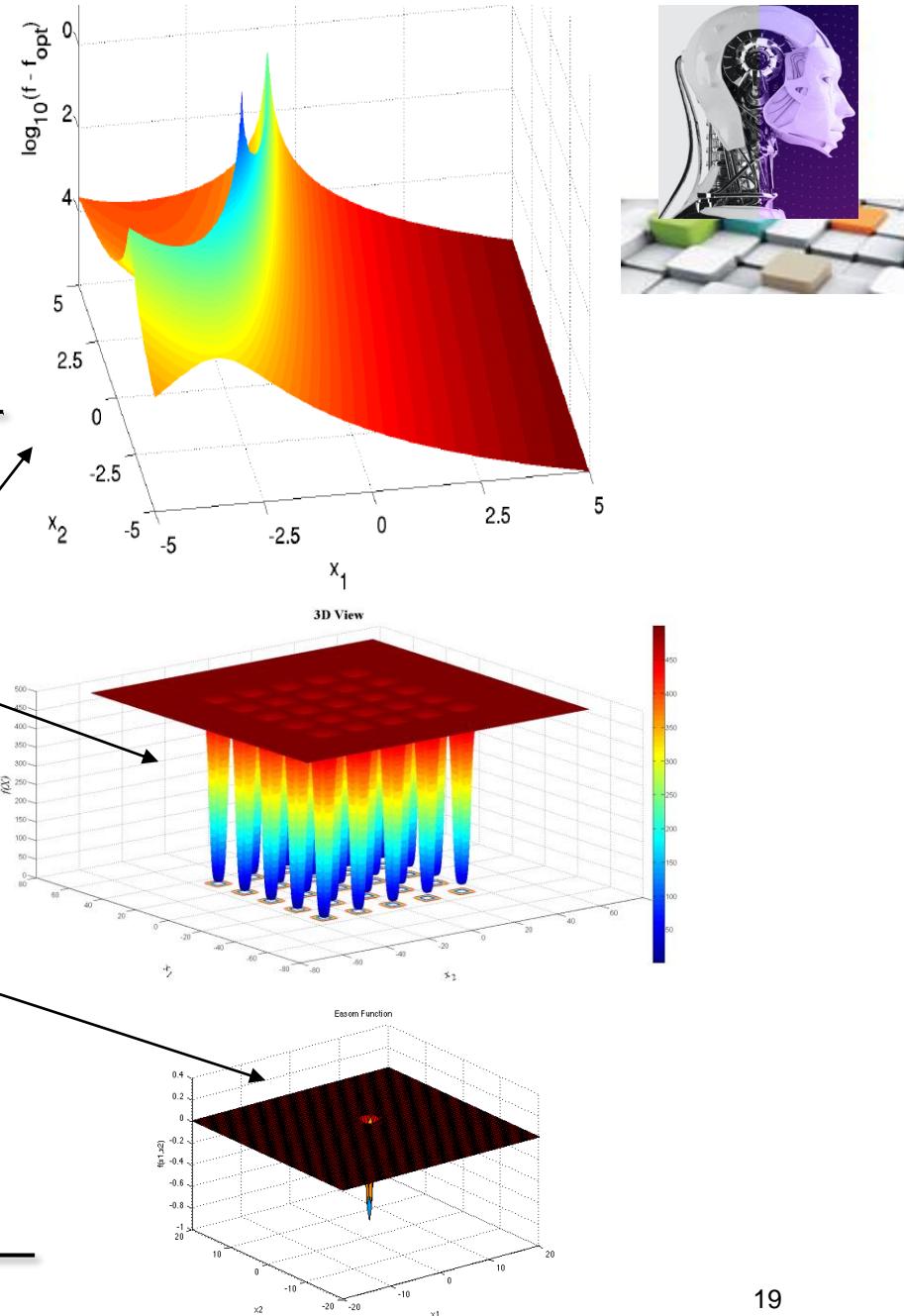
Optimi



Label	Function name	lb	ub	opt	isContinuous	isSeparable	isMultimodal
F1	Sphere	-5.12	5.12	0	yes	yes	no
F2	Rosenbrock	-2.048	2.048	0	yes	yes	no
F3	Step	-5.12	5.12	0	yes	yes	no
F4	Quartic <i>with noise</i>	-1.28	1.28	1	yes	yes	yes
F5	Foxhole	-65.536	65.536	1	yes	no	yes
F6	Rastrigin	-5.12	5.12	0	yes	yes	yes
F7	Schwefel	-500	500	0	yes	yes	yes
F8	Griewangk	-600	600	0	yes	no	yes
F9	Ackley	-32.768	32.768	0	yes	no	yes
F10	Easom	-100	100	-1	yes	no	no
F11	Schwefel's Double Sum	-65.536	65.536	0	yes	no	no
F12	Royal Road	—	—	0	no	yes	n/a
F13	Goldberg	—	—	0	no	yes	n/a
F14	Whitley	—	—	0	no	yes	n/a

Benchmark Function Optimisation

Label	Function name	lb	ub
F1	Sphere	-5.12	5.12
F2	Rosenbrock	-2.048	2.048
F3	Step	-5.12	5.12
F4	Quartic <i>with noise</i>	-1.28	1.28
F5	<u>Foxhole</u>	-65.536	65.536
F6	Rastrigin	-5.12	5.12
F7	Schwefel	-500	500
F8	Griewangk	-600	600
F9	Ackley	-32.768	32.768
F10	<u>Easom</u>	-100	100
F11	Schwefel's Double Sum	-65.536	65.536
F12	<u>Royal Road</u>	$s_1 = 11111111*****; s_2 = *****1111111*****; s_3 = *****1111111*****; s_4 = *****1111111*****; s_5 = *****1111111*****; s_6 = *****1111111*****; s_7 = *****1111111*****; s_8 = *****1111111*****;$	
F13	Goldberg		
F14	Whitley		





Binary vs Gray Encoding

- Gray encoding ensures a Hamming distance of 1 for the adjacent numbers
- Shown to be useful in GAs empowering the algorithm to mutate a solution in the right direction

Dec	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Components of Evolutionary Algorithms and Settings



<i>Label</i>	<i>dim</i>	<i>bits</i>	<i>Chrom. Len.</i>	<i>Pop. size</i>	<i>Max. hcteps</i>
F1	10	30	300	60	600
F2	10	30	300	60	600
F3	10	30	300	60	600
F4	10	30	300	60	600
F5	2	30	60	20	120
F6	10	30	300	60	600
F7	10	30	300	60	600
F8	10	30	300	60	600
F9	10	30	300	60	600
F10	6	30	180	36	360
F11	10	30	300	60	600
F12	8	8	64	20	128
F13	30	3	90	20	180
F14	6	4	24	20	48

- Representation: Gray encoding
- Initialisation: random
- Mate selection: Tournament with tour size of 2
- Crossover: 1PTX ($p_c=1.0$)
- Traditional mutation based on bit-flip with mutation rate $1/(2 \times \text{chromosome_length})$
- A trans-generational EA with a replacement method which keeps only two best individuals from the previous generation.

Components of Evolutionary Algorithms and Settings II



<i>Label</i>	<i>dim</i>	<i>bits</i>	<i>Chrom.</i> <i>Len.</i>	<i>Pop.</i> <i>size</i>	<i>Max.</i> <i>hcteps</i>
F1	10	30	300	60	600
F2	10	30	300	60	600
F3	10	30	300	60	600
F4	10	30	300	60	600
F5	2	30	60	20	120
F6	10	30	300	60	600
F7	10	30	300	60	600
F8	10	30	300	60	600
F9	10	30	300	60	600
F10	6	30	180	36	360
F11	10	30	300	60	600
F12	8	8	64	20	128
F13	30	3	90	20	180
F14	6	4	24	20	48

- Memes (using bit-flip):
 - GA: Genetic Algorithm (none)
 - MA0: MA with SDHC (best imp.)
 - MA1: MA with NDHC (first imp.)
 - MA2: MA with RMHC
 - MA3: MA with DBHC
- Expectedly, poor performing memes (using biased moves) are designed:
 - MA4: SDHC \Rightarrow neighborhood move: AND with 0
 - MA5: SDHC \Rightarrow neighborhood move: OR with 1
 - MA6: NDHC \Rightarrow neighborhood move: AND with 0
 - MA7: NDHC \Rightarrow neighborhood move: OR with 1

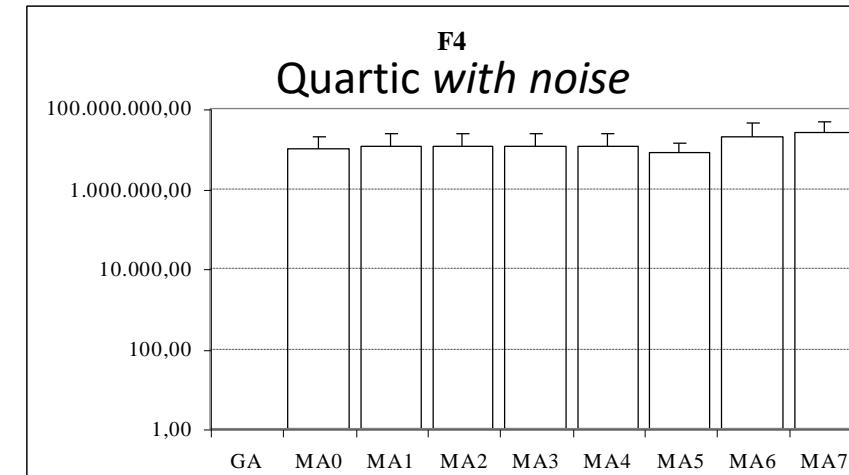
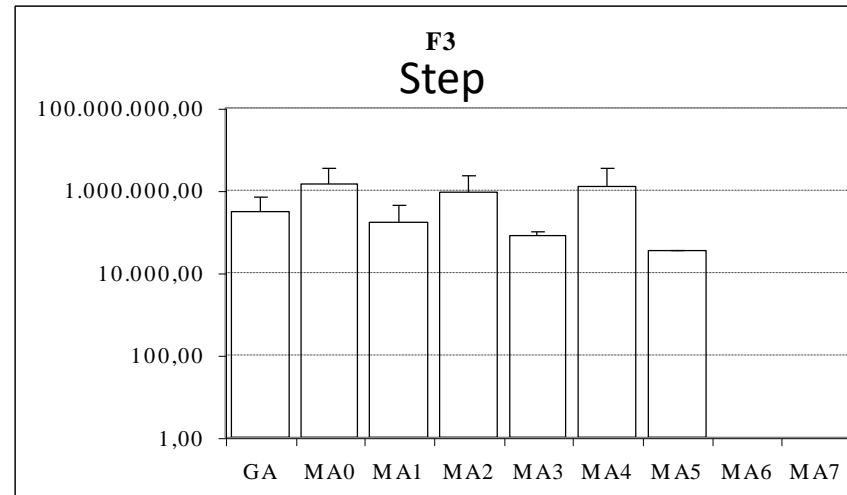
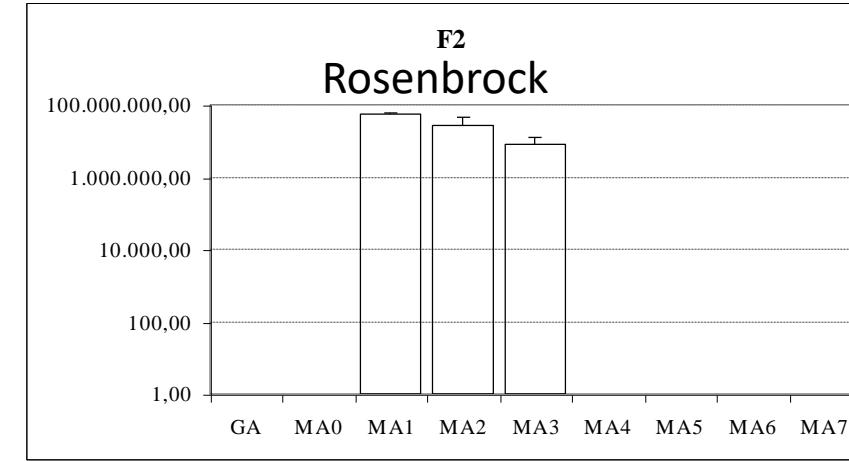
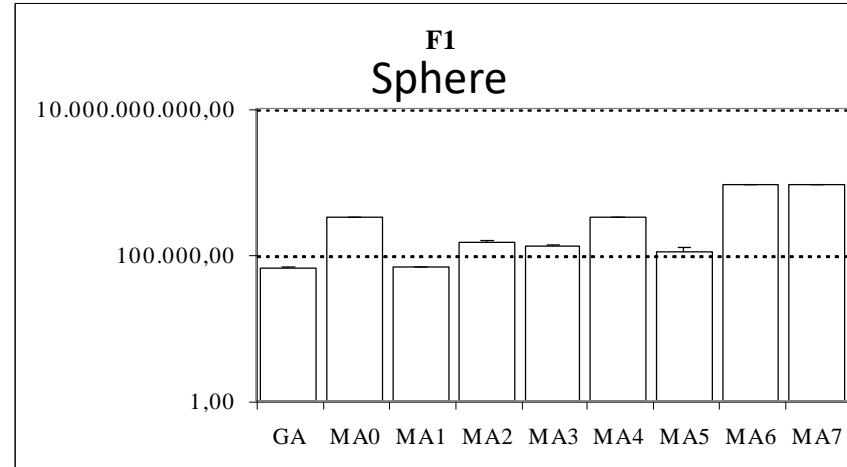
Termination and Performance Comparison Criteria



- Termination: Runs are terminated whenever the overall CPU time exceeds 600 sec., or an expected fitness (optimum) is achieved.
 - ▶ Pentium IV 2 GHz. machines with 256 MB RAM are used
- All runs are repeated 50 times.
- Performance indicators:
 - ▶ Success rate (effectiveness): The ratio of the number of runs returning the expected optimal solution to the total number of runs (50)
 - ▶ Average number of evaluations/configurations (efficiency)
 - ▶ Bar chart plots showing the average no. of evaluations in log scale for each algorithm, if the success rate is 100% (all 50 runs yield expected optimum), otherwise no bar is drawn.

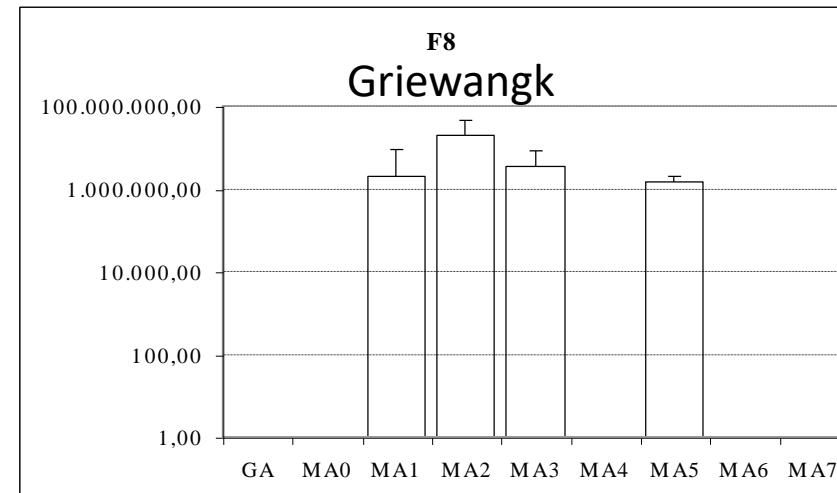
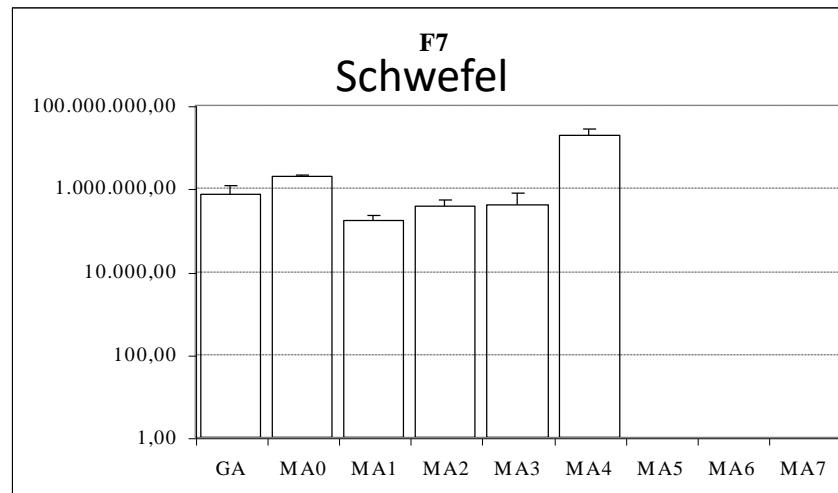
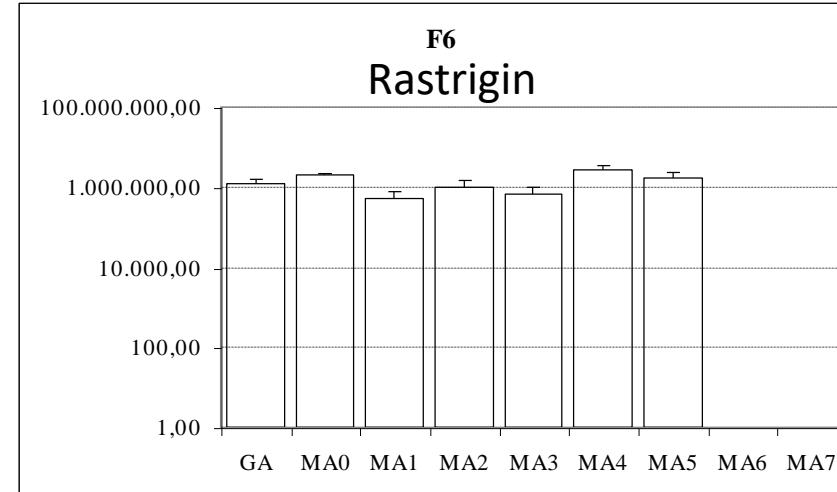
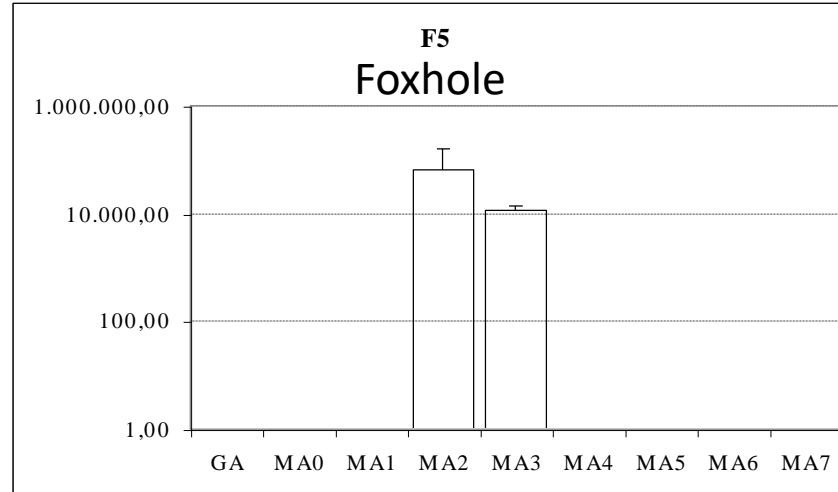


Results



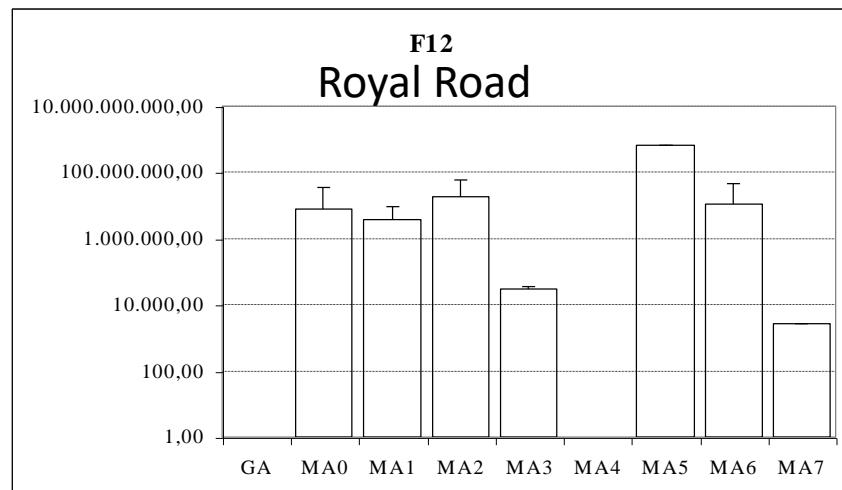
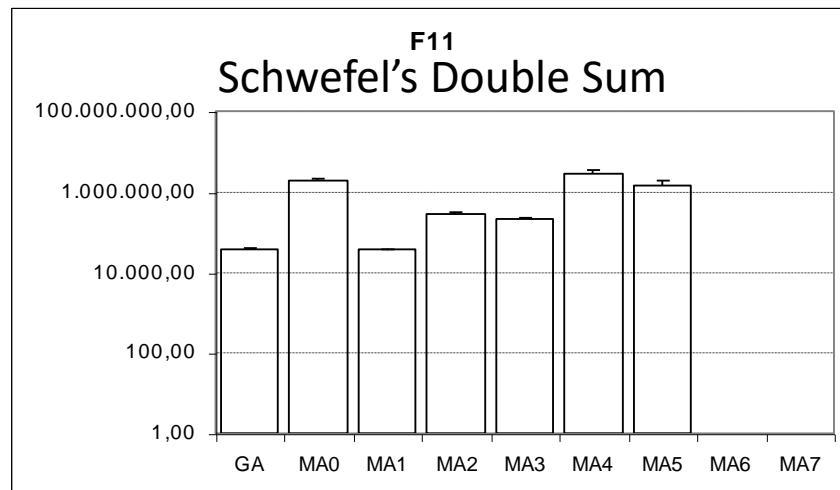
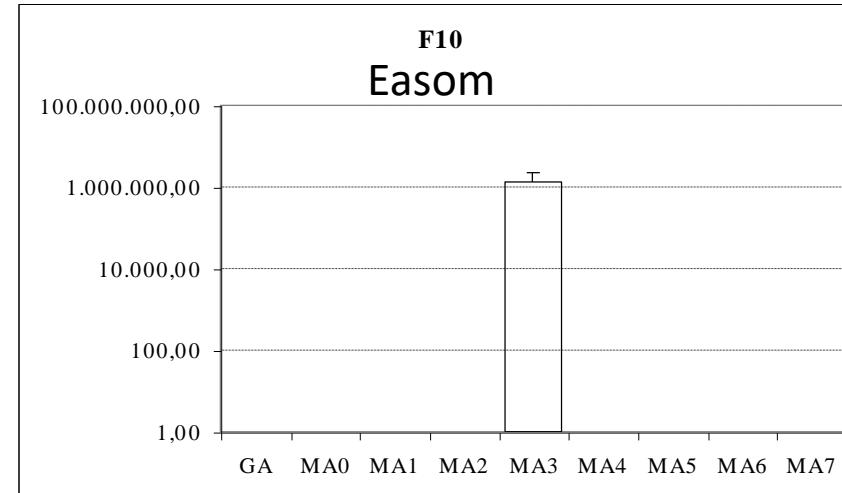
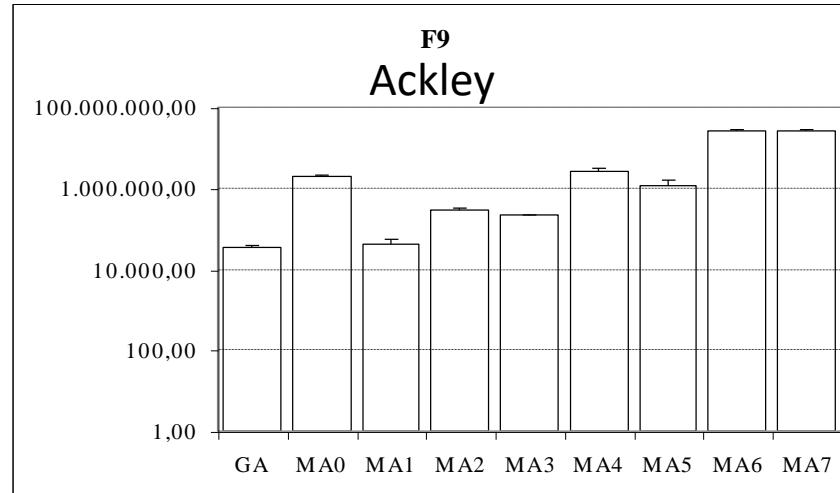


Results II

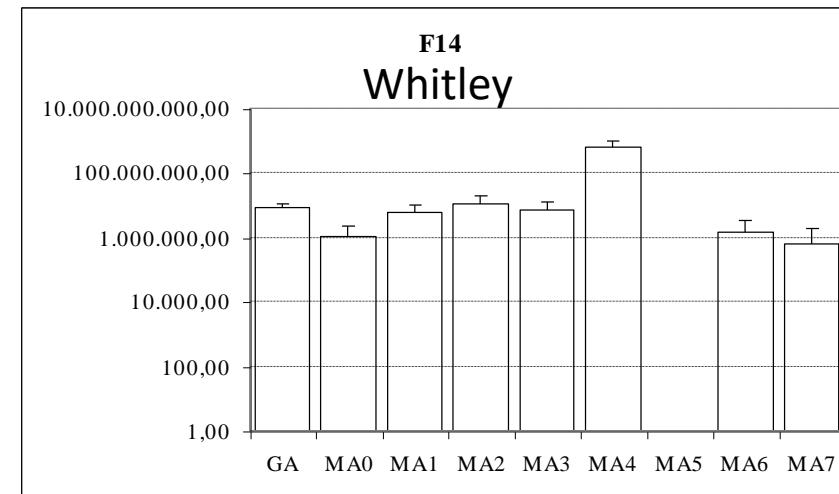
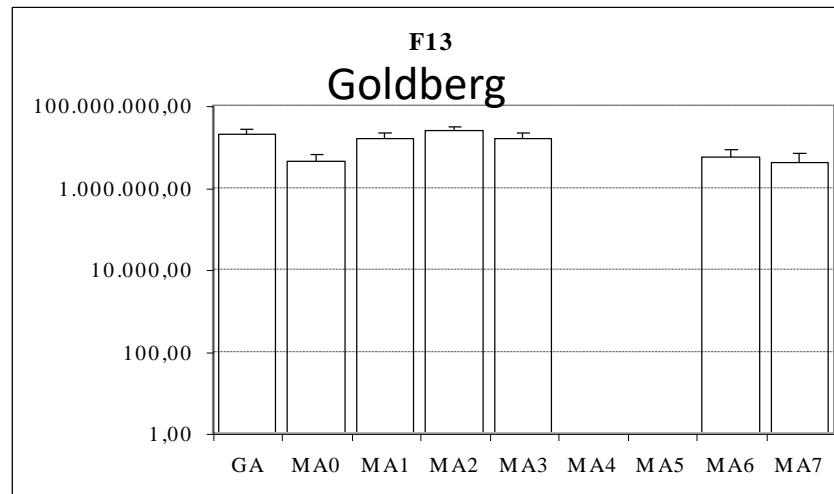




Results III



Results IV





Summary

- MA0 (SDHC) is the best meme choice for F4, F13 and F14. (noisy + deceptive)
- MA1 (NDHC) is the best meme choice for F6-F8. (multimodal)
- MA3 is (DBHC) the best meme choice for F2, F3, F5, F10, F12. (functions with plateaus)
- For functions F1, F11 (unimodal) and F9, GA performs slightly better than the memetic algorithm with the meme MA1 (NDHC). The performance difference is insignificant.
- In the overall, MA2 (RMHC) and MA3 (DBHC) turn out to be the worst and the best meme, respectively, among MA0-MA3, considering the success rates and average number of evaluations.
- Different memes yield different performances, designing the right meme for the problem in hand is important.



E. Özcan, and M. Erenturk, A Brief Review of Memetic Algorithms for Solving Euclidean 2D Traveling Salesrep Problem, Proc. of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks, pp. 99-108, June 2004. [[PDF](#)]

TRAVELLING SALESMAN PROBLEM

Binary Representation for Encoding Permutation



- Binary representation and classical random initialisation, cross-over and mutation operators are not suitable for encoding permutation
 - ▶ E.g., for TSP, could cause illegal tours to form
 - not all cities visited
 - undefined city codes
 - cities visited more than once
 - loops formed within the tour
 - ▶ needs repair algorithms



Example

Given $N=5$ cities \Rightarrow 3 bits per city,

2 5 3 1 4

individual 1: 0|10101011001100

5 2 1 4 3

individual 2: 1|01010001100011

2 | 5 3 1 4

5 | 2 1 4 3

After applying 1-point crossover:

1 2 1 4 3

2 2 1 4 3

individual 1: 001010001100011

6 5 3 1 2

5 5 3 1 4

individual 2: 110101011001100

Generic Permutation based Genetic Operators





Partially Mapped Crossover (PMX)

- Builds offspring by
 - ▶ choosing a subsequence of a tour from one parent
 - choose two random cut points to serve as swapping boundaries
 - swap segments between cut points
 - ▶ preserving the order and position of as many cities as possible from other parent
- Exploits important similarities in the value and ordering simultaneously



Example:

p1: (1 2 3 | 4 5 6 7 | 8 9)

p2: (4 5 2 | 1 8 7 6 | 9 3)

step 1: swap segments

o1: (x x x | 1 8 7 6 | x x)

o2: (x x x | 4 5 6 7 | x x)

this also defines mappings:

1↔4, 8↔5, 7↔6, 6↔7

step 2: fill in cities from other parents if no conflict

o1: (x 2 3 | 1 8 7 6 | x 9)

o2: (x x 2 | 4 5 6 7 | 9 3)

step 3: use mappings for conflicted positions

o1: (**4** 2 3 | 1 8 7 6 | **5** 9)

o2: (**1** 8 2 | 4 5 6 7 | 9 3)



Order Crossover (OX)

- Builds offspring by
 - ▶ choosing a subsequence of a tour from one parent
 - ▶ preserving relative order of cities from other parent
- Exploits the property that ordering of cities important, not positions

9-3-4-5-2-1-8-7-6 and

4-5-2-1-8-7-6-9-3 are identical



Example:

p1: (1 2 3 | 4 5 6 7 | 8 9)

p2: (4 5 2 | 1 8 7 6 | 9 3)

step 1: copy segments into offspring

o1: (x x x | 4 5 6 7 | x x)

o2: (x x x | 1 8 7 6 | x x)

step 2: starting from 2nd cut point of one parent, cities from other parent copied in same order, omitting symbols already present; if end of string reached, continue from beginning of string

sequence of cities in 2nd parent (from 2nd cut point) is

9-3-4-5-2-1-8-7-6 (remove 4,5,6,7 which are in 1st offspring)

9-3-2-1-8

place into first offspring o1: (2 1 8 | 4 5 6 7 | 9 3)

similarly the 2nd offspring o2: (3 4 5 | 1 8 7 6 | 9 2)



Cycle Crossover (CX)

- Builds offspring by
 - ▶ choosing each city and its position from one of the parents
 - ▶ and when a cycle is completed, the remaining cities filled in from the other parent
- Preserves absolute positions of the elements in the parent sequence



Cycle Crossover (CX)

p1: (1 2 3 4 5 6 7 8 9)
p2: (4 1 2 8 7 6 9 3 5)

o1: (1 x x x x x x x)
o1: (1 x x 4 x x x x x)
o1: (1 x x 4 x x x 8 x)
o1: (1 x 3 4 x x x 8 x)
o1: (1 2 3 4 x x x 8 x)

Randomly select a starting point(city) in p1: 1

Copy cities from p1 until a cycle is obtained while mapping from p1 to each corresponding city in p2

2 requires selection of 1 causing a cycle, so rest filled from other parent:

o1: (1 2 3 4 7 6 9 8 5)

similarly

o2: (4 1 2 8 5 6 7 3 9)

More Genetic Operators – Crossover for TSP



Operator Name	Date	Authors
Alternating Position Crossover (AP)	(1999)	Larranaga, Kuijpers, Poza, Murga
Cycle Crossover (CX)	(1987)	Oliver, Smith and Holland
Distance Preserving Crossover (DPX)	(1996)	Freisbein and Merz
Edge Assembly Crossover (EAX)	(1997)	Nagata and Kobayashi
Edge Recombination Crossover (ER)	(1989)	Whitley, Timothy, Fuquay
Heuristic Crossover (HEU)	(1987)	Grefenstette
Inver-over Operator (IOO)	(1998)	Tao and Michalewicz
Maximal Preservative Crossover (MPX)	(1988)	Mühlenbein, Schleuter and Krämer
Position Based Crossover (POS)	(1991)	Syswerda
Order Crossover (OX1)	(1985)	Davis
Order Based Crossover (OX2)	(1991)	Syswerda
Partially mapped Crossover (PMX)	(1985)	Goldberg and Lingle
Voting Recombination Crossover (VR)	(1989)	Mühlenbein
Natural Crossover (NX)	(2000)	Jung and Moon
Voronoi Quantized Crossover (VQX)	(2002)	Seo and Moon

⋮

More Genetic Operators – Mutation for TSP



<u>Operator Name</u>	<u>Date</u>	<u>Authors</u>
Displacement Mutation (DM)	(1992)	Michalewicz
Exchange Mutation (EM)	(1990)	Banzhaf
Insertion Mutation (ISM)	(1988)	Fogel
Inversion Mutation (IVM)	(1990)	Fogel
Scramble Mutation (SM)	(1991)	Syswerda
Simple Inversion Mutation (SIM)	(1975)	Holland

⋮

GA vs MA for Solving TSP Experiments

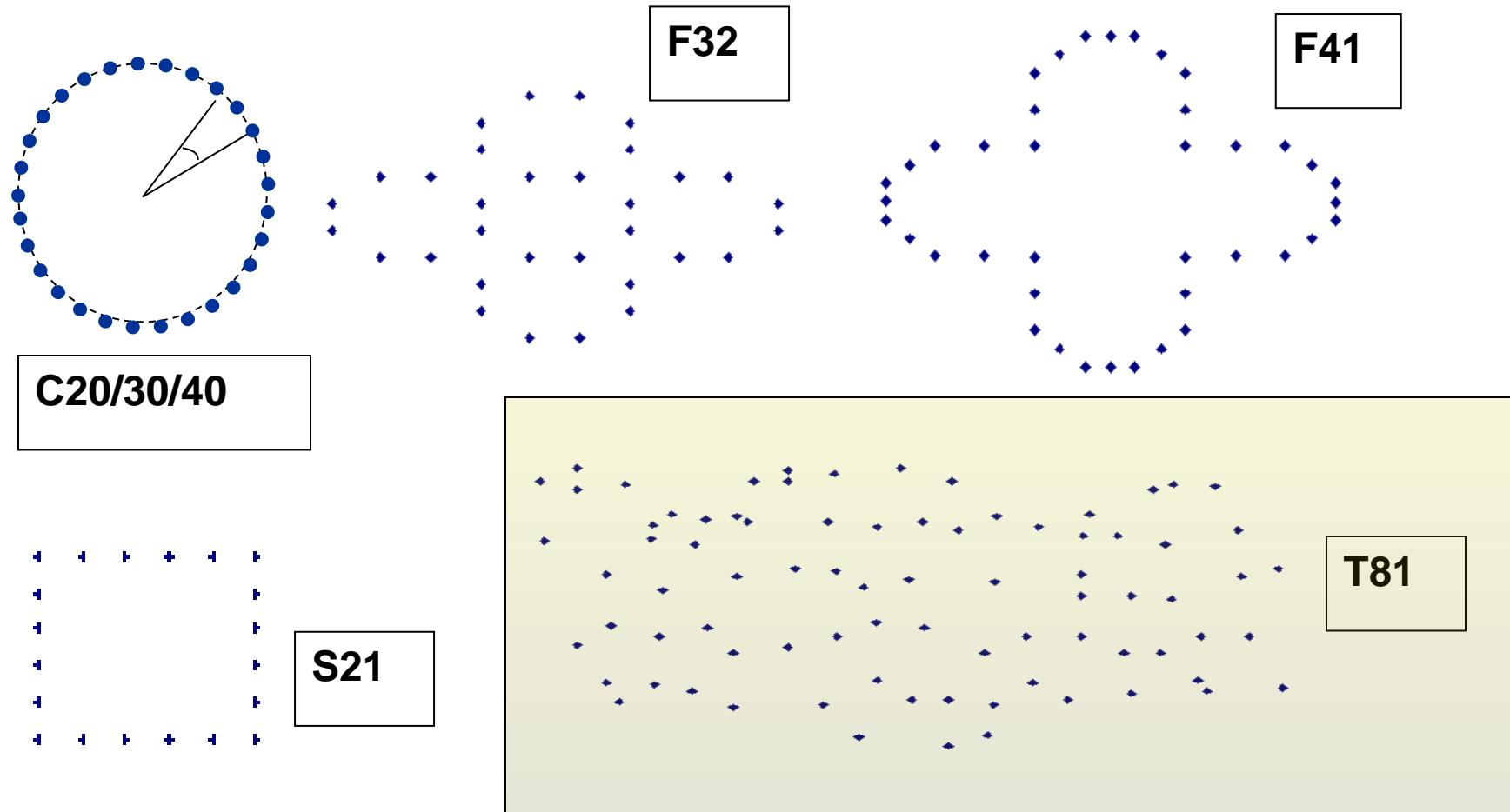
– Components



- Representation: *path representation* (permutation)
 - ▶ For example, (1 3 5 2 6 4) represents a tour starting from city 1, visiting 3, 5, 2, 6, 4 in that order and returning back to 1.
- Fitness function: path length
- Crossover: PMX, 2PTX (with repair/patch-up), OX1
- Mutation Operators: ISM, EM,
- Mate Selection: RANK, TOUR
- Replacement: TG, SS
- Hill Climbing: RMHC using ISM, RMHC using EM
 - A hill climbing step is applied, as long as, max. number of iterations is not exceeded and the indiv. is improved.



Experimental Data



Results



GA TYPE	D.L.	ISM		EM	
		α	β	α	β
SSGA	C20	145.655	62.575	110.786	71.431
	C30	147.590	62.716	142.798	62.716
	C40	207.804	62.768	157.104	121.454
	S21	116.598	60.000	98.714	60.000
	F32	159.191	91.094	118.792	108.694
	F41	207.856	77.609	151.658	120.573
SSGA	C20	149.799	62.575	110.654	62.575
	C30	186.314	62.716	131.118	88.975
	C40	243.077	62.768	144.962	114.914
	HC	S21	60.000	99.411	60.000
	F32	157.291	89.288	114.841	106.003
	F41	205.571	68.168	134.152	92.426
TGGA	C20	173.683	62.575	167.233	62.575
	C30	236.148	62.716	213.842	60.000
	C40	344.740	62.768	278.614	62.768
	S21	169.553	60.000	151.198	60.000
	F32	201.821	92.945	182.890	84.180
	F41	287.101	68.168	241.825	68.168
TGGA	C20	153.128	62.575	147.378	62.575
	C30	198.448	62.716	187.580	62.716
	C40	309.523	62.768	282.683	62.768
	HC	S21	60.000	133.237	60.000
	F32	172.591	84.180	159.893	86.734
	F41	239.901	68.168	241.260	68.168

α : avr. fitness per generation at each run

β : best fitness across 100 runs

From the best to worst, considering the average fitness per generation at each run (100 runs)

- Crossover: **OX1**, 2PTX, PMX
- Mutation: **EM**, ISM (ISM is better in finding the best), 1/chromosome-length
- Mate Selection: **TOUR** (tour-size:2), **RANK**
- Replacement: **TG** (keep top 2 and replace the rest with offspring, $g=popSize-2$), **SS** (replace the worst pair of individuals with the best pair among the offspring and parents)
- Approach: **MA** (Hill Climbing: **EM**, **ISM**), **GA**

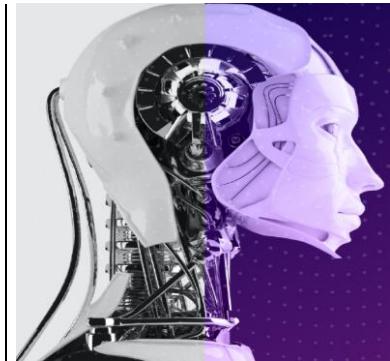
A Classification of Memetic Algorithms



Adaptive Type	Adaptive Level		
	External	Local	Global
Static	Basic meta-Lamarckian learning / Simplerandom		
Adaptive	Qualitative Adaptation		Randomdescent / Randompermdescent
	Quantitative Adaptation		Sub-Problem Decomposition/ Greedy
Self-Adaptive		Multi-memes/ Co-evolution MA	Straightchoice/ Rankedchoice/ Roulettechoice/ Decompchoice/ Biased Roulette Wheel

Yew-Soon Ong; Meng-Hiot Lim; Ning Zhu; Kok-Wai Wong, "Classification of adaptive memetic algorithms: a comparative study," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol.36, no.1, pp.141,152, Feb. 2006 [[PDF](#)]

Multimeme Memetic Algorithms





Self Adaptation for Genetic Operators

- Self Adaptation: Deciding which operators and settings to use on the fly whenever needed receiving feedback during the evolutionary search process
- Davis used varying probabilities of applying different operators
 - ▶ Uses the performance of the operator within the last few generations to update probabilities
- Grefenstette, Kakuza, Friesleben, Hartfelder used subpopulations, each having different set of parameters and operators
- Spears used an additional bit to decide whether apply 2-PTX or UX to an individual (co-evolution)



Multimeme Memetic Algorithms

- Introduce memetic material for each individual
 - ▶ **Co-evolve** genetic and memetic material
- Krasnogor formalised in his PhD thesis*.
- A meme encodes how to apply an operator (which one to apply, max. no. of iterations, acceptance criteria), when to apply, where to apply, and how frequent to apply
- Meme of each operator can be combined under a *memeplex*

*Krasnogor N (2002) Studies on the theory and design space of memetic algorithms, PhD Thesis, University of the West of England, Bristol, UK

Grammar for a Memeplex— Compound of Memes



- Memeplex = <Meme>;<Meme>:<Memeplex>
- Meme = (< Where >, < When >, < How >, < Frequency >)
- Where = < Location > Crossover ; < Location > Mutation;
- Location = After ; Before
- When = < BooleanCondition >
- How = GeneralHillClimber < Move >< Strategy >< Iterations >;
BoltzmanAdaptiveHillClimber < Move >< Strategy >< Iterations > ...
- Frequency = Always; For < Iterations >; Never ...
- Move = 2-exchange, 3-exchange
- Strategy = NextAscent; SteepestAscent; MiniMax < Size >; MaxiMin < Size >
- Size = < Num >
- Iterations = < Num >

⋮



Multimeme Memetic Algorithms – Features

- Memes represent instructions for self-improvement
 - ▶ Specify set of rules, programs, heuristics, strategies, behaviors, etc.
- Interaction between memes and genes are not direct
 - ▶ Mediated by individual (genetic and memetic material)
- Memes can evolve, change by means of nontraditional genetic transformations and metrics



Examples – Implementation

genetic material memetic material

- MAX-SAT: (1 0 0 0 1 + 2)
memeplex: Hc2 (Hc: meme and 2:meme option)
 - Hc (hill climbing operator) → 0: RMHC, 1: SDHC, 2: NDHC
 - Mp2Mo1Cp1Co2W0Hc1Hs1I2
 - TSP:(4 5 2 1 8 7 6 9 3 + 2 1 1 2 0 1 1 2)
 - Mp (mutation probability) → 0: $2/n$, 1: $1/n$, 2: $1/(2xn)$, 3: 0.1, ...
 #1 {
 - Mo (mutation operator) → 0: Exchange, 1: Insertion, ...
 #2 {
 - Cp (crossover probability) → 0: 1.0, 1: 0.95, 2: 0.90, ...
 #2 {
 - Co (crossover operator) → 0: 1PTX+repair, 1: OX, 2: CX, ...
 #2 {
 - W (where to apply HC) → 0: After Mutation, 1: After Crossover, ...
 #3 {
 - Hc (strategy/how to apply HC) → 0: first impr., 1: best improvement, ...
 #3 {
 - Hs (move/HC step) → 0: Exchange, 1: Adjacent interchange, ...
 #3 {
 - I (frequency/number of HC iterations) → 0: n , 1: $2n$, 2: $4n$, 3: $8n$, ...
 #3 {



Inheriting Memetic Material – SIM

(1 0 | 0 0 1 + Hc1Hs1I2) $f=2$
(1 1 | 0 1 1 + Hc0Hs4I1) $f=1$

(1 1 0 0 1 + Hc0Hs4I1)
(1 0 0 1 1 + Hc0Hs4I1)

```
Individual_Level_Crossover(parent1, parent2)
BEGIN
  IF(both parents carry the same meme)
    Cross parents genetic material.
    Inherit common meme to offspring.
  ELSE-IF (parent1.fitness()==parent2.fitness())
    /* the two parents have different memes           */
    /* but their fitness are comparable hence       */
    /* a random choice is made                      */
    Cross parents genetic material.
    Choose a meme randomly from any of the two parents.
    Inherit selected meme to offspring.
  ELSE
    /* parents don't share memes nor fitness values */
    /* hence the fittest individual                  */
    /* imposes its meme preference                 */
    Cross parents genetic material.
    Choose meme from fittest parent.
    Inherit the chosen meme to offspring.
END
```

Krasnogor N, Smith JE (2001) Emergence of profitable search strategies based on a simple inheritance mechanism. In: Proc of the Genetic and Evolutionary Comp. Conf., pp 432–439

<http://eprints.uwe.ac.uk/11088/1/11088.pdf>



Mutating Memes during Evolution

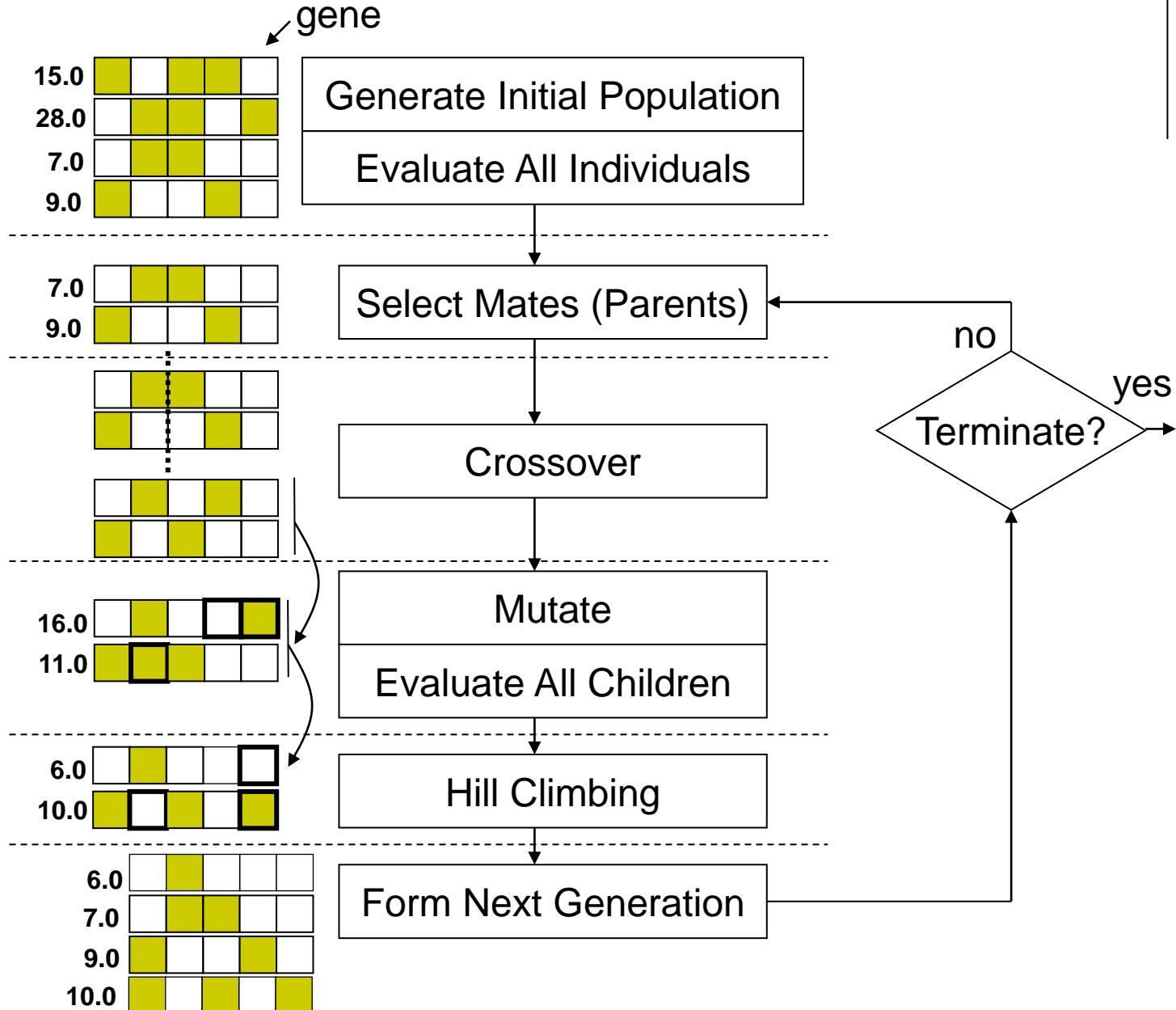
- Innovation rate (IR) $\in [0,1]$, is the probability of mutating the memes
- Mutation randomly sets the meme option to one of the other options
 - ▶ MAX-SAT: $(1\ 0\ 0\ 0\ 1 + Hc2) \rightarrow 0: (1\ 0\ 0\ 0\ 1 + Hc0)$
 - ▶ TSP: $(4\ 5\ 2\ 1\ 8\ 7\ 6\ 9\ 3 + Mp2Mo1Cp1Co2W0Hc1Hs1I2) \rightarrow (4\ 5\ 2\ 1\ 8\ 7\ 6\ 9\ 3 + Mp0Mo2Cp0Co1W1Hc3Hs0I4)$
- IR=0; no innovation, if a meme option is not introduced in the initial generation, it will not be reintroduced again
- IR=1; All different strategies implied by the available M memes might be equally used.



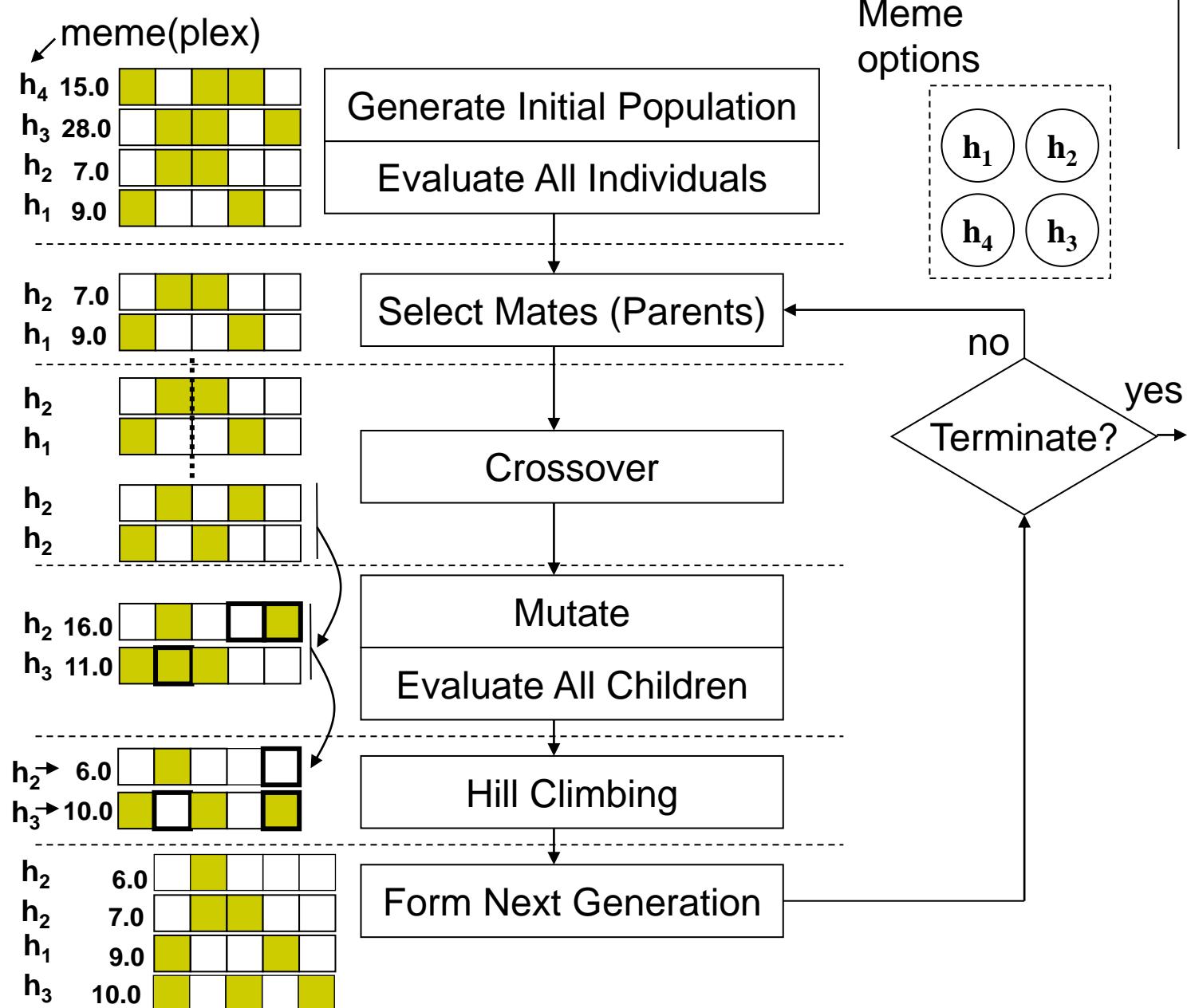
Measure for Evaluating Meme Performance

- *Concentration of a meme* ($c_i(t)$) is the total number of individuals that carry the meme i at a given generation t ,
 - ▶ Crude measure of a meme success; gives no information about continual usage of a meme
- *Evolutionary activity of a meme* ($a_i(t)$) is the accumulation of meme concentration until a given generation,
$$a_i(t) = \begin{cases} \int_0^t c_i(t)dt, & \text{if } c_i(t) > 0 \\ 0, & \text{otherwise} \end{cases}$$
 - ▶ Slope in a plot represents the rate of increase of a meme concentration

Memetic Algorithms (MAs)



Multimeme Memetic Algorithms (MMAs)





Ender Özcan, Burak Bilgin, Emin Erkan Korkmaz, A Comprehensive Analysis of Hyper-heuristics, Intelligent Data Analysis, 12:1, pp. 3-23, 2008. [\[PDF\]](#)

BENCHMARK FUNCTION OPTIMISATION - REVISITED

MMA Experiments for Benchmark Function Optimisation – Additional Settings



- IR rate is fixed as 0.20
- Two sets of experiments are performed

MMA1. A single *good* meme and two *poor performing* memes are used to test the power of MMAs in identifying the *good* ones.

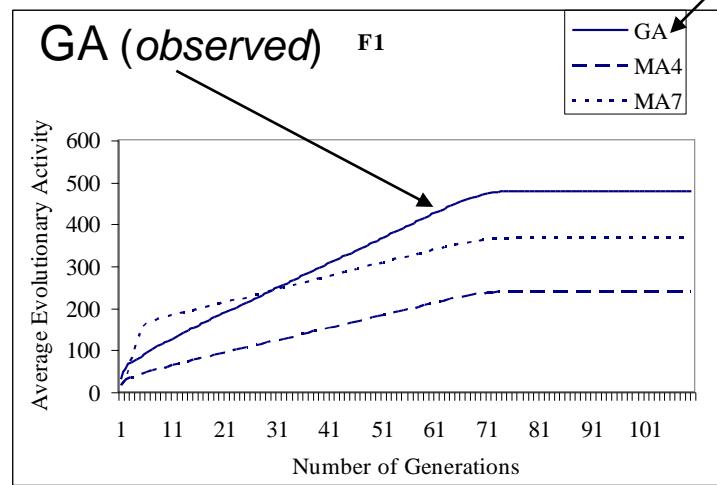
MMA2. Memes GA, MA0, MA1, MA2 and MA3 are used to observe whether MMA will provide some type of synergy between memes.

MMA 1.a

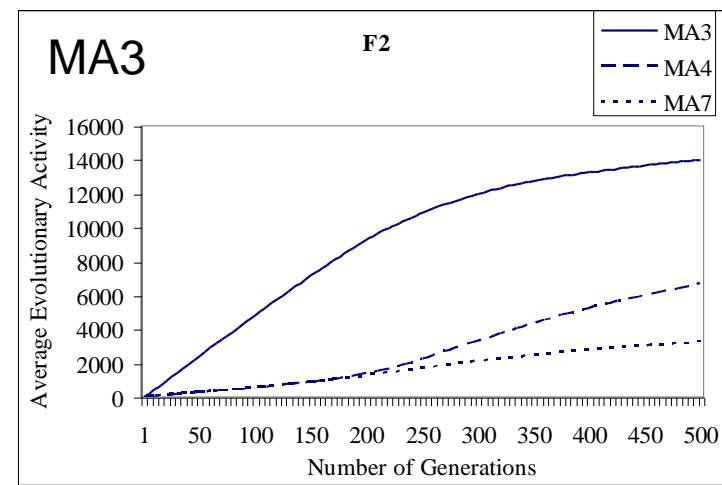
*Good meme (GA)
followed by two poor
performing memes*



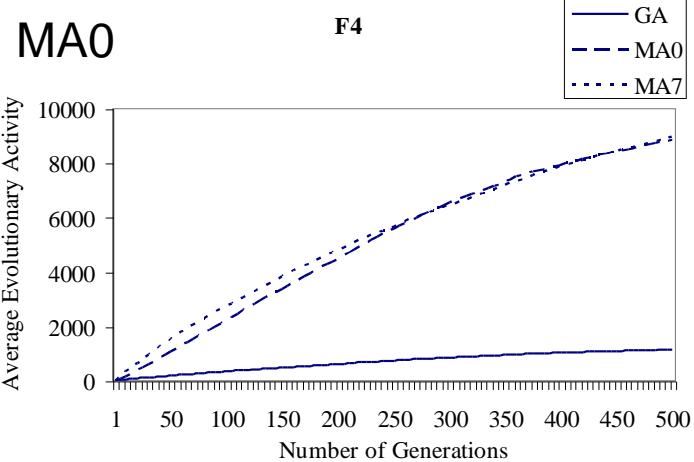
GA (expected – good meme)



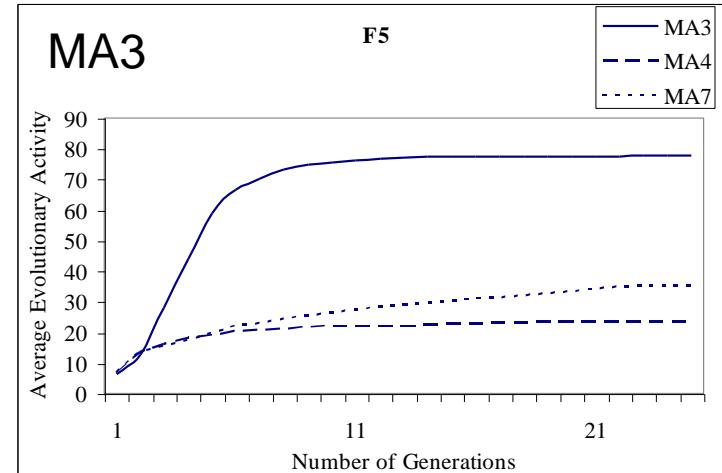
MA3



MA0



MA3



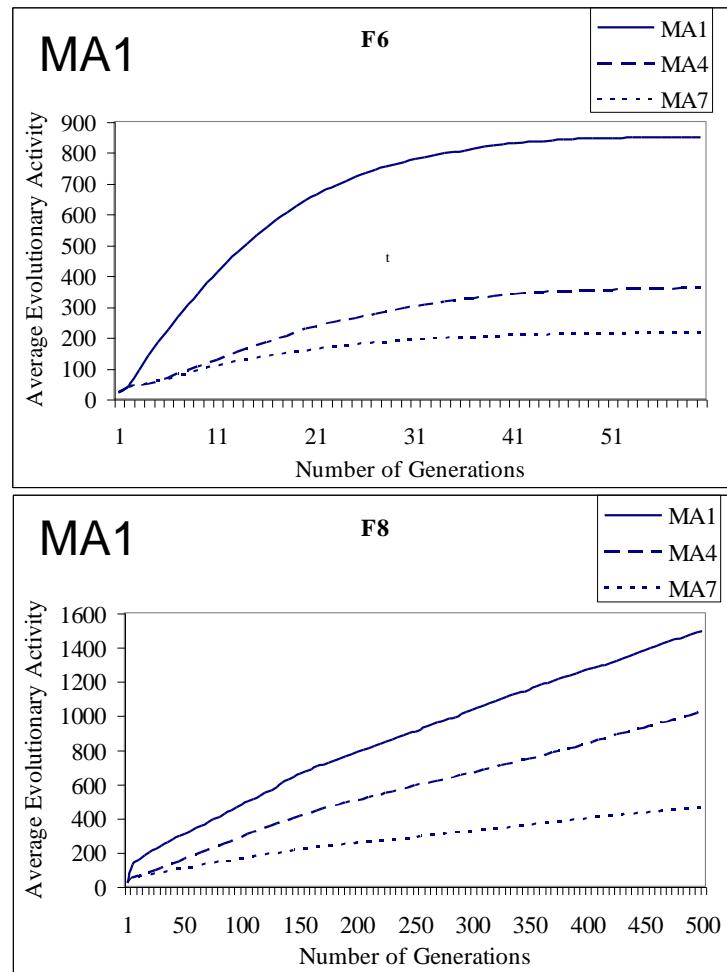
MA0

MA3

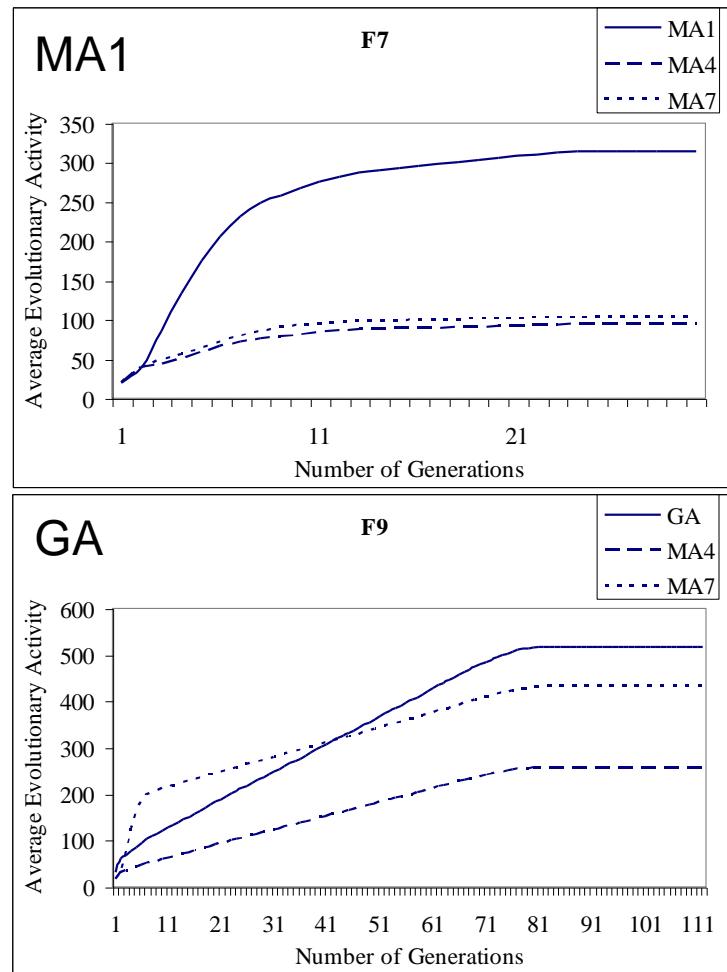
MMA 1.b



MA1



MA1



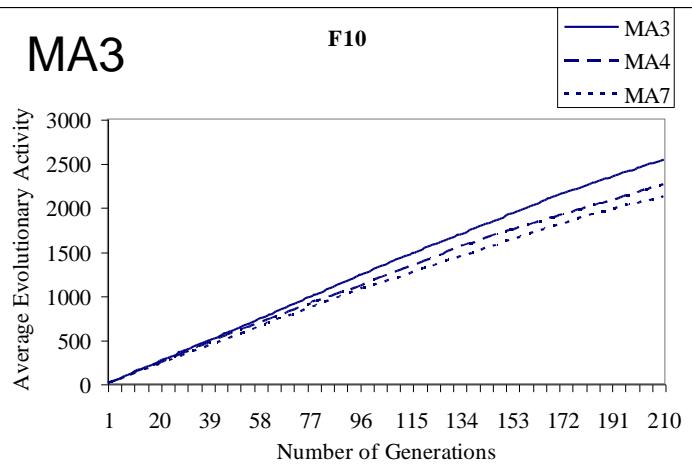
MA1

GA

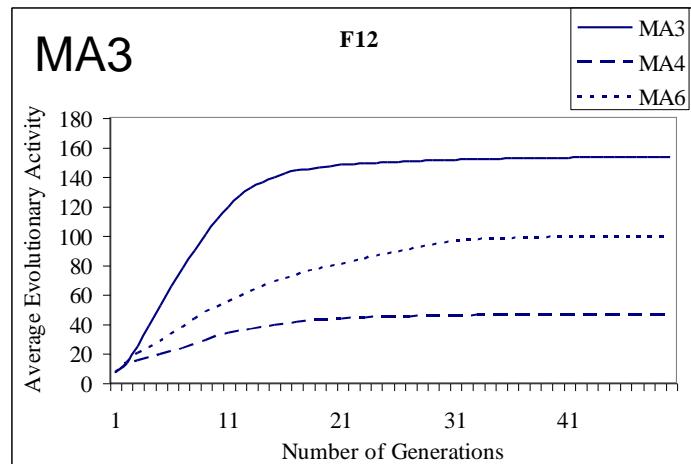
MMA 1.c



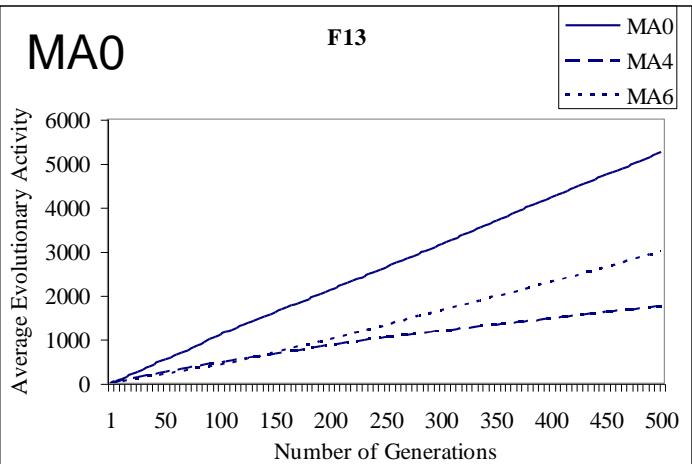
MA3



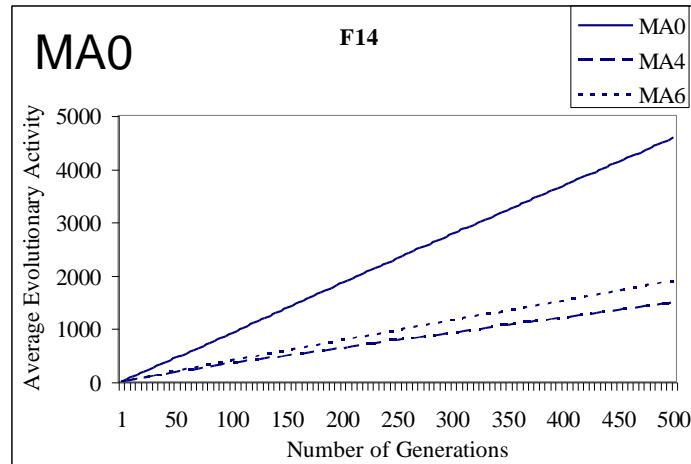
MA3



MA0



MA0



MA0

MA0

MMA 2 Experiments – Synergy Between Memes

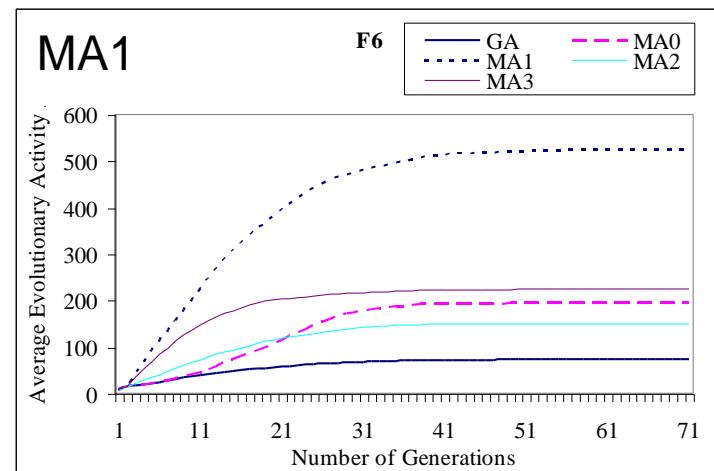
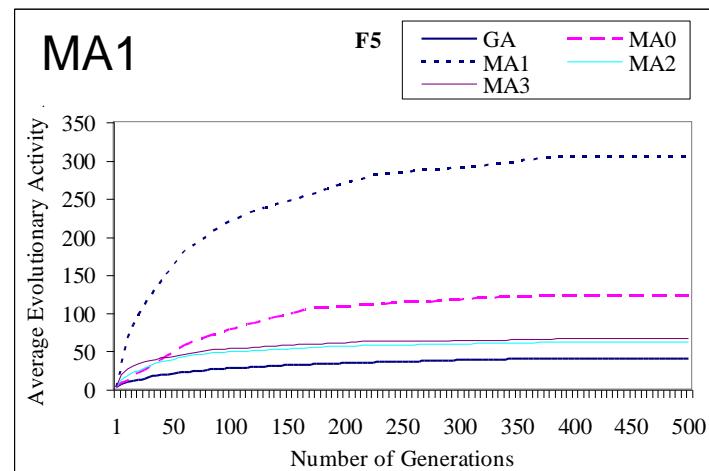
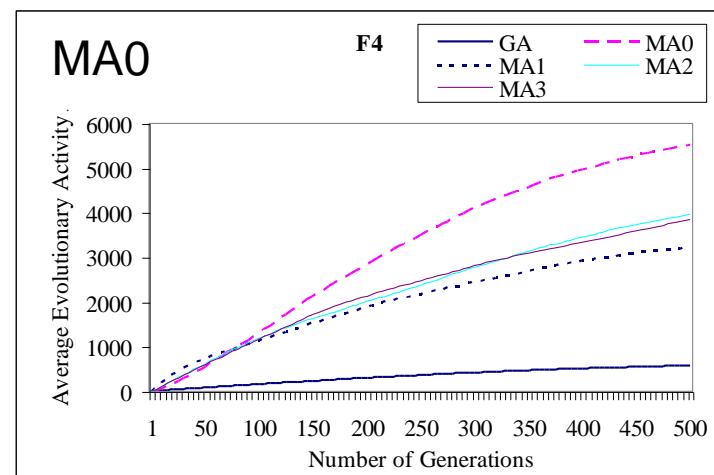
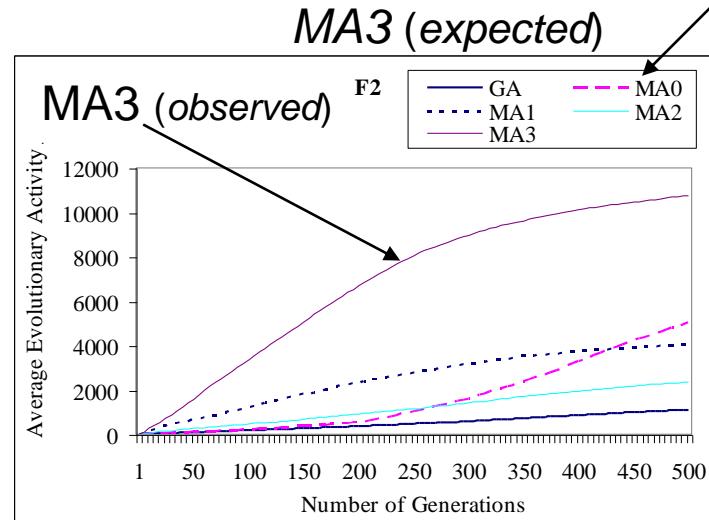


<i>label</i>	<i>type</i>	<i>avr. no. of evals.</i>		<i>st.dev.</i>
F1	MMA	17,580		2,226
	MA1	92,256		0
F2	MMA	23,605,004	24,364,979	
	MA3	8,455,507	3,803,504	
F3	MMA	72,252		11,772
	MA3	82,769		16,512
F4	MMA	12,926,879	11,435,876	
	MA0	9,494,844	10,332,574	
F5	MMA	46,975		79,394
	MA3	11,619		2,293
F6	MMA	553,306	231,124	
	MA1	525,398	262,055	
F7	MMA	349,250	324,544	
	MA1	167,799	60,577	
F8	MMA	5,215,787	9,658,230	
	MA1	1,906,134	6,646,991	
F9	MMA	43,871		12,193
	MA1	180,783		12,647
F10	MMA	3,100,515	4,565,736	
	MA3	1,340,811	988,971	
F11	MMA	17,580	2,226	
	MA1	36,060	0	
F12	MMA	31,297	14,961	
	MA3	29,246	4,936	
F13	MMA	7,667,352	2,832,376	
	MA0	4,348,896	1,617,951	
F14	MMA	3,674,932	2,623,300	
	MA0	1,072,117	1,111,825	

Second entry for each function indicates the best performing hill climbing under the generic MA framework where HC is applied after mutation

MMA 2.a

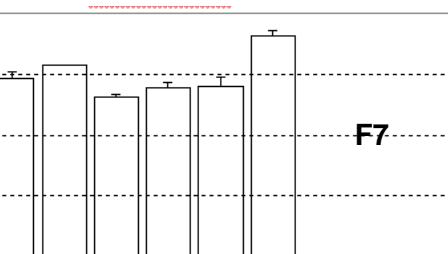
Memes are fixed



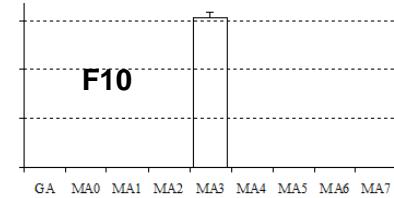
MA3

MA1

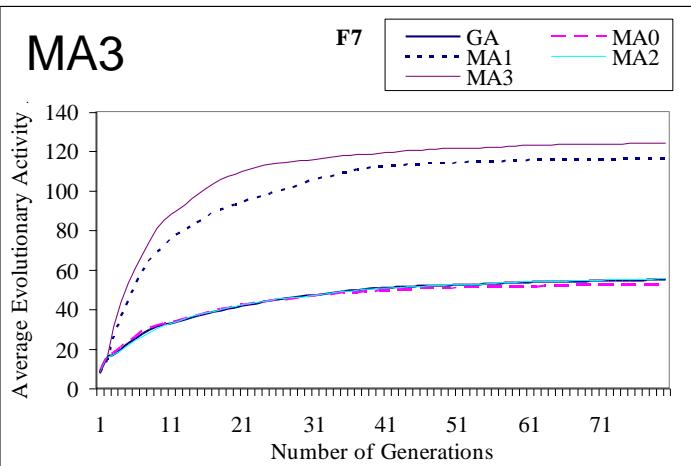
MMA 2.b



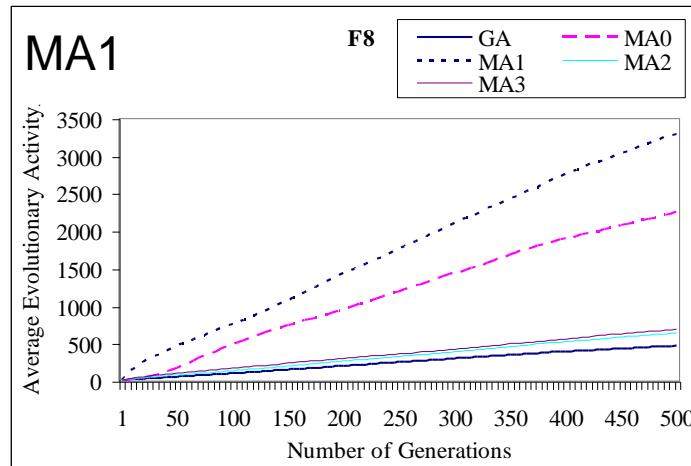
MA1



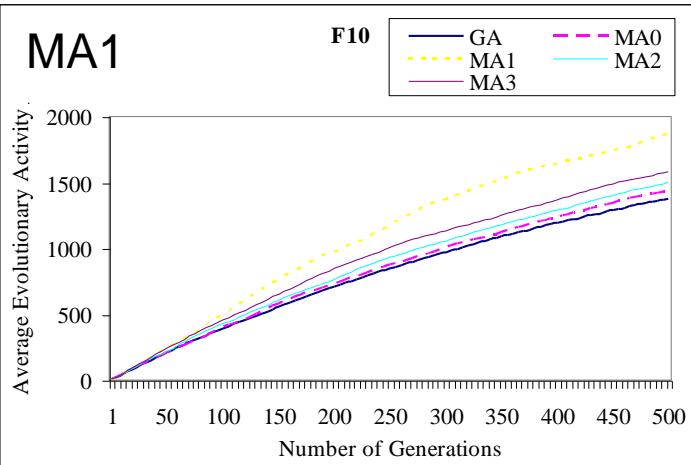
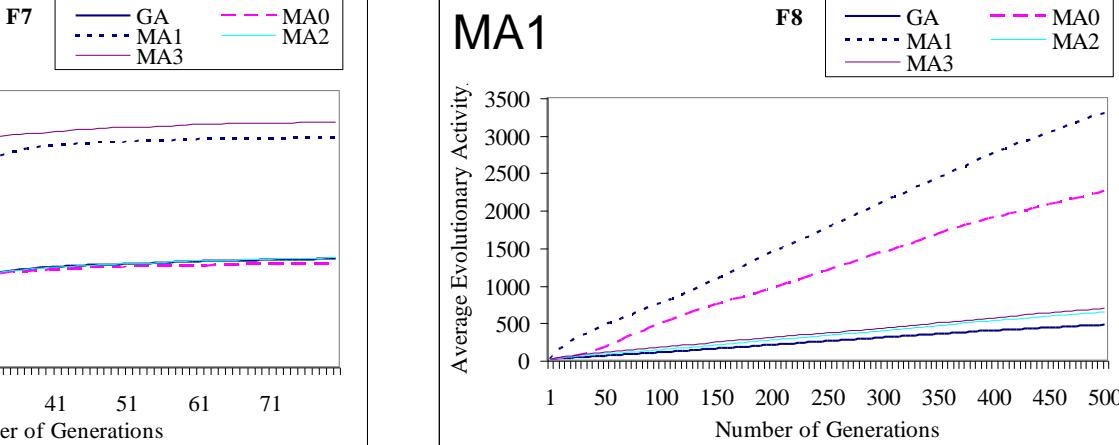
MA1



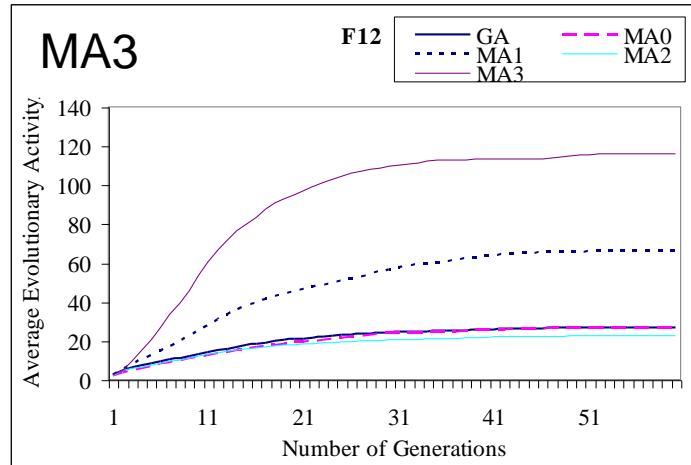
MA3



MA1



MA3



MA3

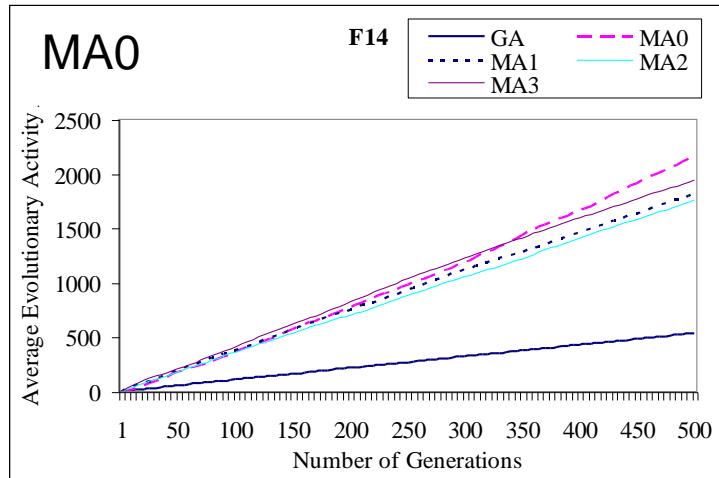
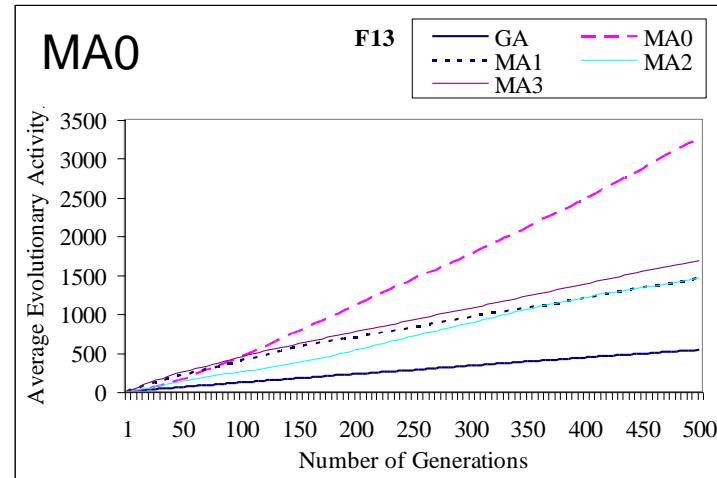


MMA 2.c



MA0

MA0



Results



- Average evolutionary activity plots show that the multimeme approach successfully identifies useful memes.
 - ▶ Even if there are useless operators in the system
- Any hill climber seems to attain the optimum fast for F1, F3 and F11 under any setting
- In all runs full success is achieved for all cases.



Results II

- MMA can identify the best meme or a meme that does not perform significantly better than the best meme for almost each benchmark function
- Comparing the experimental results obtained using MMA and MA with the best meme for each benchmark function indicate that MA with the best meme is superior based on the average number of evaluations, except for F1, F3, F9 and F11
 - ▶ Learning is time consuming hence there is a trade-off



Weaknesses of Evolutionary Algorithms

- Limited theoretical and mathematical analyses – this is a growing field of study
 - ▶ Schema theorem and building block hypothesis
 - ▶ Markov chain analysis
 - ▶ Chaos theory
 - ▶ Martingale theory (for convergence analysis)
 - ▶ Runtime analysis with respect various parameter settings (e.g., expected runtime analysis based on fitness levels/partitions, drift analysis)

Per Kristian Lehre and Pietro S. Oliveto, GECCO 2021, [Runtime Analysis of Evolutionary Algorithms: Basic Introduction](#)

- Considered slow for online applications and for some large offline problems
 - ▶ Speedy hardware
 - ▶ Parallel/distributed processing



Overall Summary

- Genetic Algorithms represent a subclass of nature inspired Evolutionary Computation Techniques
 - ▶ GAs creates an initial population of individuals and performs the search by applying selection, crossover and mutation on individuals at each evolutionary cycle
 - ▶ GA components including the parameter settings require careful tailoring to the problem in hand
- Memetic Algorithms hybridise GAs with Hill Climbing
- There is a variety of benchmark functions used for performance comparison of search algorithms
- In general, a memetic algorithm performs better than a genetic algorithm
- Choice of meme (operators and their settings) along with the encoding influence the performance of a memetic algorithm



Overall Summary (cont.)

- Multimeme Algorithm can indeed learn how to choose an operator and relevant settings through the evolutionary process (co-evolution)
 - ▶ There is a trade-off as learning requires time and a memetic algorithm with a single setting could perform better
- If there is limited number of operators and settings MMA can be used. If the number of operators and settings are large then additional analyses and methods could be needed to reduce the number of options leading to an MMA with improved performance
- There is no single recipe for even choosing the set of memes while designing a memetic algorithm for solving a given problem.
- Synergy between memes/meme components could be possible

Q&A



Thank you.

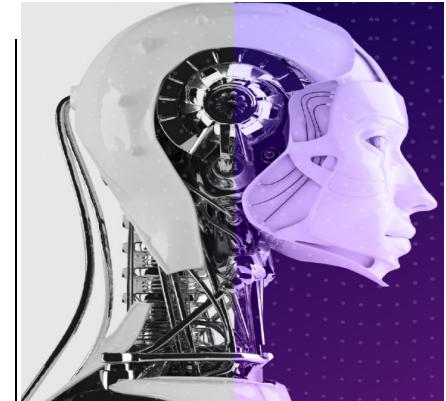
Ender Özcan ender.ozcan@nottingham.ac.uk

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
<http://www.nottingham.ac.uk/~pszeo/>

Hyper-heuristics I

Lecture 7

Ender Özcan



UNITED KINGDOM • CHINA • MALAYSIA

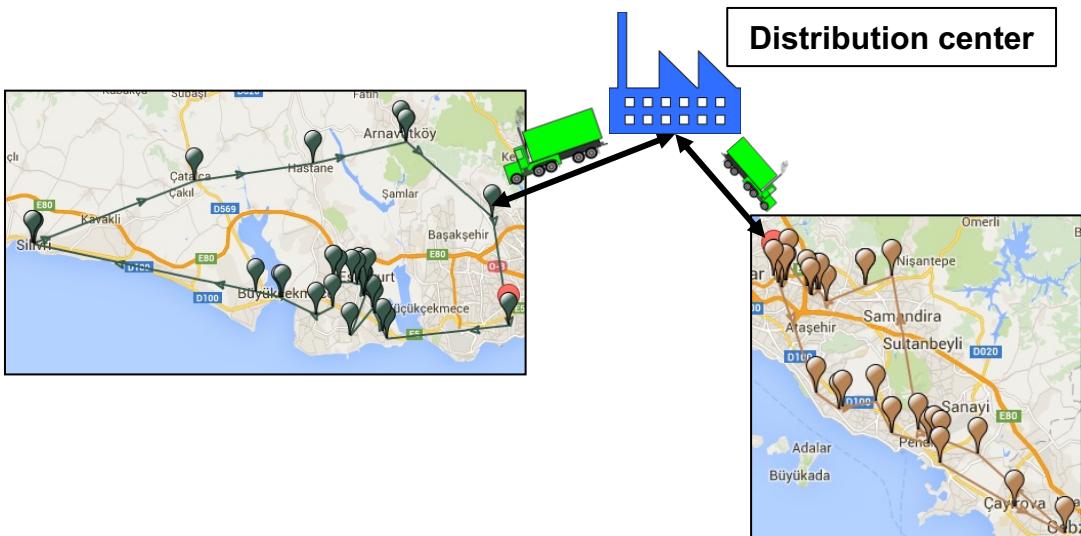
State-of-the-art in (Meta)Heuristic Optimisation



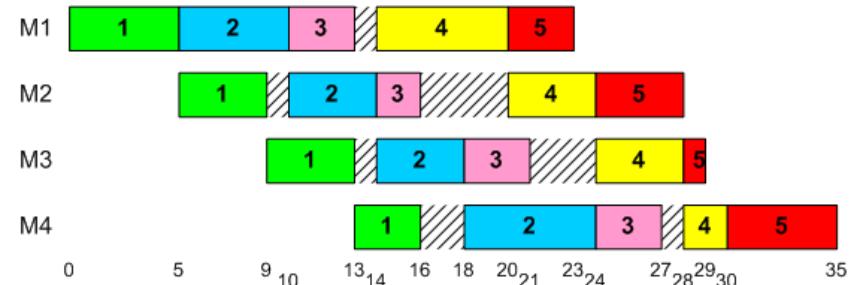
Trial and Error:

- Design and implement algorithmic components
- Configure the algorithm and tune the parameters: Test on selected instances (revisit the design options)
- Performance analyses (revisit the design options)

Vehicle Routing



Flowshop Scheduling



Nurse Rostering

John

03	04	05	06	07	08	09	10	11	12	13	14	15	16
M	T	W	T	F	S	S	M	T	W	T	F	S	S
E	E	N	N	D			D	N	N				

Gem

03	04	05	06	07	08	09	10	11	12	13	14	15	16
M	T	W	T	F	S	S	M	T	W	T	F	S	S
D	D			D	D	D	D	D	D				



Hyper-heuristics

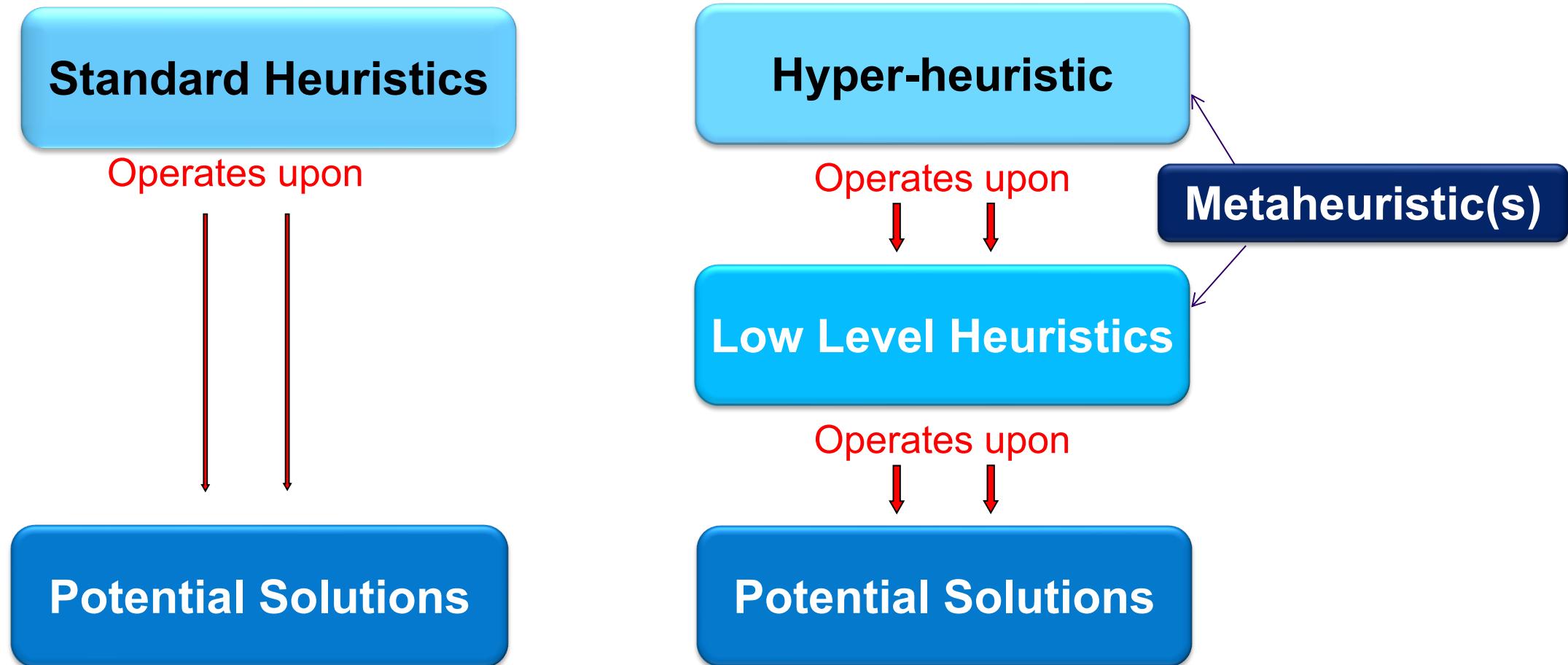
A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computationally difficult problems

- A class of methodologies for cross-domain search

- J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, Recent Advances in Selection Hyper-heuristics, European Journal of Operational Research, DOI:10.1016/j.ejor.2019.07.073 (invited review) (Open Access - available online [[PDF](#)]).
- E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A Survey of the State of the Art, Journal of the Operational Research Society, 64 (12) , pp. 1695-1724, 2013. [[PDF](#)]



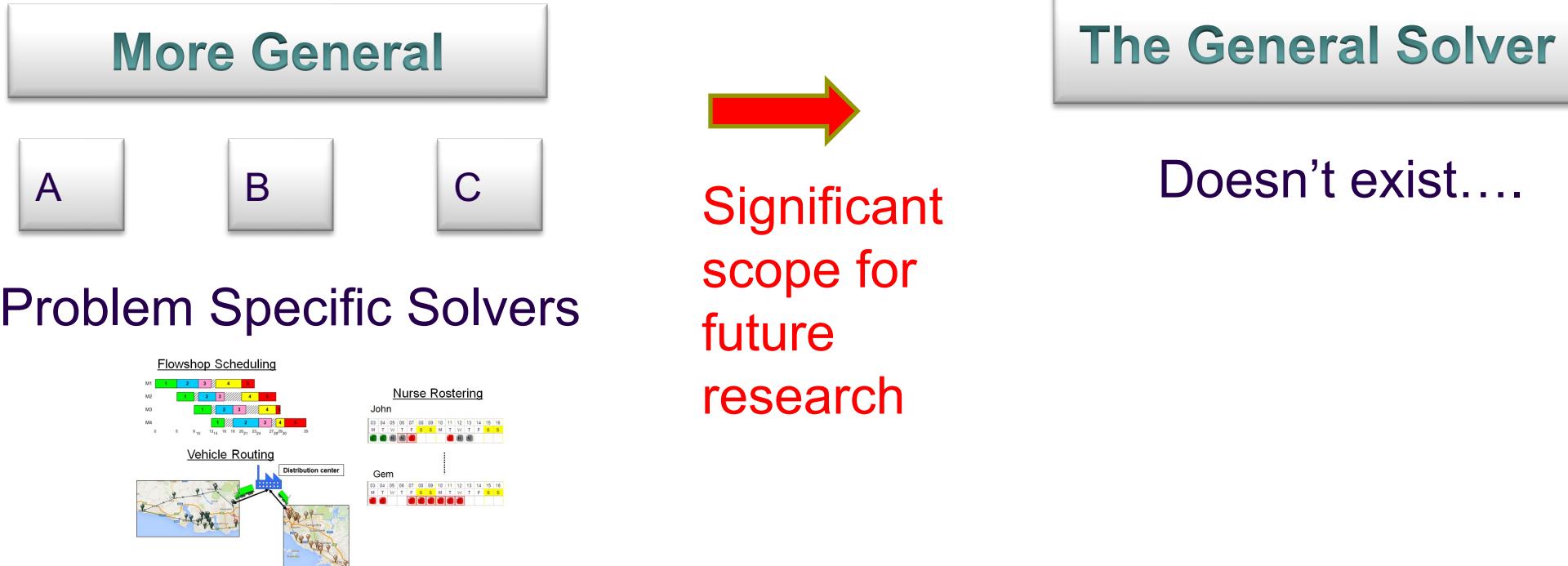
Different Search Spaces



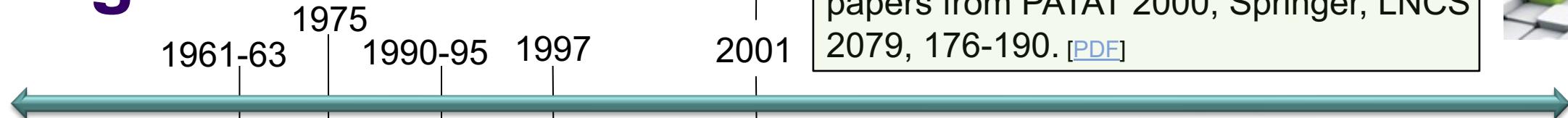
Motivation – Cross Domain Search



- Hyper-heuristic research is motivated by raising the level of generality of search methods . What are the limits?
- **Grand Challenge**



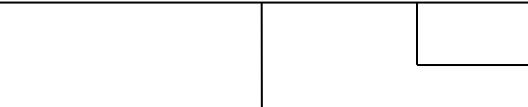
Hyper-heuristics: Origins



Cowling P.I., Kendall G. and Soubeiga E., 2001. A Hyperheuristic Approach to Scheduling a Sales Summit, selected papers from PATAT 2000, Springer, LNCS 2079, 176-190. [[PDF](#)]

Fisher H. and Thompson G.L., 1963. Probabilistic Learning Combinations of Local Job-shop Scheduling Rules. Ch 15,:225-251, Prentice Hall, New Jersey.

Crowston W.B., Glover F., Thompson G.L. and Trawick J.D. Probabilistic and Parameter Learning Combinations of Local Job Shop Scheduling Rules. ONR Research Memorandum, GSIA,CMU, Pittsburgh, (117), 1963

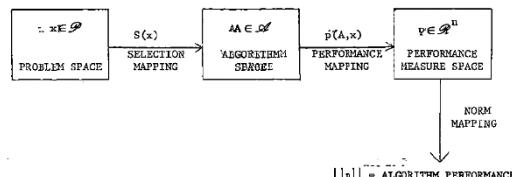


THE ALGORITHM SELECTION PROBLEM

John R. Rice
Computer Science Department
Purdue University
West Lafayette, Indiana 47907

July 1975

CSD-TR 152



Storer R. H., Wu S. D. , Vaccari R., 1992. New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling, INFORMS, 38(10), 1495-1509.

Fang H.-L., Ross P. and Corne D., 1994. A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems., in'ECAI' , 590-594.

Denzinger J., Fuchs M. and Fuchs M., 1997. High performance ATP systems by combining several AI methods. In Proc. of the 15th IJCAI, 102-107.

A Selection Hyper-heuristic Framework for Cross-domain Search

Cowling P.I., Kendall G. and Soubeiga E., 2001. A Hyperheuristic Approach to Scheduling a Sales Summit, selected papers from PATAT 2000, Springer, LNCS 2079, 176-190. [[PDF](#)]



- No domain knowledge, other than that embedded in a range of simple knowledge-poor heuristics.
- Robust enough to effectively handle a wide range of problems and problem instances from a variety of domains.

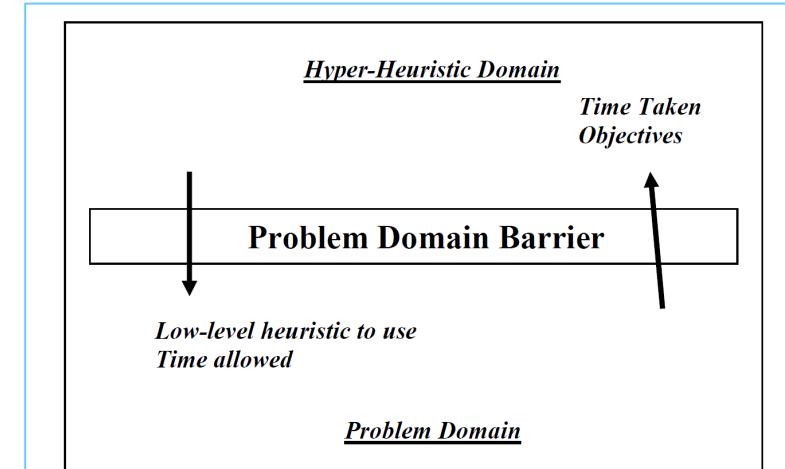


Fig. 1. The hyperheuristic approach and the problem domain barrier

2 The Sales Summit Scheduling Problem

The problem we are studying is encountered by a commercial company that organises regular sales summits which bring together two groups of company representatives. The first group, *suppliers*, represent companies who wish to sell some product or

Characteristics of Hyper-heuristics (initial framework)



- Operate on a **search space of heuristics (neighbourhood operators)** rather than directly on a search space of solutions
- Aim is to take advantage of strengths and avoid weaknesses of each heuristic (operator)
- No problem specific knowledge is required during the search over the heuristics (operator) space
- Easy to implement, practical to deploy (easy, cheap, fast)
- Existing (or computer generated) heuristics (operators) can be used within hyper-heuristics



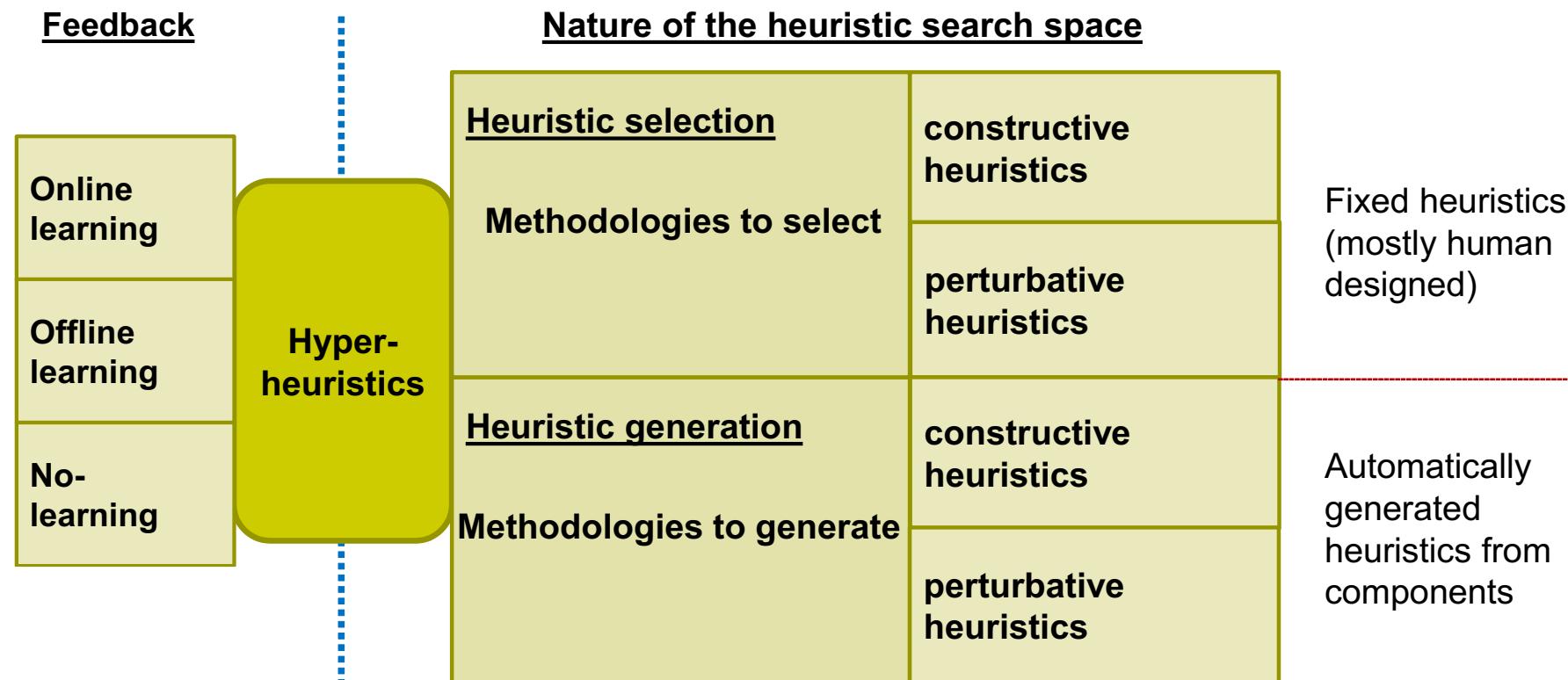
Related Areas

- Reactive search
- Algorithm selection/portfolios
- Adaptive operator selection
- Meta-learning
- Co-evolution/multimeme memetic algorithms/Memetic computing
- Variable Neighbourhood Search
- Cooperative (Distributed) Search
- Parameter control (e.g., in EAs)
- Algorithm configuration,...



Classification of Hyper-heuristics

E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan and J. Woodward, A Classification of Hyper-heuristic Approaches: Revisited, In Gendreau, M, and Potvin, JY. (eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 272, pp. 453-477. Springer Cham, 2019. [[PDF](#)]

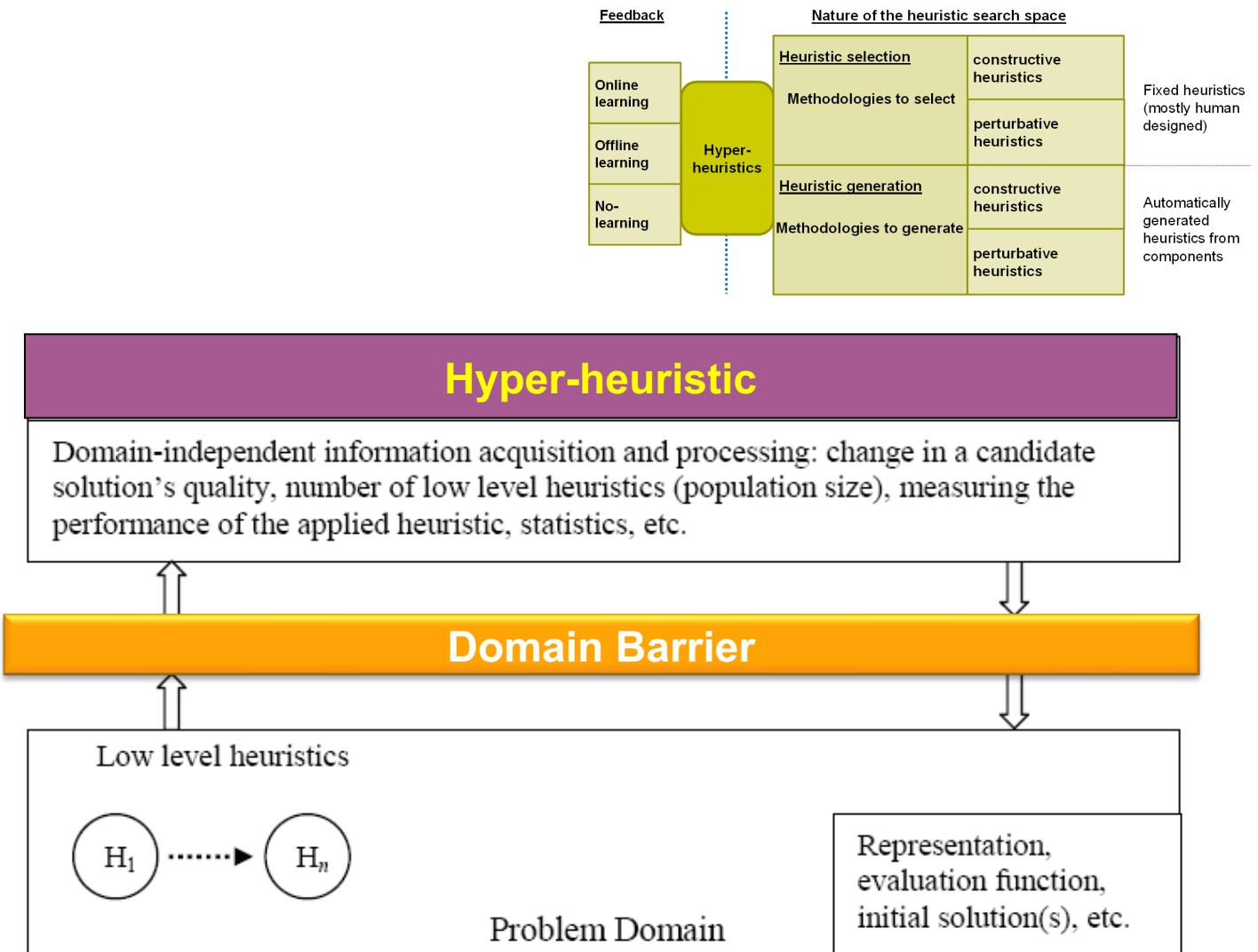


Software Tools for Heuristic Selection/Generation

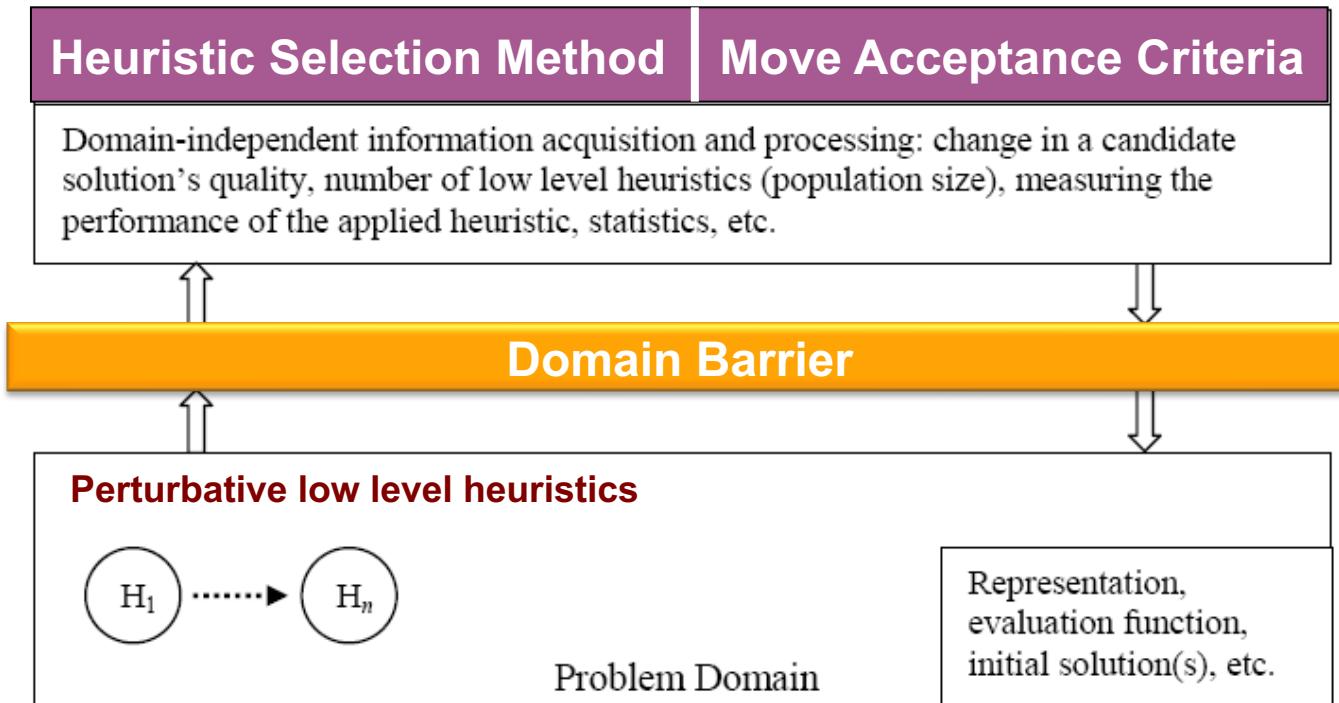
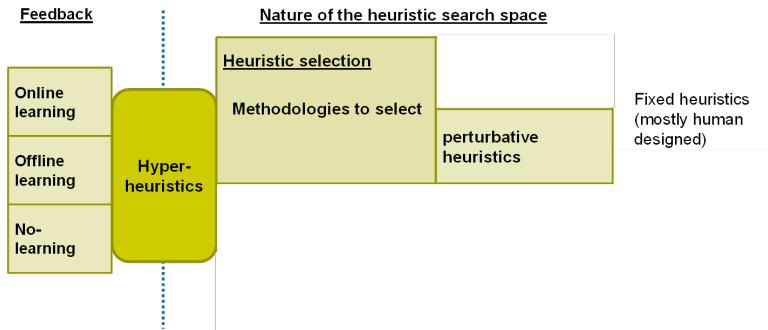


- SATzilla: algorithm portfolio oriented data-driven framework
- Simple Neighborhood-based Algorithm Portfolio in PYthon (snappy) (*old location: <http://researcher.watson.ibm.com/researcher/files/us-samulowitz/snappy.zip>*)
- Hyper-heuristics Flexible Interface (HyFlex):
<http://www.asap.cs.nott.ac.uk/external/chesc2011/>
- ParHyFlex extends MPI: **<https://tinyurl.com/zrfz2vw>**
- Hyperion provides a white-box framework giving a metaheuristic/hyper-heuristic full access to the problem domain:
<https://github.com/MitLware/mitl-hyperion>
- ECJ: **<http://cs.gmu.edu/~eclab/projects/ecj/>**

A Hyper-heuristic Framework



A Selection Hyper-heuristic Framework – Single Point Search



A Selection Hyper-heuristic Framework – Single Point Search



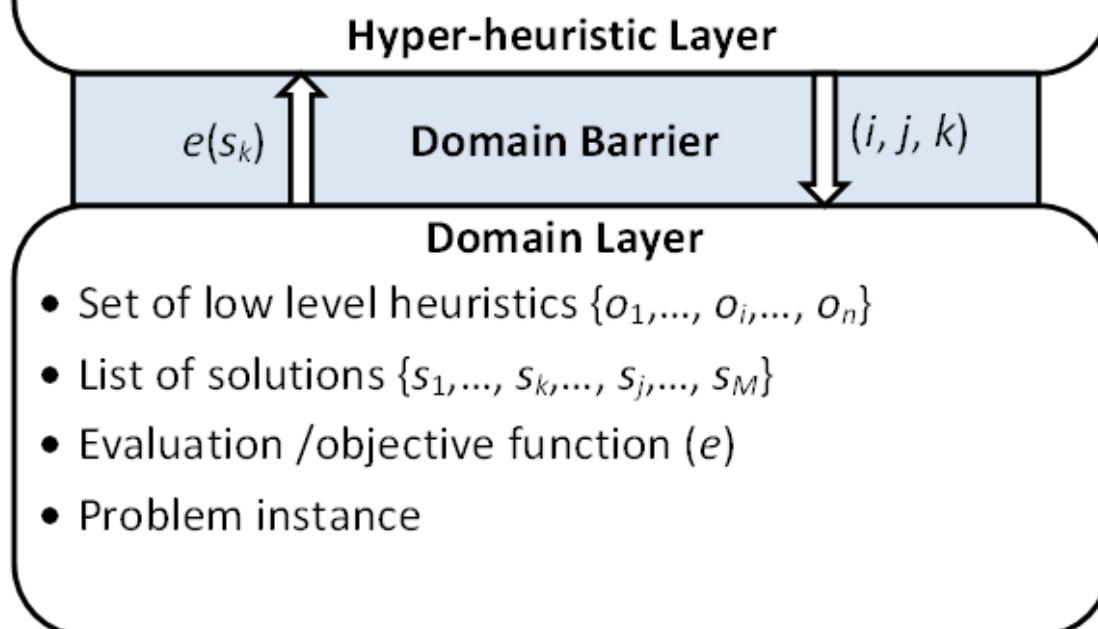
1. generate initial candidate solution p
2. while (termination criteria not satisfied){
 3. select a heuristic (or subset of heuristics) h from $\{H_1, \dots, H_n\}$
 4. generate a new solution (or solutions) s by applying h to p
 5. decide whether to accept s or not
 6. if (s is accepted) then
 7. $p=s$
 8. return p ;

Hyper-heuristics Flexible Interface (HyFlex)



<http://www.asap.cs.nott.ac.uk/external/chesc2011/> (web archive)

Methodologies to decide which low level heuristic (o_i) to apply to which solution (s_j) and at which location to store the new solution (s_k) in the list of solutions based on the history of visited solutions and their objective values.



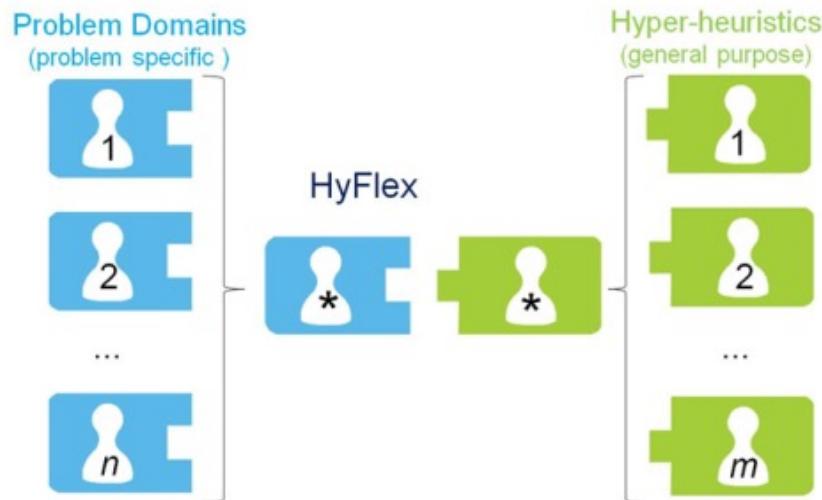
Ochoa G, Hyde M, Curtois T, Vazquez-Rodriguez JA, Walker J, Gendreau M, Kendall G, McCollum B, Parkes AJ, Petrovic S, Burke EK (2012) HyFlex: a benchmark framework for cross-domain heuristic search. In: Evolutionary Computation in Combinatorial Optimization, LNCS 7245, pp 136–147

Hyper-heuristics Flexible Interface (HyFlex)



<http://www.asap.cs.nott.ac.uk/external/chesc2011/> (web archive)

- Defines behaviours of components and arranges the interaction between them



Separation between the problem-specific and the general-purpose parts, both of which are reusable and interchangeable through the HyFlex interface



HyFlex v1.0 Java Implementation

- Currently there are 6 problem domain implementations
- heuristic types: mutational, ruin-recreate, local search, crossover
- parameters: intensity, depth of search

MAX-SAT

**Bin
Packing**

Flow Shop

**Personnel
Scheduling**

TSP

VRP

Heuristic IDs	LLH0	LLH1	LLH2	LLH3	LLH4	LLH5	LLH6	LLH7
MAX-SAT	MU ₀	MU ₁	MU ₂	MU ₃	MU ₄	MU ₅	RC₀	HC ₀
Bin Packing	MU ₀	RC ₀	RC₁	MU ₁	HC ₀	MU ₂	HC ₁	XO ₀
PS	HC ₀	HC ₁	HC ₂	HC ₃	HC₄	RC ₀	RC ₁	RC ₂
PFS	MU ₀	MU ₁	MU ₂	MU ₃	MU₄	RC ₀	RC₁	HC ₀
TSP	MU ₀	MU ₁	MU ₂	MU ₃	MU₄	RC₀	HC ₀	HC ₁
VRP	MU ₀	MU ₁	RC ₀	RC₁	HC ₀	XO ₀	XO₁	MU ₂
Heuristic IDs	LLH8	LLH9	LLH10	LLH11	LLH12	LLH13	LLH14	
MAX-SAT	HC₁	XO ₀	XO₁					
PS	XO ₀	XO ₁	XO₂	MU₀				
PFS	HC ₁	HC ₂	HC₃	XO ₀	XO ₁	XO ₂	XO ₃	
TSP	HC₂	XO ₀	XO ₁	XO ₂	XO₃			
VRP	HC ₁	HC₂						

Example Domain: (1D Offline) Bin Packing



Pack a **set** of items of sizes s_i for $i = 1, \dots, n$

- ▶ Sizes are integer values and $s_i \in [1, C]$
- ▶ C is the fixed capacity of each bin

in such a way that

- ▶ Never exceed bin capacity
- ▶ Minimise number of bins used

Standard NP-hard problem

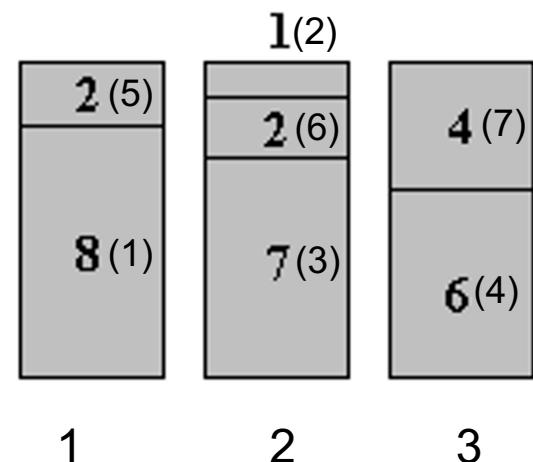


Example Problem Instance

- Given the set of ($n=$) 7 items

$$S = \{ \overset{i:}{\underset{1}{8}}, \overset{2}{1}, \overset{3}{7}, \overset{4}{6}, \overset{5}{2}, \overset{6}{2}, \overset{7}{4} \}$$

and bins of size ($C=$)10, pack the items into as few bins as possible



$\overset{i:}{\underset{1}{<1}}, \overset{2}{2}, \overset{3}{2}, \overset{4}{3}, \overset{5}{1}, \overset{6}{2}, \overset{7}{3} >$

Example Domain: Bin Packing

Low Level Mutational Heuristics



Bin Packing MU₀ RC₀ RC₁ MU₁ HC₀ MU₂ HC₁ XO₀

- Parameter ‘intensity of mutation’
 - ▶ [0.2: repeat 1 time] – [1.0: repeat 5 times]
- **Swap (MU₀)**: Select two different pieces at random, and swap them if there is space.
- **Split a Bin (MU₁)**: This heuristic selects a bin at random from those with more pieces than the average. It then splits this bin into two bins, each containing half of the pieces from the original bin.
- **Rearrange the Lowest Filled Bin (MU₂)**: Remove all of the pieces from the lowest filled bin, and repack them into the other bins if possible, with the best-fit heuristic.

Example Domain: Bin Packing

Low Level Local Search Heuristics



Bin Packing MU₀ RC₀ RC₁ MU₁ HC₀ MU₂ HC₁ XO₀

- Parameter ‘depth of search’
 - ▶ 0.2: repeat 10 times – 1.0: repeat 20 times
- **First-improvement/IE – Swap (HC₀)** : Select two different pieces at random, and swap them if there is space, and if it will produce an improvement in fitness.
- **First-improvement/IE – Swap from Lowest Bin (HC₁)**: Take the largest piece from the lowest filled bin, and exchange with a smaller piece from a randomly selected bin.

Example Domain: Bin Packing

Low Level Ruin&Recreate Heuristics



- Parameter ‘intensity of mutation’
 - ▶ 0.2: $x=3$, 0.4: $x=6$, 0.6: $x=9$, 0.8: $x=12$, 1.0: $x=15$
- **Destroy x Highest Bins (RC_0)**: Remove all the pieces from the x highest filled bins
- **Destroy x Lowest Bins (RC_1)**: Remove all the pieces from the x lowest filled bins

Crossover

- **(XO_0): Exon Shuffling Crossover**

Philipp Rohlfshagen and John Bullinaria. A genetic algorithm with exon shuffling crossover for hard bin packing problems. In Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO'07), pages 1365–1371, London, U.K., 2007.



The University of
Nottingham

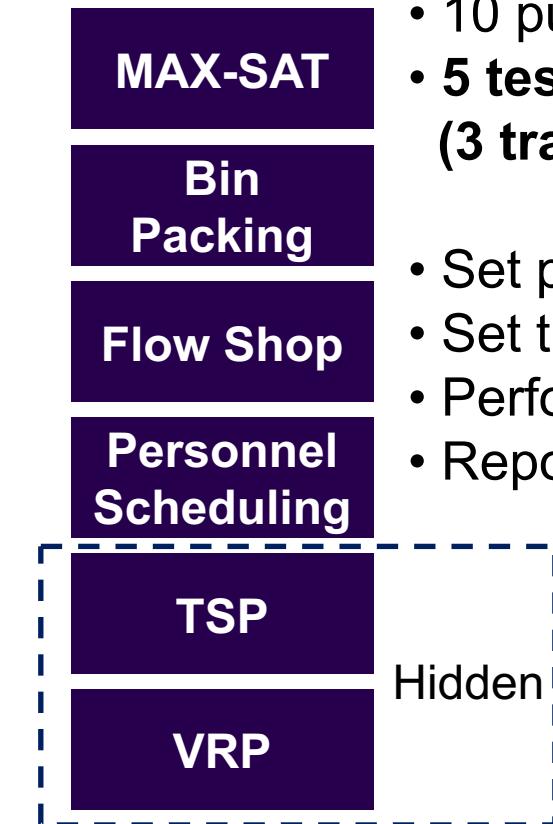
Organising Partners:



Sponsor:



EVENTMAP®



- 10 public training instances
- **5 test instances**
(3 training + 2 hidden/all hidden)
- Set problem instance
- Set time limit (10 min.)
- Perform 31 runs
- Report median

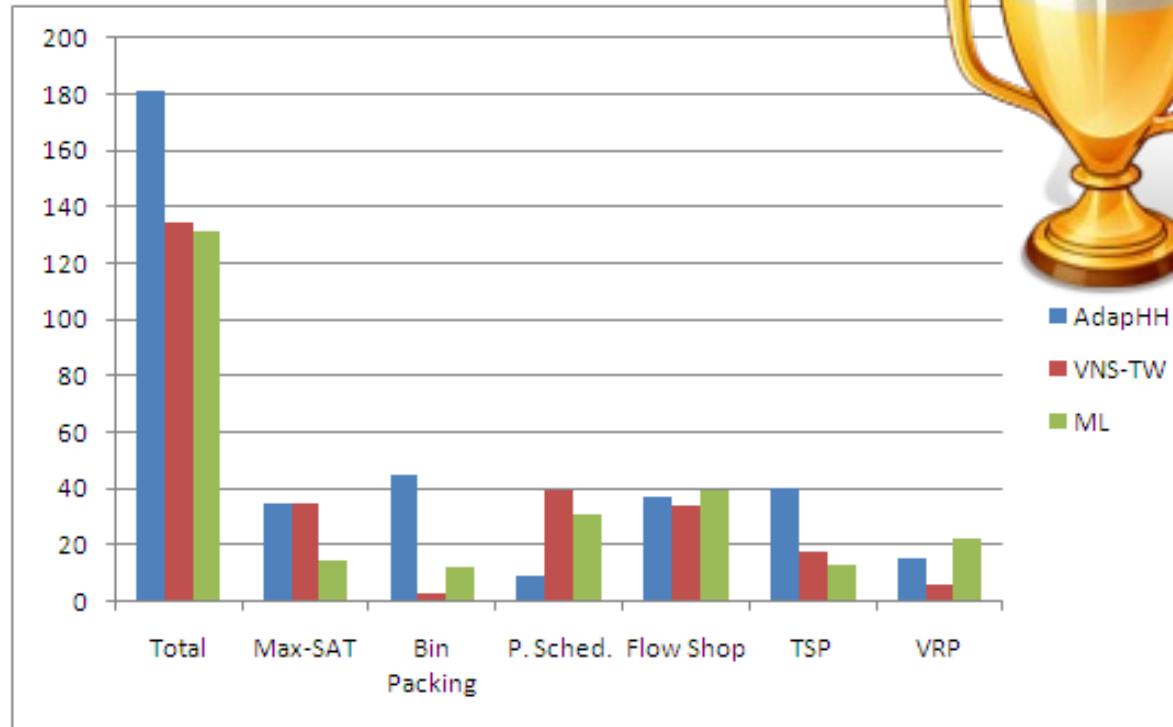
Ranking: Formula 1
scoring system



Rank	Score
1	10
2	8
3	6
4	5
5	4
6	3
7	2
8	1
rest	0



And the winner is...



**AdapHH – M. Mısır
K. Verbeeck
P. De Causmaecker
G. Vanden Berghe**

AdapHH – Overview



$$\begin{aligned}
 p_i &= w_1 \left[\left(C_{p,best}(i) + 1 \right)^2 \left(t_{remain}/t_{p,spent}(i) \right) \right] \times b + \\
 &\quad w_2 \left(f_{p,imp}(i)/t_{p,spent}(i) \right) - w_3 \left(f_{p,wrs}(i)/t_{p,spent}(i) \right) + \\
 &\quad w_4 \left(f_{imp}(i)/t_{spent}(i) \right) - w_5 \left(f_{wrs}(i)/t_{spent}(i) \right)
 \end{aligned}$$

$$\begin{aligned}
 b &= \begin{cases} 1, & \sum_{i=0}^n C_{p,best}(i) > 0 \\ 0, & \text{otw.} \end{cases} \\
 avg &= \left\lfloor \left(\sum_i^n QI_i \right) / n \right\rfloor
 \end{aligned}$$

$$pl = ph_{duration}/t_{subset}$$

$$ph_{duration} = t_{total}/ph_{requested}$$

$$exc(i) = t_{perMove}(i)/t_{perMove}(fastest)$$

$$\sigma > 2.0 ; exc(i) > 2\varpi ; nb > 1$$

$$pr_i = ((C_{best}(i) + 1)/t_{spent})^{(1+3tf^3)}$$

$$k = \begin{cases} ((l-1).k + iter_{elapsed})/l, & \text{if } cw = 0 \\ ((l-1).k + \sum_{i=0}^{cw} k.0.5^i.tf)/l, & \text{otherwise} \end{cases}$$

$$tf = (t_{exec} - t_{elapsed})/t_{exec}$$

$$cw = iter_{elapsed}/k$$

Relay hybridisation

Input: $list_size = 10; \gamma \in (0.02, 50); p, p' \in [0 : 1]$

```

1  $\gamma = (C_{best,s} + 1)/(C_{best,r} + 1)$ 
2 if  $p \leq (C_{phase}/pl)^\gamma$  then
3   select LLH using a LA and apply to  $S \rightarrow S'$ 
4   if  $size(list_i) > 0$  and  $p' \leq 0.25$  then
5     | select a LLH from  $list_i$  and apply to  $S' \rightarrow S''$ 
6   else
7     | select a LLH and apply to  $S' \rightarrow S''$ 
  end
end

```

$$l = l_{base} + (l_{initial} - l_{base} + 1)tf^3$$

AILLA move acceptance

Input: $i = 1, K \geq k \geq 0, l > 0$
for $i=0$ to $l-1$ do $bestList(i) = f(S_{initial})$

```

1 if adapt_iterations  $\geq K$  then
2   if  $i < l - 1$  then
3     |  $i++$ 
4   end
5 if  $f(S') < f(S)$  then
6    $S \leftarrow S'$ 
7   w_iterations = 0
8   if  $f(S') < f(S_b)$  then
9     |  $i = 1$ 
10    |  $S_b \leftarrow S'$ 
11    | w_iterations = adapt_iterations = 0
12    | bestList.remove(last)
13    | bestList.add(0, f(S_b))
14  end
15 else if  $f(S') = f(S)$  then
16  |  $S \leftarrow S'$ 
17 else
18  | w_iterations ++
19  | adapt_iterations ++
20  | if w_iterations  $\geq k$  and  $f(S') \leq bestList(i)$  then
21    | |  $S \leftarrow S'$  and w_iterations = 0
22  end
end

```



CHeSC Results

Rank	Hyper-heuristic	Score	Rank	Hyper-heuristic	Score
1	AdapHH	181.00	11	ACO-HH	39.00
2	VNS-TW	134.00	12	GenHive	36.50
3	ML	131.50	13	DynILS	27.00
4	PHUNTER	93.25	14	SA-ILS	24.25
5	EPH	89.75	15	XCJ	22.50
6	HAHA	75.75	16	AVEG-Nep	21.00
7	NAHH	75.00	17	GISS	16.75
8	ISEA	71.00	18	SelfSearch	7.00
9	KSATS-HH	66.50	19	MCHH-S	4.75
10	HAEA	53.50	20	Ant-Q	0.00



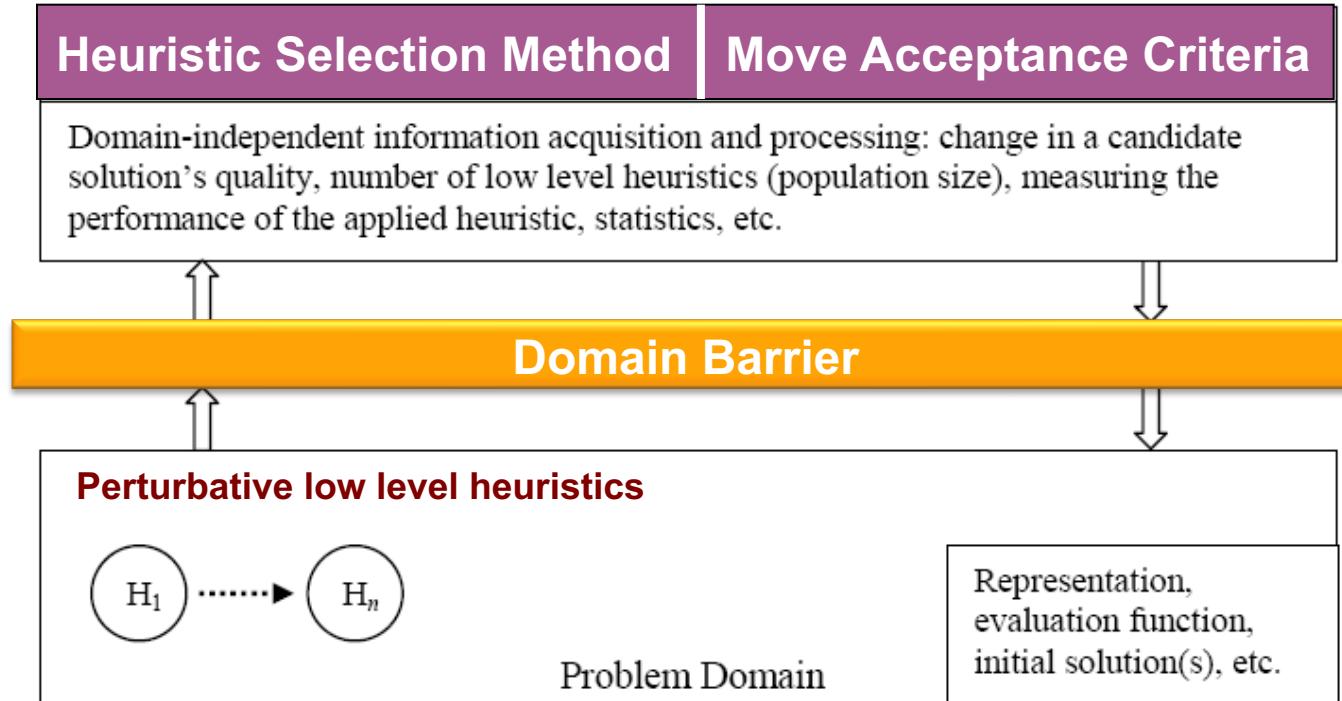
Extended HyFlex Problem Domains

- There are 3 new problem domains, each with 10 instances added to HyFlex benchmark

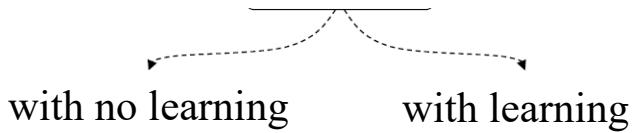
problem domain	abbrev.	init	ls	mut	rr	xo	total
Max-SAT	SAT	1	2	4	1	2	10
Bin Packing	BP	1	2	3	2	1	9
Permutation Flow Shop	PFS	1	4	5	2	3	15
Personnel Scheduling	PSP	1	4	1	3	3	12
Traveling Salesman	TSP	1	6	5	1	3	16
Vehicle Routing	VRP	1	4	4	2	2	13
0-1 Knap Sack	KP	1	6	5	2	3	17
Quadratic Assignment	QAP	1	2	2	3	2	10
Max-Cut	MAC	1	3	2	3	2	11

* S. Adriaensen, G. Ochoa, and A. Nowe, A benchmark set extension and comparative study for the hyex framework, In IEEE Congress on Evolutionary Computation, 2015, pp. 784-791.

A Selection Hyper-heuristic Framework – Single Point Search (revisited)



Heuristic (Operator) Selection

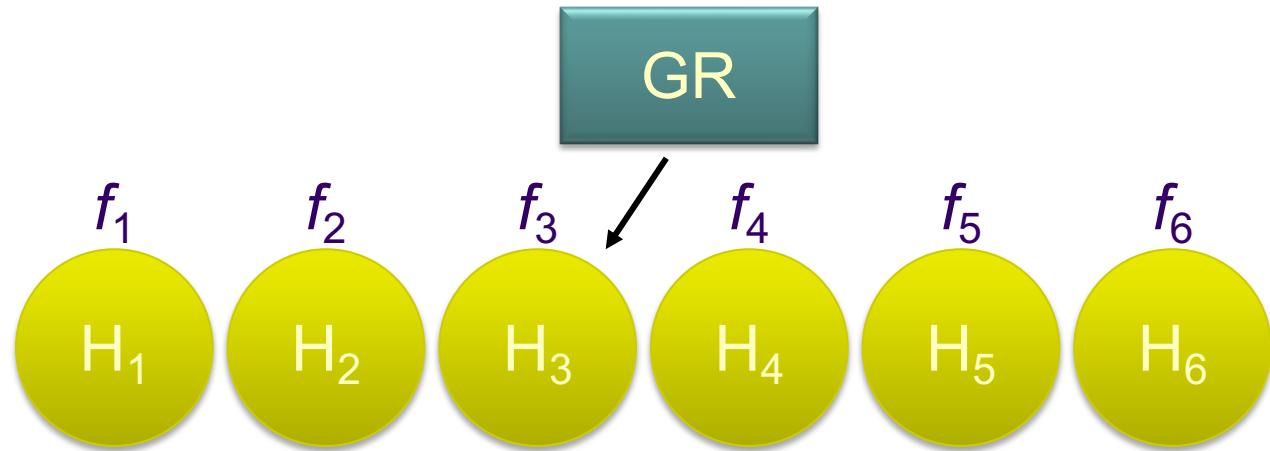


Component name	Reference(s)
Heuristic selection with no learning	
➡ Simple Random	Cowling et al (2000, 2002b)
➡ Random Permutation	Cowling et al (2000, 2002b)
Heuristic selection with learning	
Peckish	Cowling and Chakhlevitch (2003)
Greedy	Cowling et al (2000, 2002b); Cowling and Chakhlevitch (2003)
➡ Random Gradient	Cowling et al (2000, 2002b)
➡ Random Permutation Gradient	Cowling et al (2000, 2002b); Cowling et al (2000, 2002b); Maashi et al (2015); Drake et al (2015)
Choice Function	
Reinforcement Learning	Nareyek (2003); Pisinger and Ropke (2007)
Reinforcement Learning with Tabu Search	Burke et al (2003); Dowsland et al (2007)
Quality Index and Tabu based Learning Heuristic Selection	Misir et al (2009, 2012)
Dominance-based Selection	Kheiri and Özcan (2011; 2015)
Probability-based Selection	Lehrbaum and Musliu (2012)
Adaptive pursuit	Walker et al (2012)

Heuristic Selection – Greedy (GR)



- Apply each low level heuristic to the candidate solution and choose the one that generates the best objective value



$f_3 < f_1, f_2, f_4, f_5, f_6$ at step n

Heuristic Selection – Reinforcement Learning (RL)

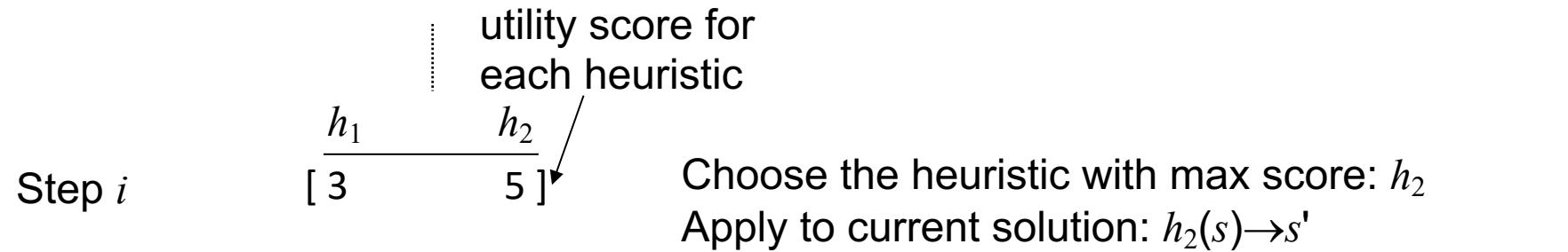


- A machine learning technique
- Inspired by related psychological theory
 - ▶ Reward and punishment
- Concerned with how an agent ought to take actions in an environment to maximize some notion of long-term reward
- Maintains a score for each heuristic
 - ▶ If an improving move then increase (e.g., +1), otherwise decrease (e.g., -1) the score of the heuristic



RL – Example Iteration

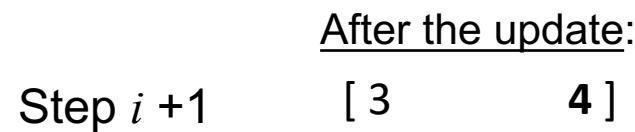
- There are 2 low level heuristics in a given domain implementation



Update scores for the next step:

If s' is s better than (or equal) s (improvement)
then give reward to $h_2 \uparrow$ score (+1)
Otherwise, penalise (worsening) $h_2 \downarrow$ score (-1)

Assuming h_2 creates a worsening solution,
its score is decreased by 1: $(5-1)=4$



Heuristic Selection – Choice Function (CF)

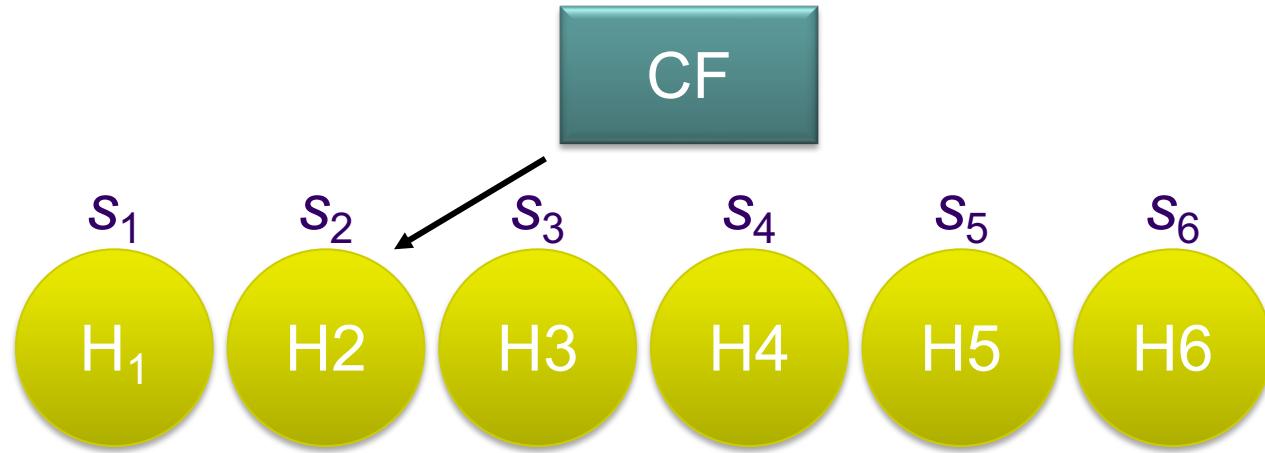


- The choice function maintains a record of the performance of each heuristic. Three criteria are maintained:
 - 1) Its individual performance
 - 2) how well it has performed with other heuristics
 - 3) the elapsed time since the heuristic has been called ($f_3(h_j)$)

$$f_1(h_j) = \sum_n \alpha^{n-1} \frac{I_n(h_j)}{T_n(h_j)} \quad f_2(h_k, h_j) = \sum_n \beta^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)}$$

$$F_n(h_j) = \alpha_n f_1(h_j) + \beta_n f_2(h_k, h_j) + \gamma_n f_3(h_j)$$

Heuristic Selection – Choice Function (CF)



$s_2 > s_1, s_3, s_4, s_5, s_6$ at step n

Simplified CF – Example Iteration

f_1 and f_2 are performance scores per iteration where α, β, γ are fixed (oversimplified for illustrative purposes)



- There are 2 low level heuristics in a given domain implementation.

		Represents the heuristic invoked in previous step	
		h_1	h_2
Step i	↓	[3 5]	f_1
h_1	[1 5]	f_2	
h_2	[2 3]	f_3	
	[7 4]	f_3	

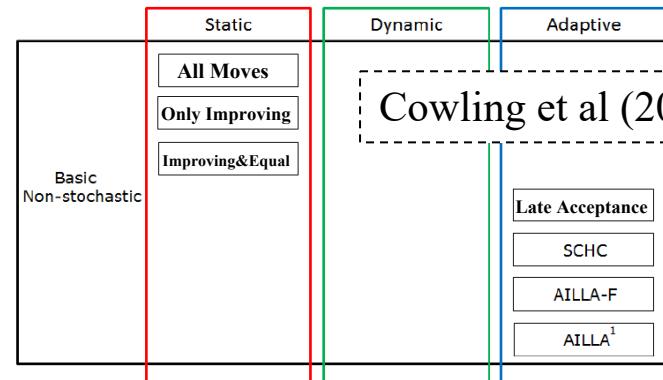
Assume that at **Step (i -1)**, h_2 was invoked and $\alpha=1.0, \beta=0.5, \gamma=1.0$ are fixed for all steps, so
 $F_i(h_1)=f_1(h_1) + 0.5 f_2(h_2, h_1) + f_3(h_1)= 3+0.5*2+7 = 11$
 $F_i(h_2)=f_1(h_2) + 0.5 f_2(h_2, h_2) + f_3(h_2)= 5+0.5*3+4 = 10.5$
Choose the heuristic with max F_i score: h_1
Apply to current solution: $h_1(s) \rightarrow s'$

Update scores for the next step:
If s' is s better than (or equal) s (improvement)
then give reward to $f_1(h_1)$ & $f_2(h_2, h_1)$ ↑scores (+1)
Otherwise, penalise them both by ↓scores (-1)

		After the updates:
		h_1 h_2
Step $i+1$	[4 5]	f_1
h_1	[1 5]	f_2
h_2	[3 3]	f_3
	[0 4]	f_3

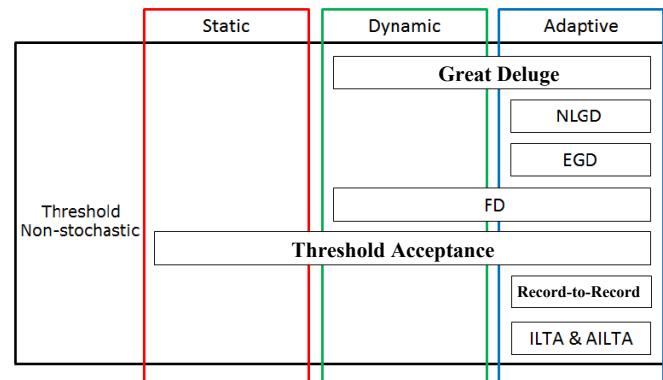
Assuming h_1 creates an improving solution,
 $f_1(h_1)$ and $f_2(h_2, h_1)$ scores are increased by 1:
 $(3+1)=4$ and $(2+1)=3$, also since h_1 is used now, $f_3(h_1)$ is reset to 0.

Move Acceptance

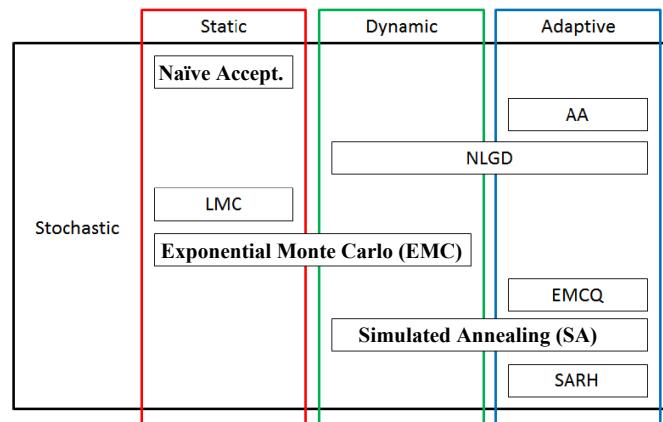


Özcan et al (2009);
Jackson et al. (2013)

Mısır et al (2012)

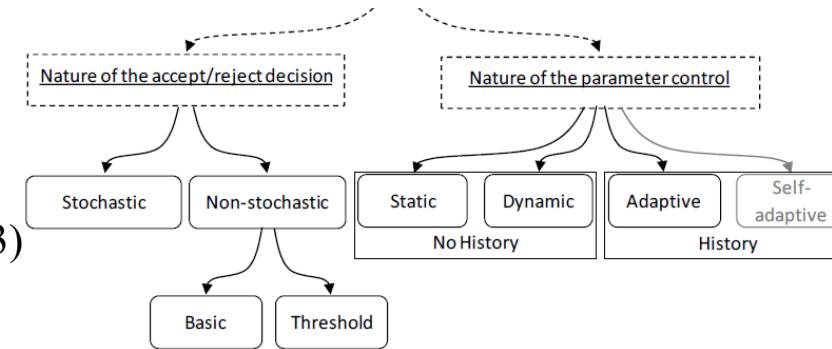


Kheiri and Özcan (2015)
Kendall and Mohamad (2004b)
Mısır et al (2009)



Burke et al (2010); Kheiri and Özcan (2012); Asta and Özcan (2015)

Ayob and Kendall (2003)
Bai and Kendall (2005); Bilgin et al (2006); Pisinger and Ropke (2007);
Antunes et al (2009)
Dowsland et al (2007); Bai et al. (2007a)



A Few Misconceptions about Hyper-heuristics – Personal View



- Hyper-heuristics do not require parameter tuning
- Hyper-heuristics are all tested under a fair setting (hyflex)
 - ▶ Time allocated (and instances used) for tuning
 - ▶ Reusable vs disposable heuristics
- Applying a hyper-heuristic to a new domain is easy
- Domain specific information should not be passed to the hyper-heuristics (objective value is *not* a domain specific information, *all others* are)

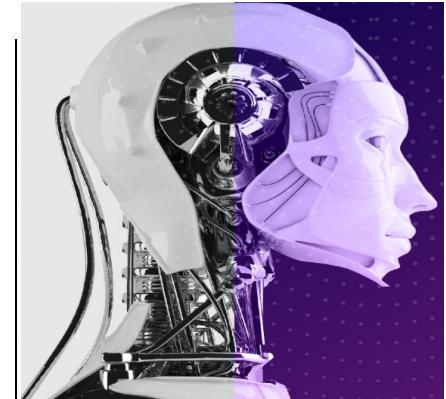
Focus of Recent Studies on Selection Hyper-heuristics



- 1) Existing hyper-heuristics/metaheuristics for cross-domain search are tuned/configured for performance improvement
- 2) Existing hyper-heuristics are applied to new problem domains
- 3) “New” hyper-heuristics are created and tested
 - ▶ Hyper²-heuristics (based on metaheuristics and data science techniques) are emerging
 - Multi-stage approaches (iterated)
 - Generated hyper-heuristics
 - ▶ Multi-objective hyper-heuristics
- 4) Hybrid approaches are appearing (e.g., exact&inexact, constructive&perturbative, etc...)

An Iterated Multi-stage Selection Hyper-heuristic

A. Kheiri and E. Özcan, An Iterated Multi-stage Selection Hyper-heuristic, European Journal of Operational Research, (250)1:77–90, 2016



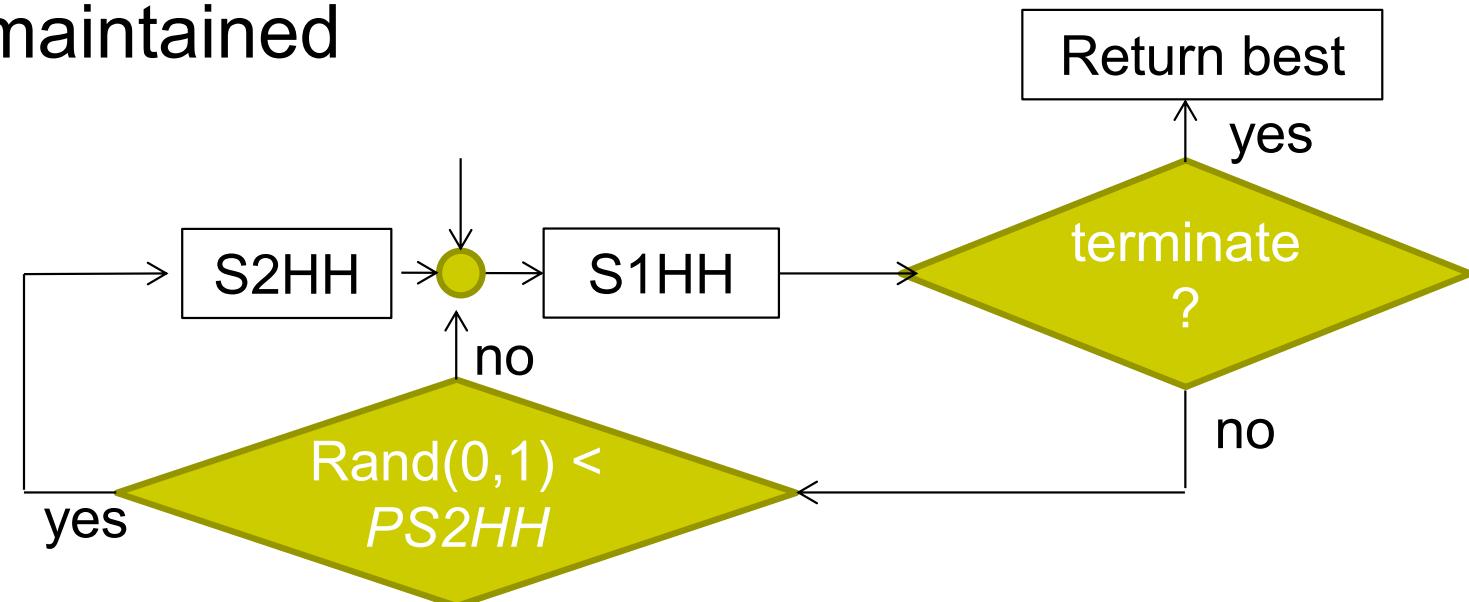
The University
of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA



General Framework

- Crossover operators are ignored
- The parameter value of a low level heuristic is randomly reset if there is no improvement after its application, otherwise the same value is maintained





Stage 1 Hyper-heuristic

- Select a low level heuristic i with probability

$$score_i / \sum_{\forall k} (score_k)$$

- Apply the chosen heuristic
- Accept/reject based on an adaptive threshold acceptance method
- Stage 1 terminates if a duration of s_1 is exceeded without any improvement

An Adaptive Threshold Move Acceptance Method

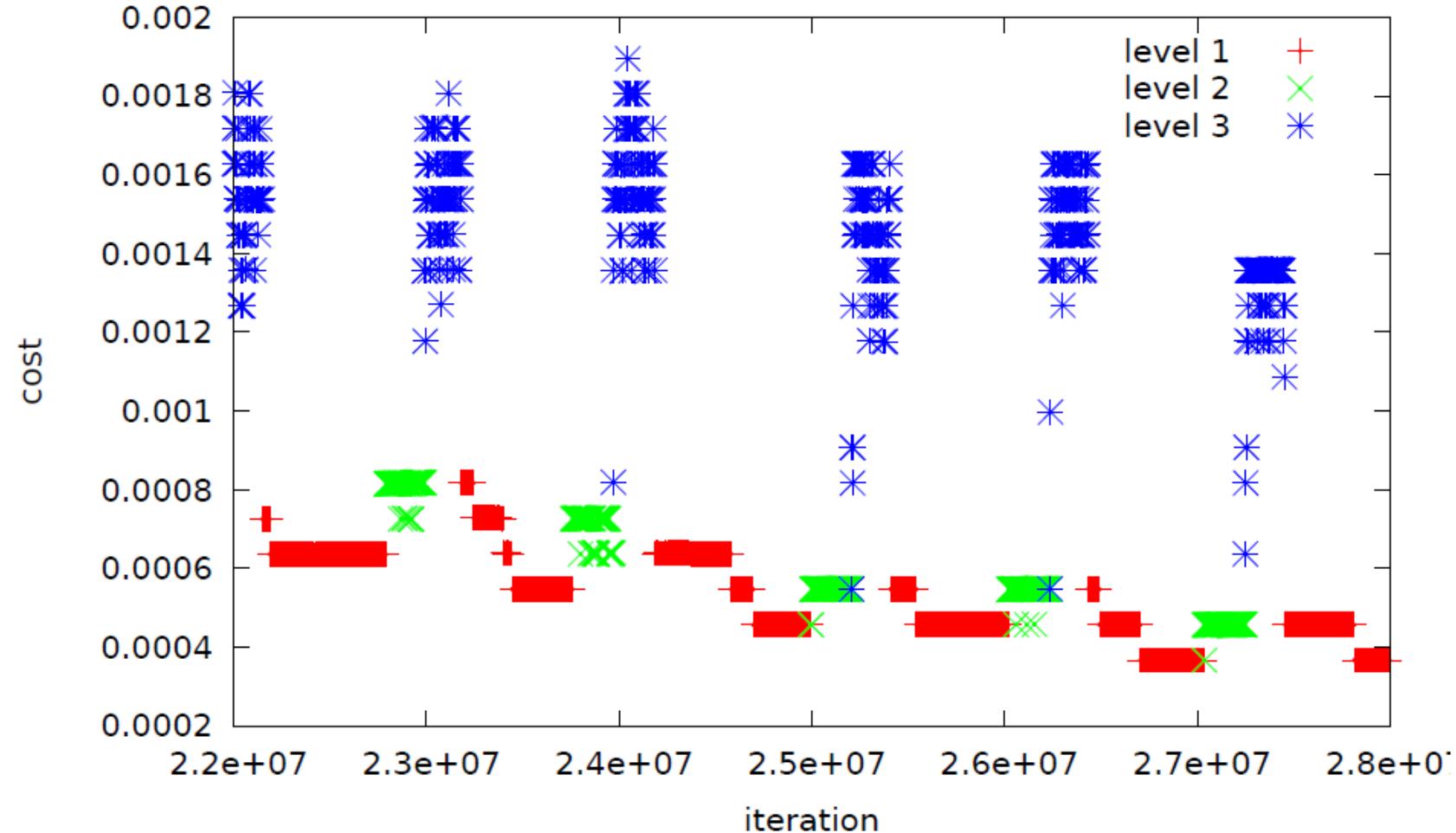


- Moves are accepted using the criteria

```
if  $S_{new}$  isBetterThan  $S_{current}$  OR  $f(S_{new})$  isBetterThan  $(1 + \epsilon)f(S_{beststage})$ 
then
    |  $S_{current} \leftarrow S_{new}$ 
end
```

- where $\epsilon = \frac{\lfloor \log(f(S_{beststage})) \rfloor + c_i}{f(S_{beststage})}$, C is a circular list, and c_i is an integer value in $C = \{c_0, \dots, c_i, \dots, c_{(k-1)}\}$ updated before/in Stage 2, fixed in Stage 1
- ϵ is updated if no improvement for a duration of d

Behaviour of Move Acceptance – Illustration



Stage 2 Hyper-heuristic



Given N LLHs, e.g., $\text{LLH}_1, \text{LLH}_2$

Pair up all and increase the number of LLHs to $N+N^2$

$$\text{LLH}_3 \leftarrow \text{LLH}_1 + \text{LLH}_1$$

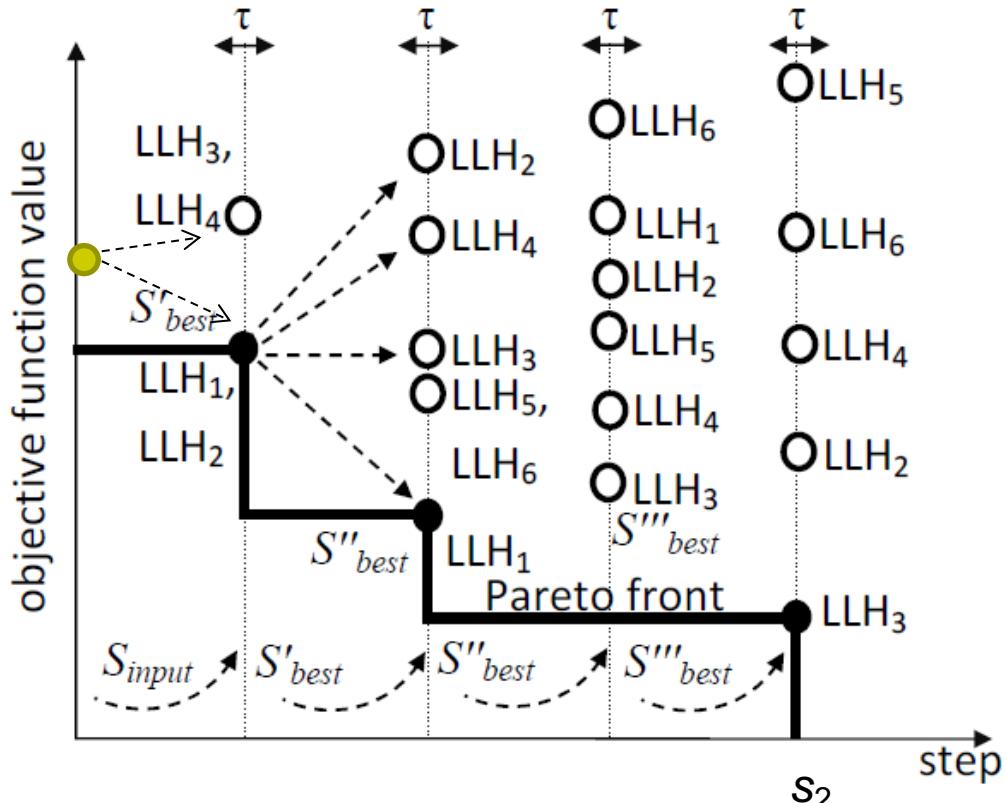
$$\text{LLH}_4 \leftarrow \text{LLH}_2 + \text{LLH}_2$$

$$\text{LLH}_5 \leftarrow \text{LLH}_1 + \text{LLH}_2$$

$$\text{LLH}_6 \leftarrow \text{LLH}_2 + \text{LLH}_1$$

Reduce the Number of LLHs ($N+N^2 \rightarrow n$) + Assign Probabilities

$\text{LLH}_1=2, \text{LLH}_2=1, \text{LLH}_3=1$
50% 25% 25%



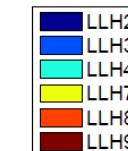
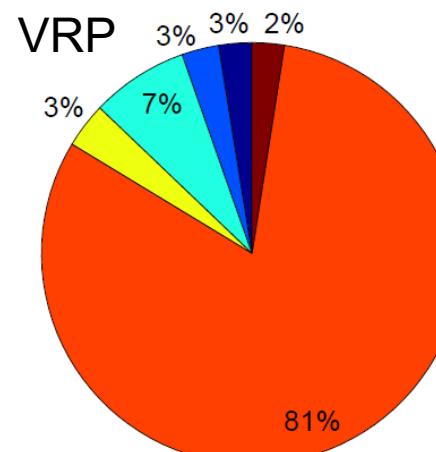
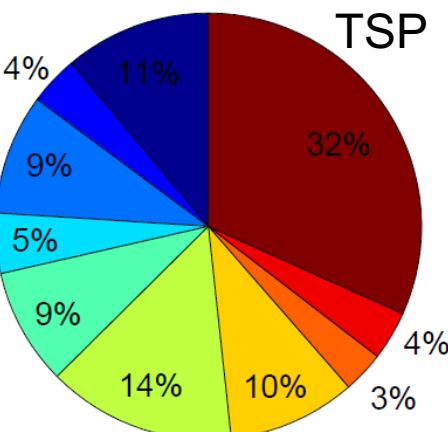
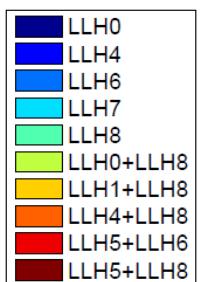
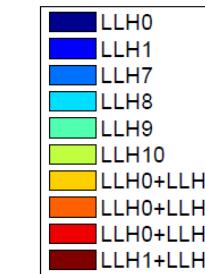
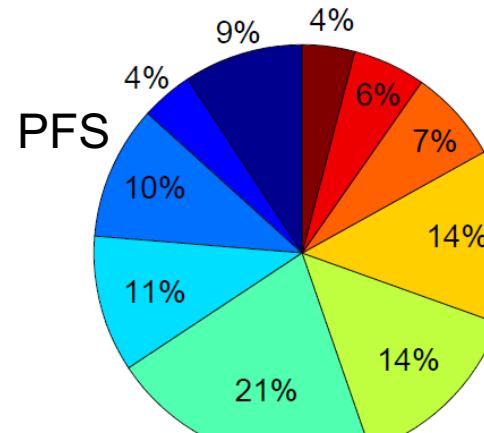
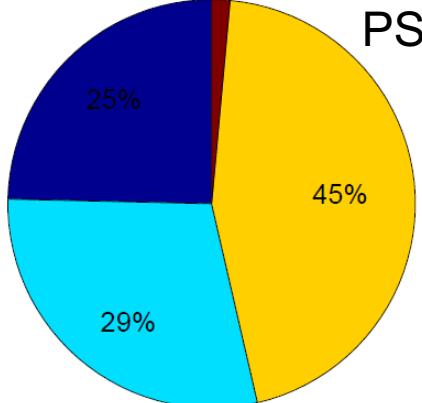
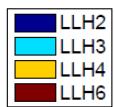
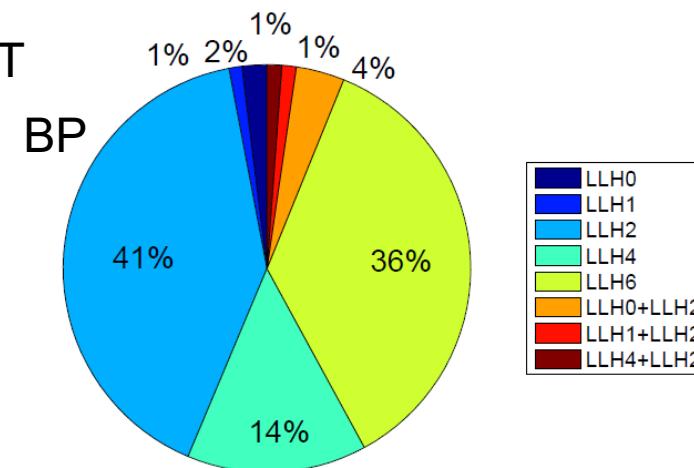
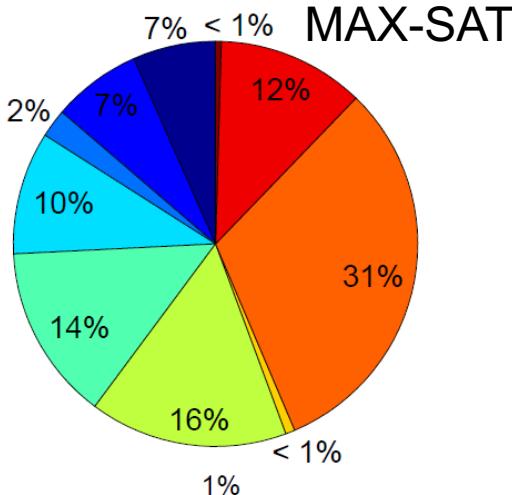
6 LLHs \rightarrow 3 LLHs



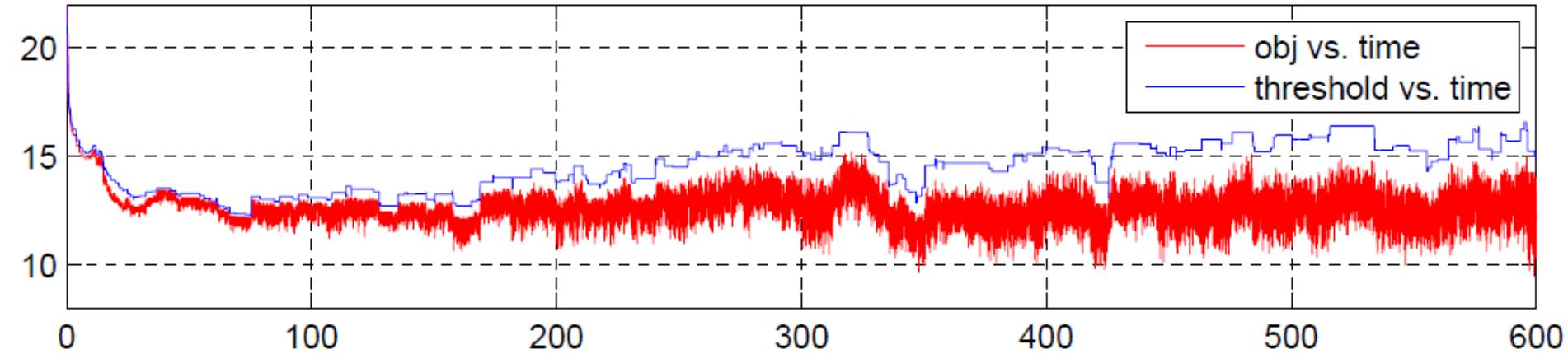
Parameter Tuning

- The proposed approach introduces 6 system parameters to be set
 - $\tau = \{10, 15, 20, 30\}$ (in milliseconds)
 - $d = \{7, 9, 10, 12\}$ (in seconds)
 - $s_1 = \{10, 15, 20, 25\}$ (in seconds)
 - $s_2 = \{3, 5, 10, 15\}$ (in steps/iterations)
 - $P_{S2HH} = \{0.1, 0.3, 0.6, 0.9, 1.0\}$
 - $C = \{\{0\}, \{3\}, \{6\}, \{9\}, \{0, 3, 6, 9\}\}$

Relay hybridisation

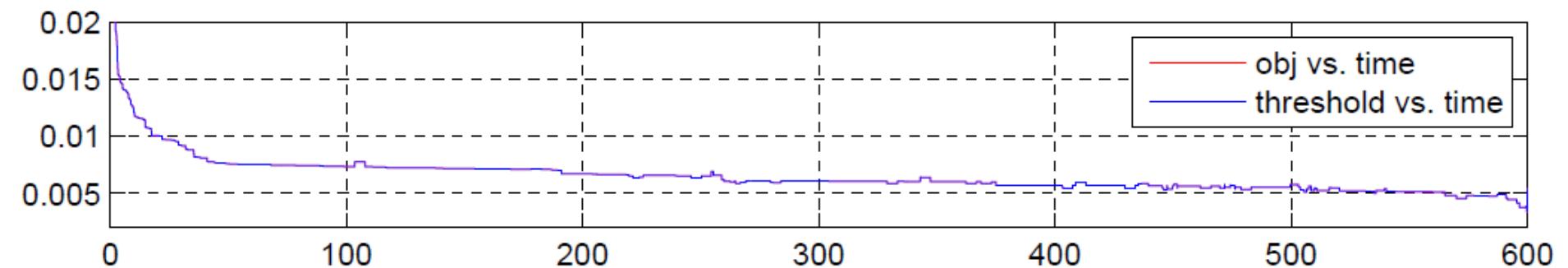


MAX-SAT





Bin Packing





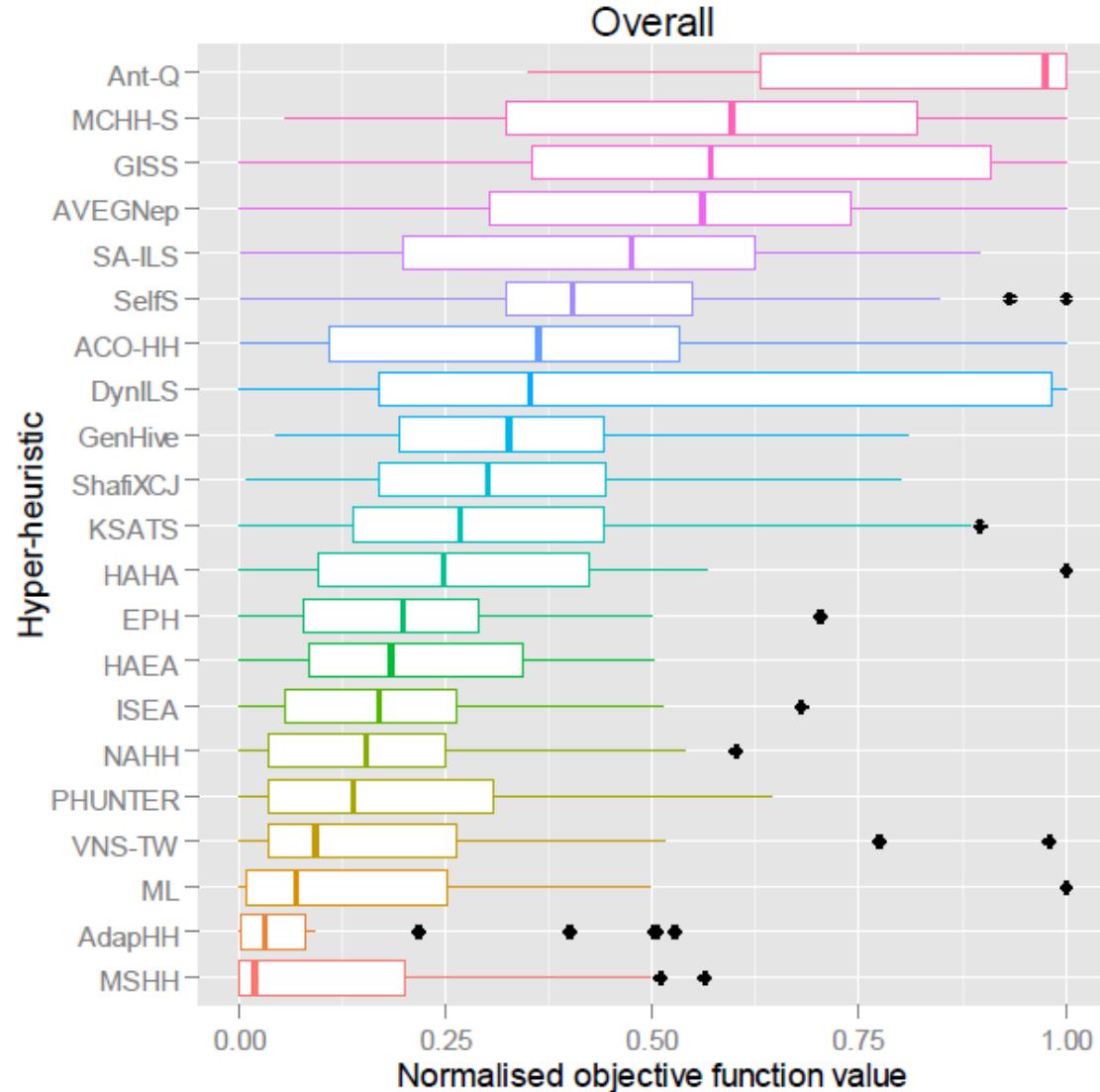
Performance Comparison

		MSHH				S1HH				S2HH			
Domain	Instance	avg.	std.	median	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.
SAT	Inst1	0.9	0.7	1.0	0.0	>	6.4	4.5	1.0	>	15.0	4.6	3.0
	Inst2	3.1	3.9	2.0	1.0	>	21.3	13.3	3.0	>	44.9	9.8	18.0
	Inst3	0.7	0.5	1.0	0.0	>	7.1	7.7	0.0	>	26.3	14.0	1.0
	Inst4	1.7	1.0	1.0	1.0	>	5.7	4.3	1.0	>	20.0	4.6	12.0
	Inst5	7.6	0.9	7.0	7.0	>	10.4	1.5	7.0	>	15.4	1.7	13.0
BP	Inst1	0.0163	0.0014	0.0163	0.0136	<	0.0159	0.0010	0.0137	>	0.0198	0.0015	0.0160
	Inst2	0.0037	0.0015	0.0030	0.0025	>	0.0061	0.0015	0.0034	>	0.0104	0.0021	0.0077
	Inst3	0.0050	0.0015	0.0049	0.0025	>	0.0054	0.0012	0.0027	>	0.0128	0.0011	0.0104
	Inst4	0.1084	0.0000	0.1084	0.1083	<=	0.1084	0.0000	0.1083	>	0.1084	0.0000	0.1084
	Inst5	0.0050	0.0019	0.0044	0.0032	>	0.0055	0.0021	0.0032	>	0.0210	0.0015	0.0187
PS	Inst1	25.5	4.5	25.0	16.0	>	28.8	4.7	18.0	>	31.6	4.9	22.0
	Inst2	9668.9	217.8	9638.0	9184.0	<=	9645.3	159.6	9334.0	<	9645.8	106.7	9391.0
	Inst3	3283.7	93.3	3270.0	3132.0	>	3304.8	99.6	3134.0	>	3309.9	110.2	3172.0
	Inst4	1786.3	172.1	1760.0	1545.0	>	1801.0	142.3	1570.0	>	1836.0	291.1	1400.0
	Inst5	353.2	21.2	350.0	315.0	>	724.4	657.3	320.0	>	810.7	621.5	360.0
PFS	Inst1	6239.8	14.9	6239.0	6212.0	>	6287.6	21.9	6249.0	>	6353.3	29.8	6301.0
	Inst2	26895.2	55.3	26889.0	26775.0	<	26873.2	30.7	26822.0	>	26976.9	54.7	26849.0
	Inst3	6333.8	19.0	6325.0	6303.0	>	6360.5	16.4	6323.0	>	6405.5	23.7	6369.0
	Inst4	11363.8	32.7	11359.0	11320.0	>	11429.9	43.8	11357.0	>	11529.3	35.9	11436.0
	Inst5	26711.9	47.0	26709.0	26630.0	<	26693.1	40.7	26608.0	>	26779.1	49.8	26702.0
TSP	Inst1	48208.1	31.8	48194.9	48194.9	>	50032.0	571.1	49263.1	>	50326.5	606.6	49221.6
	Inst2	2.09e+7	9.05e+4	2.09e+7	2.07e+7	>	2.14e+7	1.12e+5	2.12e+7	>	2.13e+7	1.05e+5	2.11e+7
	Inst3	6809.1	7.1	6808.8	6796.6	>	7012.5	30.4	6964.6	>	7040.2	31.3	6988.6
	Inst4	66840.2	276.5	66843.6	66236.8	>	68908.4	382.4	68159.9	>	70241.9	704.6	68791.0
	Inst5	53011.4	469.7	52910.2	52341.3	>	54411.1	595.1	53686.0	>	55814.8	946.4	53992.4
VRP	Inst1	70998.4	3840.3	70506.5	63948.2	<	70223.0	2960.2	64273.2	>	84103.9	7225.8	68958.3
	Inst2	13421.8	251.6	13359.6	13303.9	>	13658.0	471.4	13319.6	>	13695.8	473.9	13320.0
	Inst3	148498.2	1625.8	148436.2	145466.5	<	148232.6	1935.3	145426.5	>	149553.2	2377.8	145362.7
	Inst4	21016.4	488.2	20671.4	20650.8	<=	20991.3	478.0	20653.5	>	21131.9	510.3	20657.5
	Inst5	148813.7	1272.5	149193.7	146334.6	>	148999.1	1217.1	146844.9	>	150282.6	1616.3	146666.9

Performance Comparison to CHeSC 2011 competitors



- Top with a CHeSC 2011 score of **163.60**

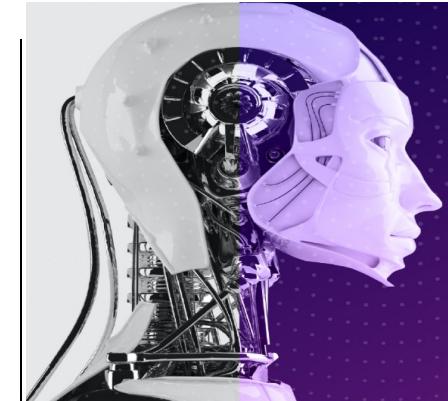




Conclusions

- The proposed use of two hyper-heuristics with effective components synergise well producing an easy-to-implement, easy-to-maintain, and successful hyper-heuristic for cross domain search
- The adaptive threshold move acceptance captures the trade-off between the extent of improvement that a heuristic can generate and the number of steps it takes to achieve that improvement

Parameter Tuning for Cross-domain Heuristic Search



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA



A Cross-domain Parameter Tuning Study

- A previous study shows that a steady state memetic algorithm (SSMA) performs better than its trans-generational variant across 6 problem domains of HyFlex*
- This study
 - ▶ Investigated its performance on 9 problem domains (5 instances from each domain)
 - ▶ Performed parameter tuning using the Taguchi design of experiments (orthogonal array) method

*E. Ozcan, S. Asta and C. Altintas, Memetic Algorithms for Cross-domain Heuristic Search, The 13th Annual Workshop on Computational Intelligence (UKCI), 2013, pp. 175-182.



A Steady State Memetic Algorithm

Algorithm 1 : Pseudocode of Steady-state Memetic Algorithm

Create a population of *popsize* random individuals

Set the parameter values for *Intensity of Mutation* (IoM),
Depth of Search (DoS) and *tournament size* (toursize)

Apply a random local search method (hill climbing) to each individual

while (termination criterion is not satisfied)

 Parent1 \leftarrow Select-Parent(population, toursize)

 Parent2 \leftarrow Select-Parent(population, toursize)

 Child \leftarrow ApplyCrossover (Rand(1, MAX_CROSSOVER), Parent1,
 Parent2)

 Child \leftarrow ApplyMutation (Rand(1, MAX_MUTATION), IoM, Child)

 Child \leftarrow ApplyLocalSearch(Rand(1, MAX_LOCALSEARCH), DoS,
 Child)

 WorstOf(population) \leftarrow Child

end while

Experimental Design

Parameter levels for memetic algorithm

Parameters	Value Options
Intensity of Mutation (IoM)	{0.2, 0.4, 0.6, 0.8, 1.0}
Depth of Search (DoS)	{0.2, 0.4, 0.6, 0.8, 1.0}
Population size (PopSize)	{5, 10, 20, 40, 80}
Tournament size (TourSize)	{2, 3, 4, 5}

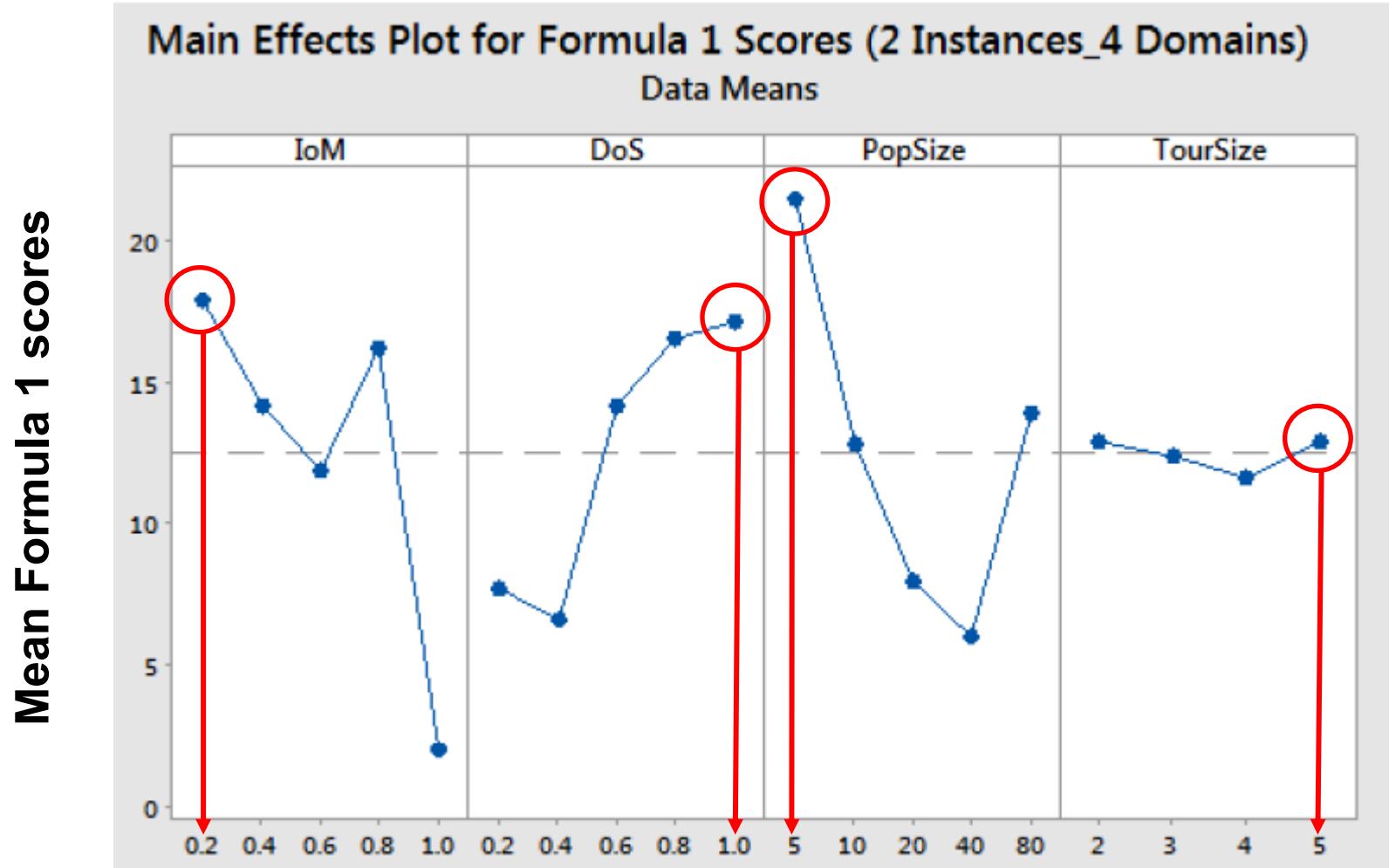
- 2 instances from each of the four public HyFlex problem domains are used for training
- The best configuration after tuning is tested on all instances

Taguchi orthogonal array

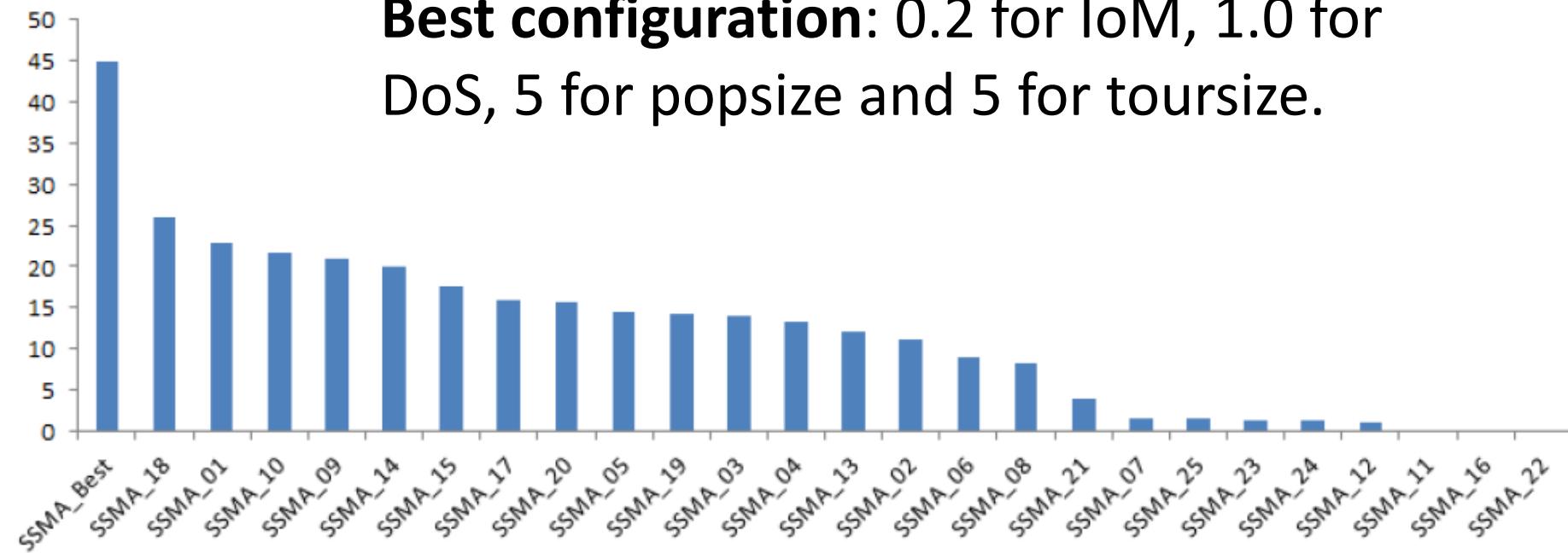
Experiment number	IoM	DoS	Pop size	Tour size
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	1	4	4	4
5	1	5	5	2
6	2	1	2	3
7	2	2	3	4
8	2	3	4	1
9	2	4	5	1
10	2	5	1	2
11	3	1	3	4
12	3	2	4	1
13	3	3	5	2
14	3	4	1	3
15	3	5	2	4
16	4	1	4	2
17	4	2	5	3
18	4	3	1	4
19	4	4	2	2
20	4	5	3	1
21	5	1	5	4
22	5	2	1	3
23	5	3	2	1
24	5	4	3	2
25	5	5	4	3



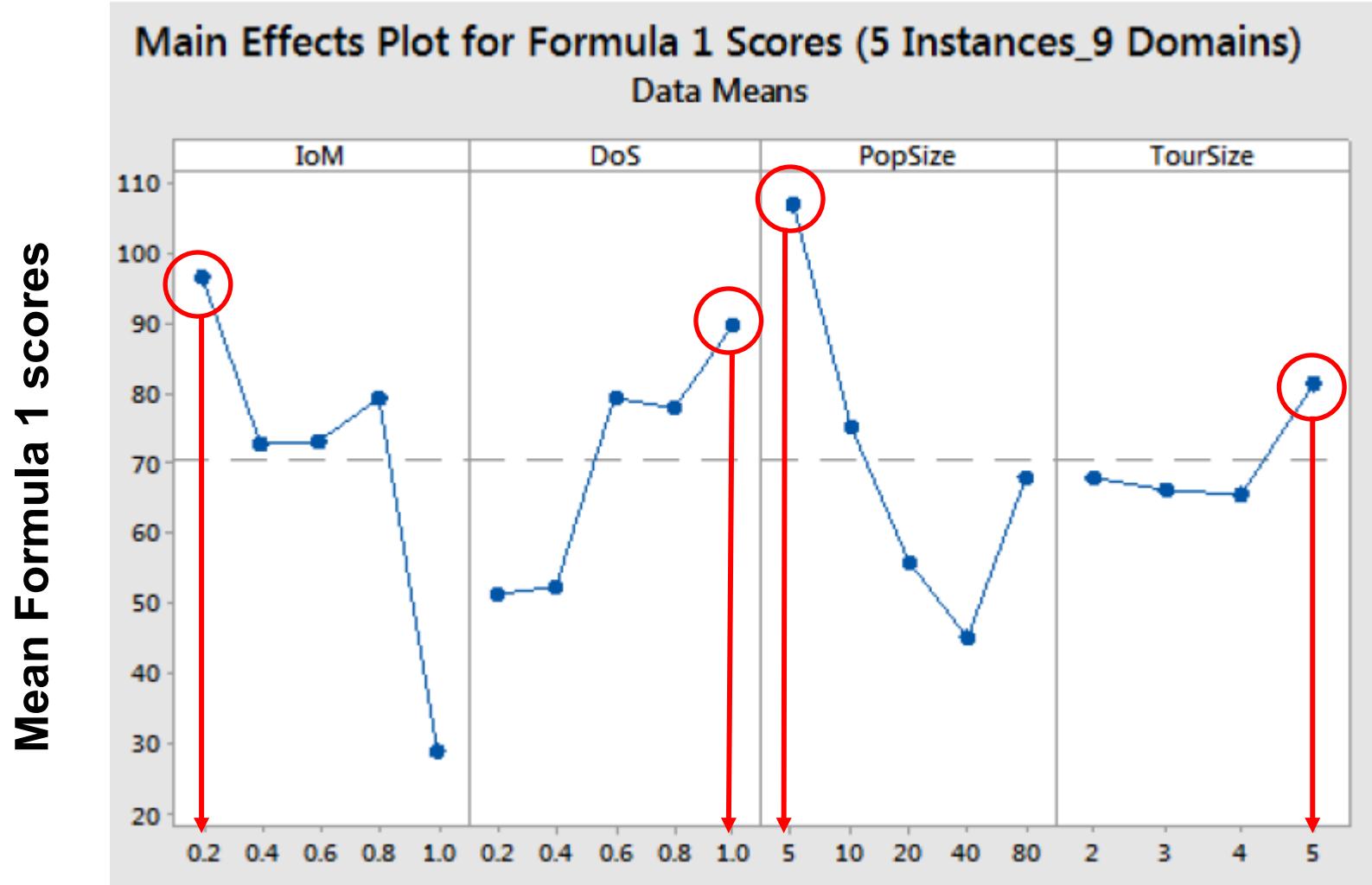
Main Effects Plot – Parameter tuning with 2 instances from 4 public domains



Confirmation Run on the Training Instances



Main Effects Plot – Parameter tuning with 5 instances from 9 public domains



Influence of Parameter Tuning



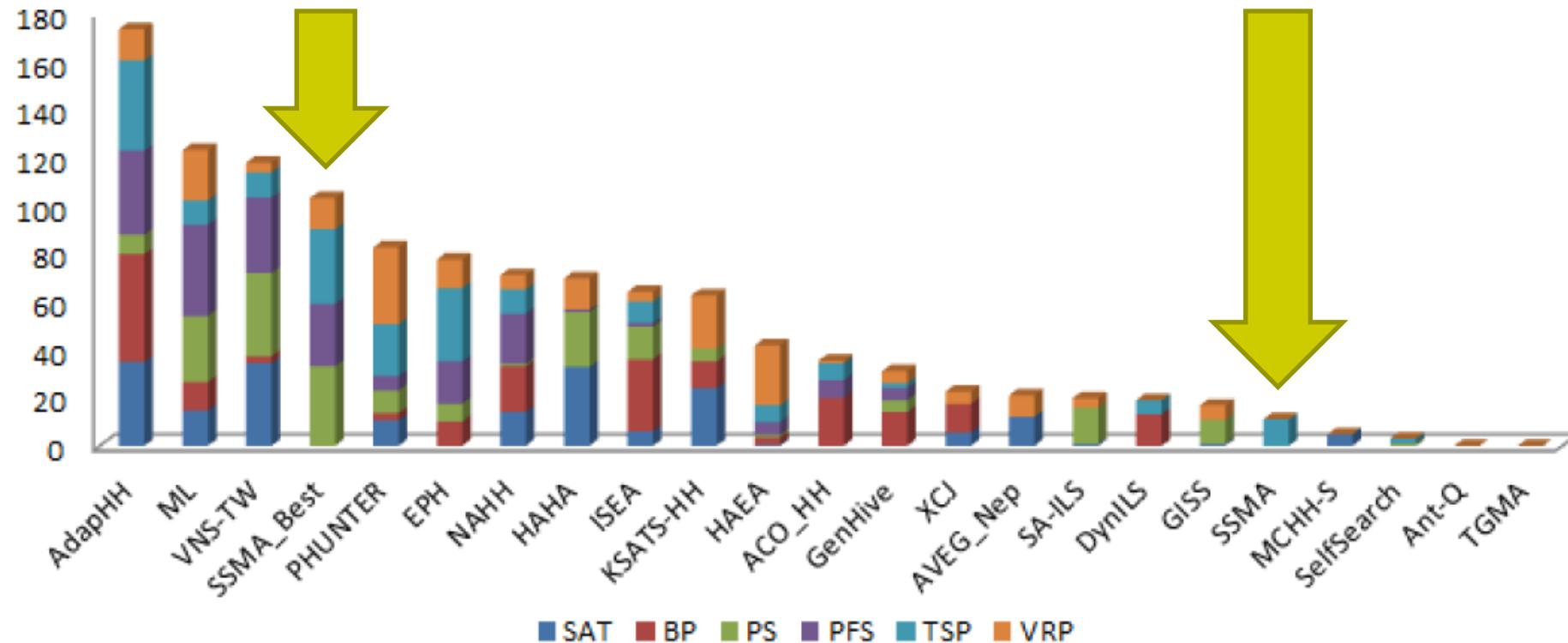
PD	ID	SSMA			SSMA-Best		
		mean	median	best vs.	mean	median	best
SAT	1	21.161	22	8 <	9.484	9	3
	2	52.484	53	37 <	30.065	36	9
	3	35	37	10 <	15.387	10	3
	4	27.742	27	26 <	13.613	13	9
	5	19.194	18	14 <	12.742	13	9
BP	1	0.083	0.082	0.074 <	0.063	0.063	0.058
	2	0.015	0.013	0.011 <	0.011	0.012	0.008
	3	0.022	0.022	0.018 >	0.034	0.034	0.029
	4	0.1115	0.1112	0.1105 <	0.1102	0.11	0.1098
	5	0.043	0.041	0.036 >	0.061	0.06	0.054
PS	1	51.129	50	37 <	21.548	21	16
	2	72015.613	70477	52056 <	9705.129	9677	9477
	3	13203.71	11859	5581 <	3211.452	3219	3146
	4	2942.419	2655	1820 <	1596.871	1589	1344
	5	435.645	431	385 <	318.065	315	290
PFS	1	6257.806	6258	6231 <	6249.581	6251	6219
	2	26884	26884	26813 <	26812.452	26811	26754
	3	6351.871	6363	6318 <	6336.387	6333	6303
	4	11441.806	11441	11410 <	11376.613	11375	11333
	5	26699.226	26703	26626 <	26632.548	26640	26515
TSP	1	48227.747	48194.92	48194.92 ≤	48221.583	48194.92	48194.92
	2	21155458.17	21160875.64	20969185.56 <	20912006.397	20885233.01	20789116.98
	3	6825.552	6825.663	6800.708 <	6811.145	6811.518	6799.111
	4	68123.369	68059.971	67423.655 <	67029.810	67043.255	66518.735
	5	53810.138	53748.537	52685.992 <	53503.576	53457.422	52247.568
VRP	1	71768.053	71480.081	67820.589 ≤	71946.305	70776.497	65967.938
	2	14324.522	14411.658	13358.611 <	13826.295	13384.024	13328.791
	3	176206.081	177131.584	167704.512 <	147512.686	148001.434	143921.208
	4	21647.018	21675.195	20678.096 <	21275.493	21648.051	20654.219
	5	152642.04	152829.839	149032.551 <	147372.783	147228.056	145266.409

Influence of the Setting {toursize=popsize=5}

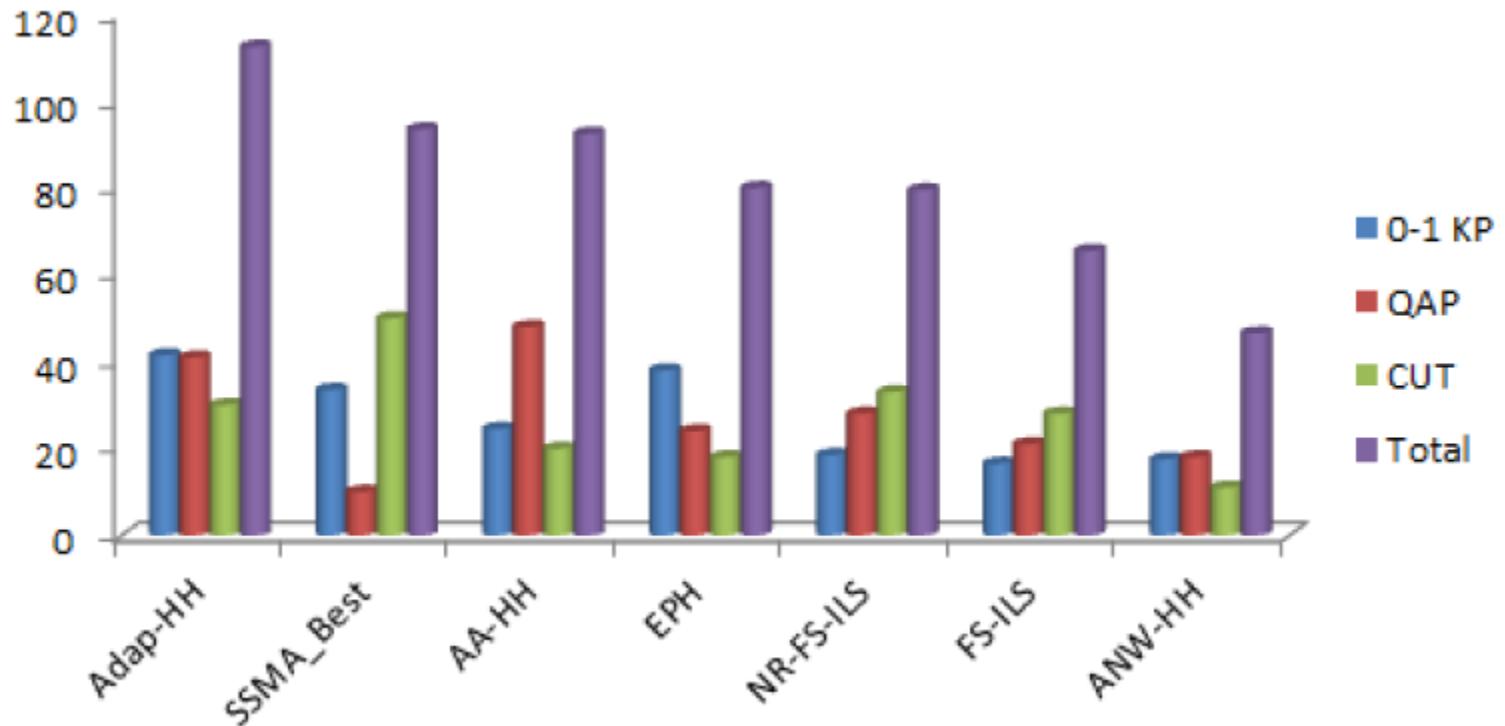


- SSMA-Best selects the best individual as both Parent 1 and 2 for crossover.
- After the application of a chosen crossover, the new offspring will be identical to the parents.
- For the HyFlex domains and instances, the crossover operators do not seem to have much of a positive influence on the overall performance of SSMA.
- Hence, SSMA under the best setting actually performs a single point based search rather than multipoint turning the overall algorithm into an approach similar to Iterated Local Search, where mutation is followed by local search at each step.

Performance of SSMA-Best wrt CHeSC 2011 Competitors



Performance of SSMA-Best on the Extended HyFlex Domains





Conclusions

- Parameter tuning has actual value in cross-domain search.
- The tuning experiments indicate the success of the SSMA-Best, which, significantly, outperforms the two memetic algorithm variants and even some of the competing search methods in CHeSC2011.
- The key observations from the empirical results are that crossover should not be used in this case and that single point based search should be preferred.

Q&A



Thank you.
Ender Özcan

ender.ozcan@nottingham.ac.uk

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
<http://cs.nott.ac.uk/~pszeo>

Hyper-heuristics II

Lecture 8

Ender Özcan



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA

441

Configuring/Tuning of Hyper/Metaheuristics for Cross- domain Search



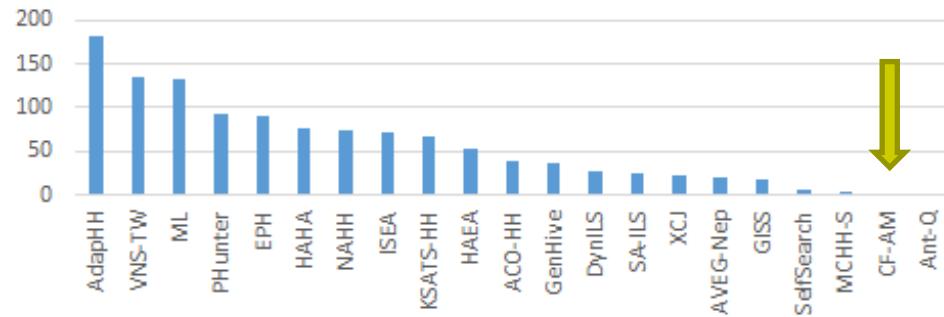
The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA

442

A (Reconfigured) Modified Choice Function Hyper-heuristic

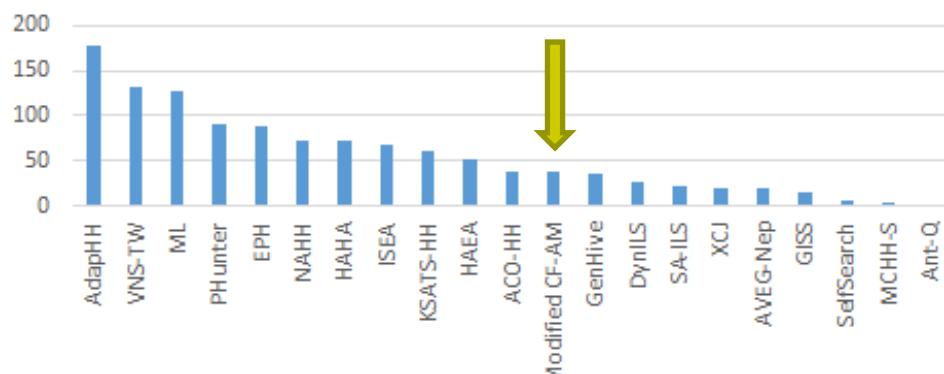
J. H. Drake, E. Özcan and E. K. Burke, A Modified Choice Function Hyper-heuristic Controlling Unary and Binary Operators, Proc. of the IEEE Congress on Evolutionary Computation (CEC), pp. 3389-3396. [[PDF](#)]



[`<code>`](#)
↓

$$F(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j)$$

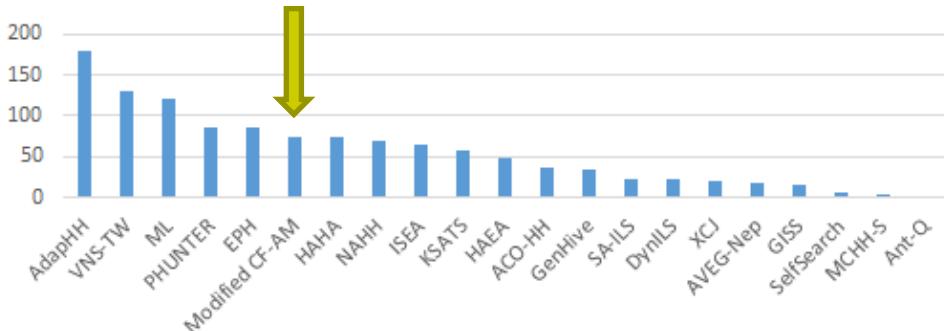
without crossover



$$F_t(h_j) = \phi_t f_1(h_j) + \phi_t f_2(h_k, h_j) + \delta_t f_3(h_j)$$

$$\delta_t = 1 - \phi_t$$

without crossover



$$F_t(h_j) = \phi_t f_1(h_j) + \phi_t f_2(h_k, h_j) + \delta_t f_3(h_j)$$

$$\delta_t = 1 - \phi_t$$

with crossover

A Graph-based Hyper-heuristic

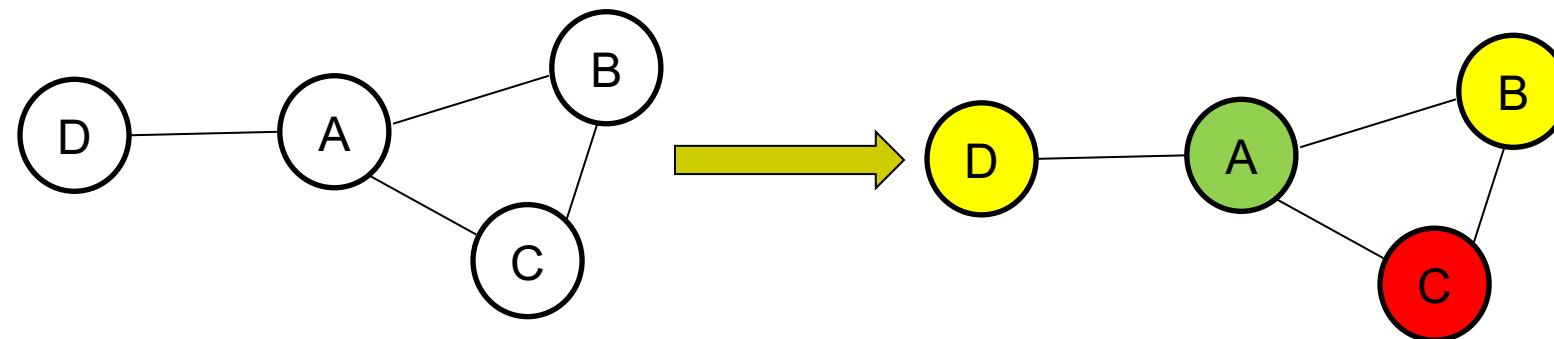
E. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research, 176(1):177-192





Graph Colouring

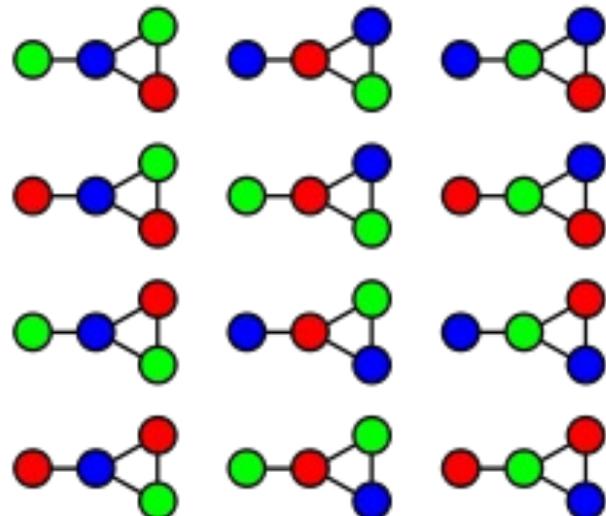
- An assignment of labels traditionally called "colours" to elements of a **graph** subject to certain constraints.
- A way of colouring the vertices of a graph such that no two adjacent vertices share the same colour; this is called a **vertex colouring**.





Graph Colouring

- **k -colouring problem:** Can the vertices of a graph be coloured using k colours so that no two vertices connected by an edge have the same colour?



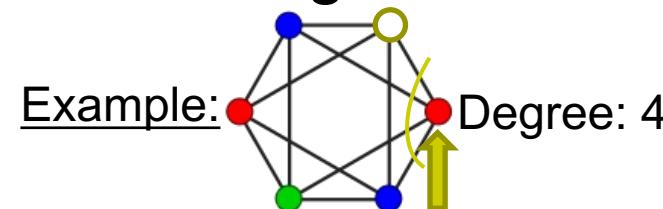
This graph can be 3-colored in 12 different ways.

- **Minimum colouring problem** is an NP-hard problem: colour the vertices of a graph using optimal (minimum) number of colours, so that no two vertices connected by an edge have the same colour.

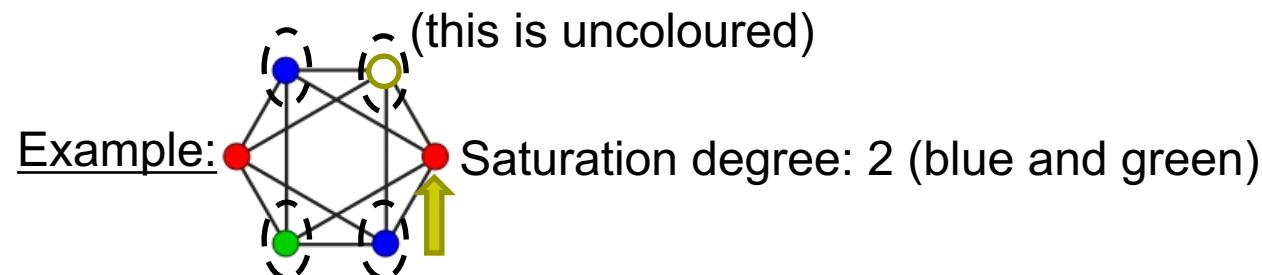


Degree and Saturation Degree of a Vertex

- Degree of a vertex: number of edges connected to that vertex.



- Saturation degree of a vertex: number of differently coloured vertices already connected to it.

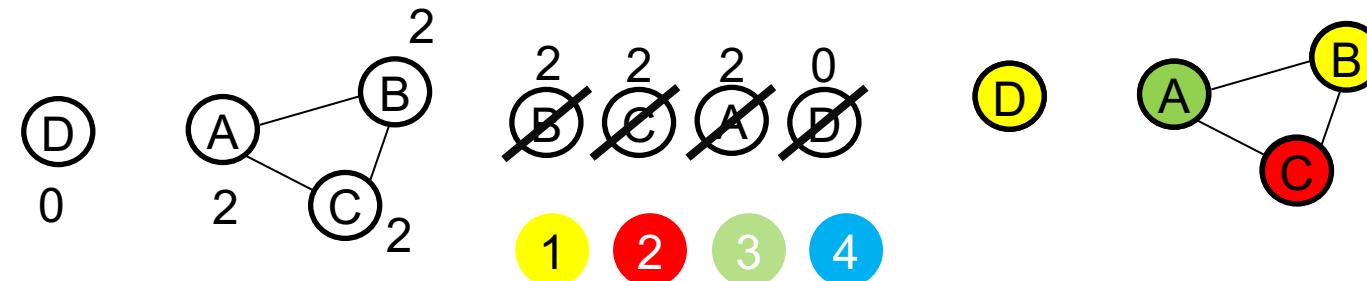


- Can we use saturation degree in a heuristic to construct a solution to a graph colouring problem?



Graph Colouring Heuristics

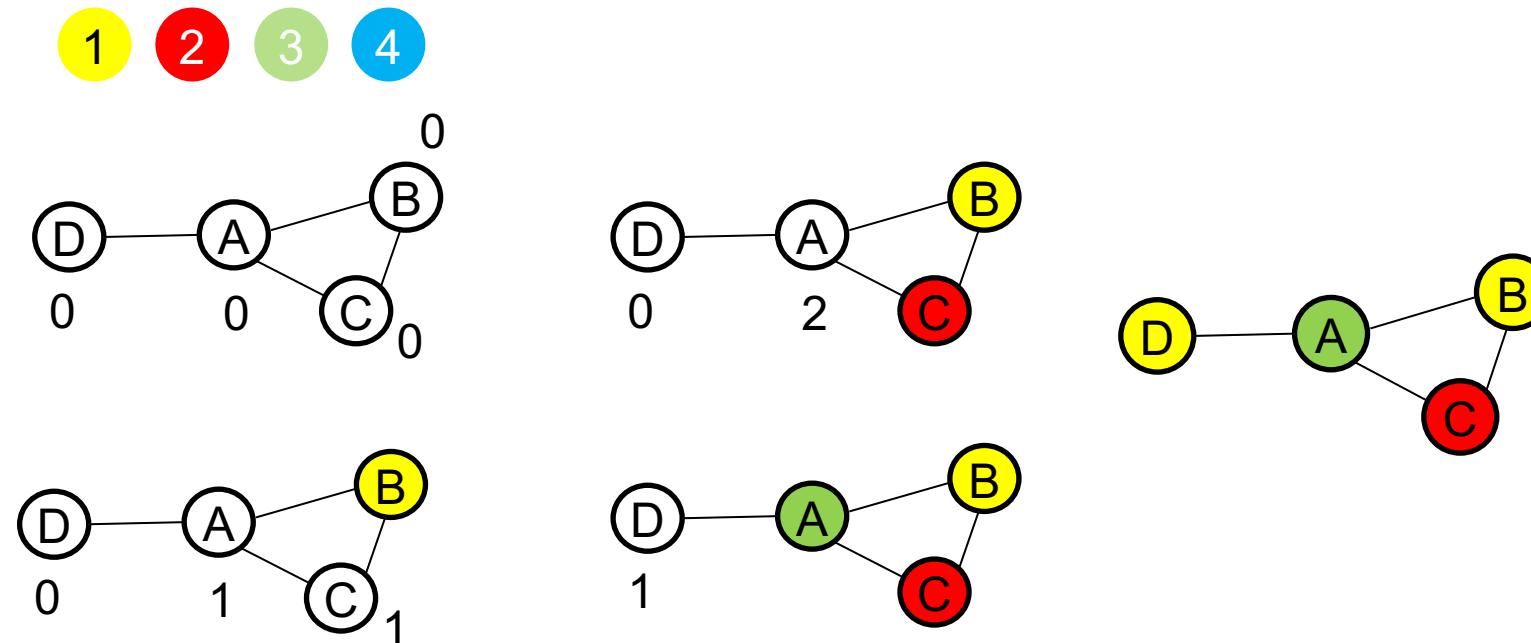
- Largest Degree:
 - ▶ Compute the degree of all vertices
 - ▶ Sort the vertices from **largest degree** to smallest
 - ▶ Colour the first vertex in the list with the next colour (starting with the first) that is different than its neighbours
 - ▶ Delete the vertex from the list go to the previous step unless no vertices left.
- ▶ Example:





Graph Colouring Heuristics

- Saturated Degree:
 - ▶ Use saturation degree at each step in the previous approach
 - ▶ Example:





Examination timetabling

- A number of exams ($e_1, e_2, e_3, \dots, e_E$), taken by different students ($s_1, s_2, s_3, \dots, s_S$), need to be scheduled to a limited time periods ($t_1, t_2, t_3, \dots, t_T$) and certain rooms ($r_1, r_2, r_3, \dots, r_R$)
- Hard Constraints
 - ▶ Exams taken by common students can't be assigned to the same time period
 - ▶ Room capacity can't be exceeded
- Soft Constraints
 - ▶ Separation between exams
 - ▶ Large exams scheduled early

Designing a Local Search Metaheuristic for Examination Timetabling



- Representation: An array of pair of integers, one representing the period assignment and the other representing the room assignment. The array size is the number of events E and each period entry has a value from 1 to T , while room has a value from 1 to R (*integer encoding*)
- Initialisation: randomly assign a period and a room (integer values within the given range) for each event
- Objective function: Number of constraint violations
- Neighbourhood (perturbation) operator:
 - ▶ OP1: Randomly pick an event and reschedule to random period
 - ▶ OP2: Randomly pick an event and assign a different room
 - ▶ OP3: Randomly pick an event and reschedule to random period, also assign a different room
 - ▶ Any of the above can be parametrised, e.g. pick X number of events and apply an operator
 - ▶ More elaborate operators can be designed

Designing Iterated Local Search for Examination Timetabling

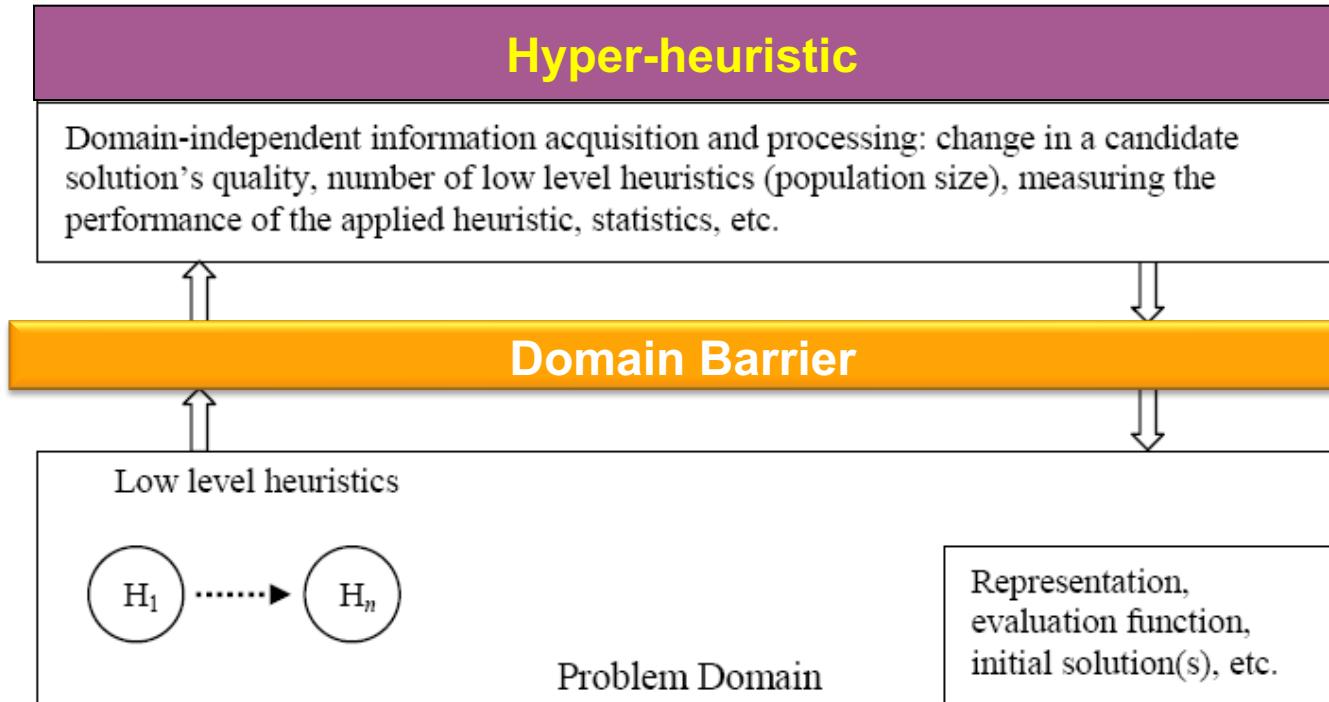


- Local Search: RMHC, DHC, SDHC, NDHC, in which the neighbourhood operator is OP3
- AcceptanceCriterion: accept improving and equal moves (non-worsening): accept a new solution s' if and only if $f(s') \leq f(s^*)$

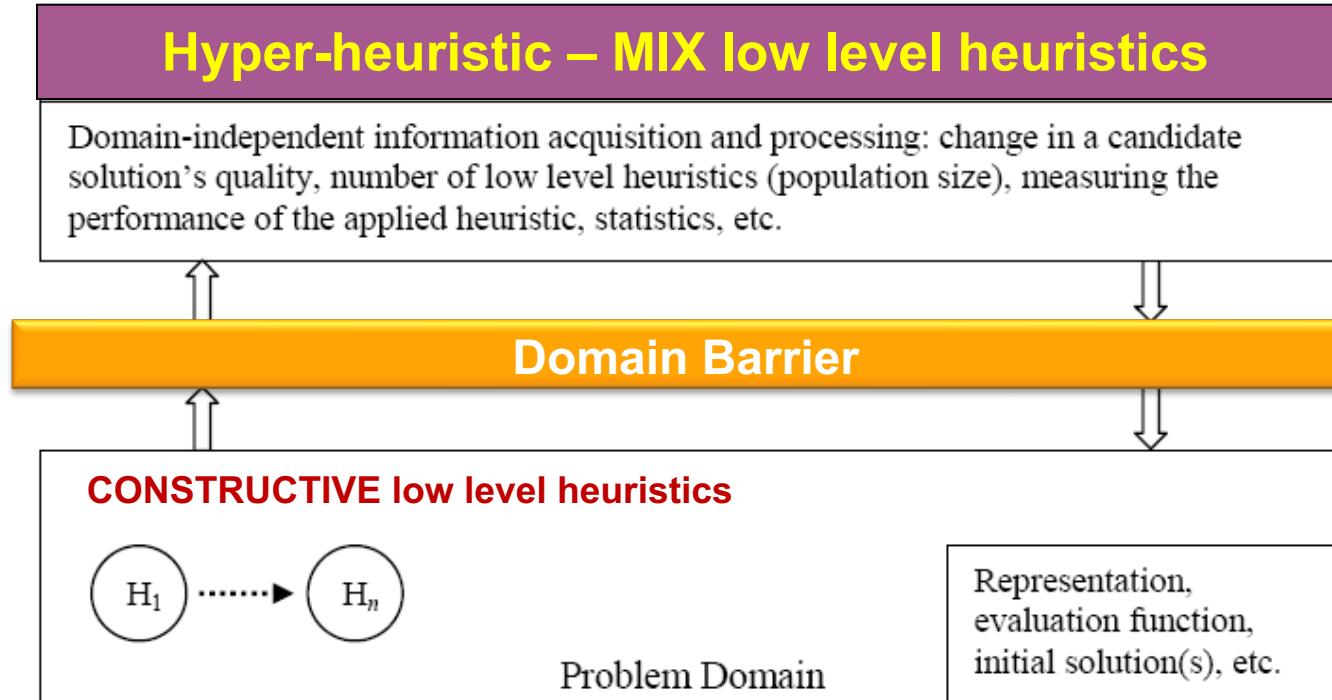
Designing Simulated Annealing for Examination Timetabling

- Initial Temperature: Objective value of the initial solution generated
- Cooling Schedule: Geometric cooling, $\alpha=0.99$
- Termination criteria: Stop and return the best solution found so far when the total number of violation is 0 or maximum number of iterations is exceeded

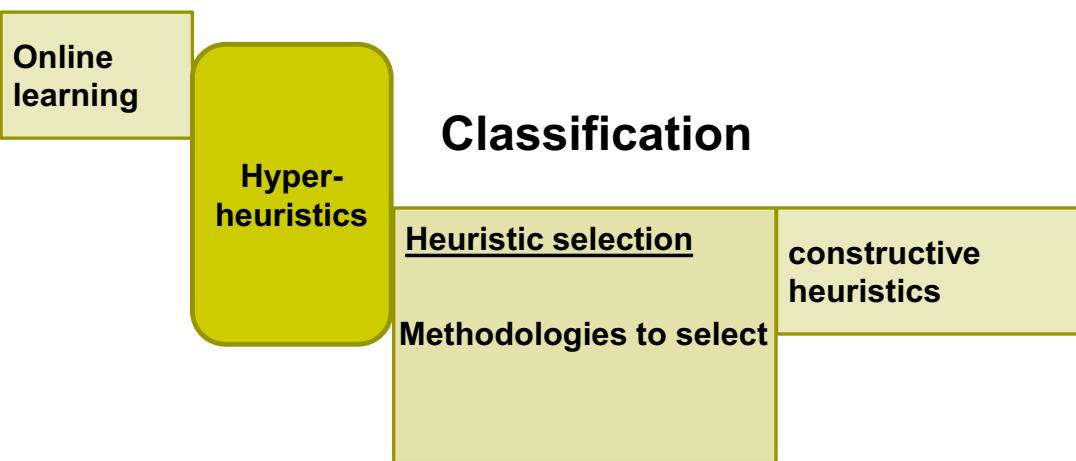
A Hyper-heuristic Framework – revisited



Methodologies to select constructive heuristics



Graph-based hyper-heuristics

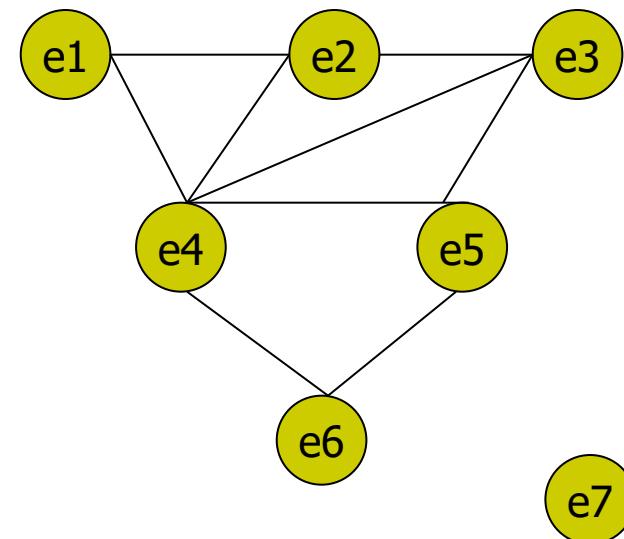


- A general framework (GHH) employing a set of low level constructive graph colouring heuristics
- **Low level heuristics**: sequential methods that order events by the difficulties of assigning them
 - ▶ 5 graph colouring heuristics
 - ▶ Random ordering strategy
- Applied to exam and course timetabling problem



Examination timetabling

- How can we represent/model this problem?
 - ▶ There are 7 exams, $e_1 \sim e_7$
 - ▶ 5 students taking different exams
 - s_1 : e_1, e_2, e_4
 - s_2 : e_2, e_3, e_4
 - s_3 : e_3, e_4, e_5
 - s_4 : e_4, e_5, e_6
 - s_5 : e_7
 - ▶ let's ignore room allocation

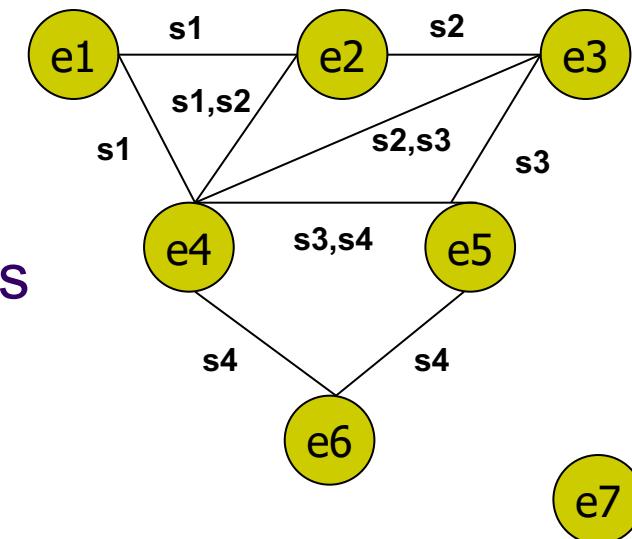




Examination timetabling

Can be modelled as a graph colouring problem

- **Nodes:** exams
- **Edges:** adjacent exams (nodes) have common students
- **Colours:** time periods
- **Objective:** assign colours (time periods) to nodes (exams), adjacent nodes with different colour, minimising time periods used



Pseudo-code of Tabu Search graph based hyper-heuristic



```
initial heuristic list  $hl = \{h_1 \ h_2 \ h_3 \ \dots \ h_k\}$ 
//Begin of Tabu Search
for  $i = 0$  to  $i = (5 * \text{the number of events})$  //number of iterations
     $h = \text{change two heuristics in } hl$  //a move in Tabu Search
    if  $h$  does not match a heuristic list in 'failed list'
        if  $h$  is not in the tabu list // $h$  is not recently visited
            for  $j = 1$  to  $j = k$  // $h$  is used to construct a complete solution
                schedule the first  $N$  events in the event list ordered using  $h_j$ 
            if no feasible solution can be obtained
                store  $h$  into the 'failed list' //update "failed list"
            else if cost of solution  $c <$  the best cost  $c_g$  obtained
                save the best solution,  $c_g = c$  //keep the best solution
                add  $h$  into the tabu list
                remove the first item from the tabu list if its length  $> 9$ 
                 $hl = h$ 
        //end if
        Deepest descent on the complete solution obtained
    //end of Tabu Search
    output the best solution with cost of  $c_g$ 
```

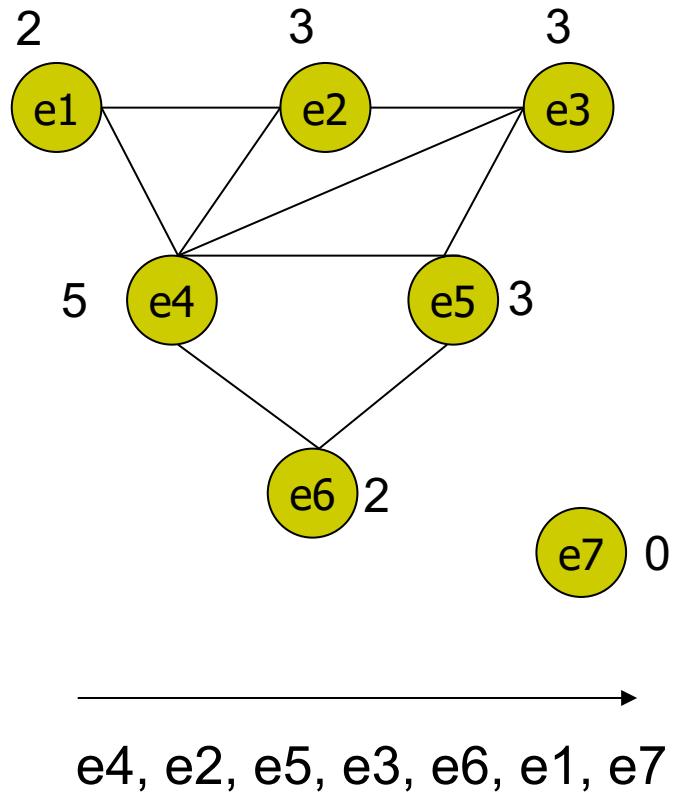
$$\text{number of events} = N * k$$



Graph-based hyper-heuristics

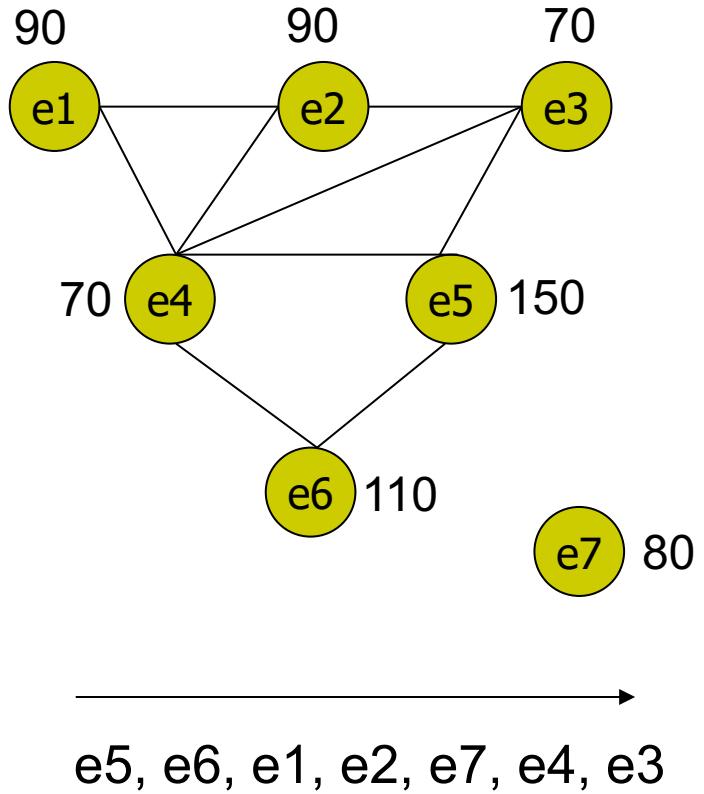
Graph Coloring Heuristics	Ordering strategies
Largest degree (LD)	Number of clashed events
Largest weighted degree (LW)	LD with number of common students
Largest enrolment (LE)	Number of students
Saturation degree (SD)	Number of valid remaining time periods
Colour degree (CD)	Number of clashed events that are already scheduled
+	
Random ordering (RO)	Orders a number events randomly

Example: Largest Degree Graph Colouring/Construction/Ordering Heuristic



Construct a timetable using one event at a time in a given order

Example: Largest Enrolment Graph Colouring/Construction/Ordering Heuristic



Construct a timetable using one event at a time in a given order



Graph-based hyper-heuristics

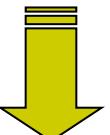
events

e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11	e12	...
----	----	----	----	----	----	----	----	----	-----	-----	-----	-----



heuristic list

SD	SD	LD	CD	LE	SD	SD	LW	SD	LD	CD	RO	...
----	----	----	----	----	----	----	----	----	----	----	----	-----



Schedule N events using a graph
colouring heuristic

order of events

e1	e9	e3	e26	e25	e6	e17	e28	e19	e10	e31	e12	...
----	----	----	-----	-----	----	-----	-----	-----	-----	-----	-----	-----

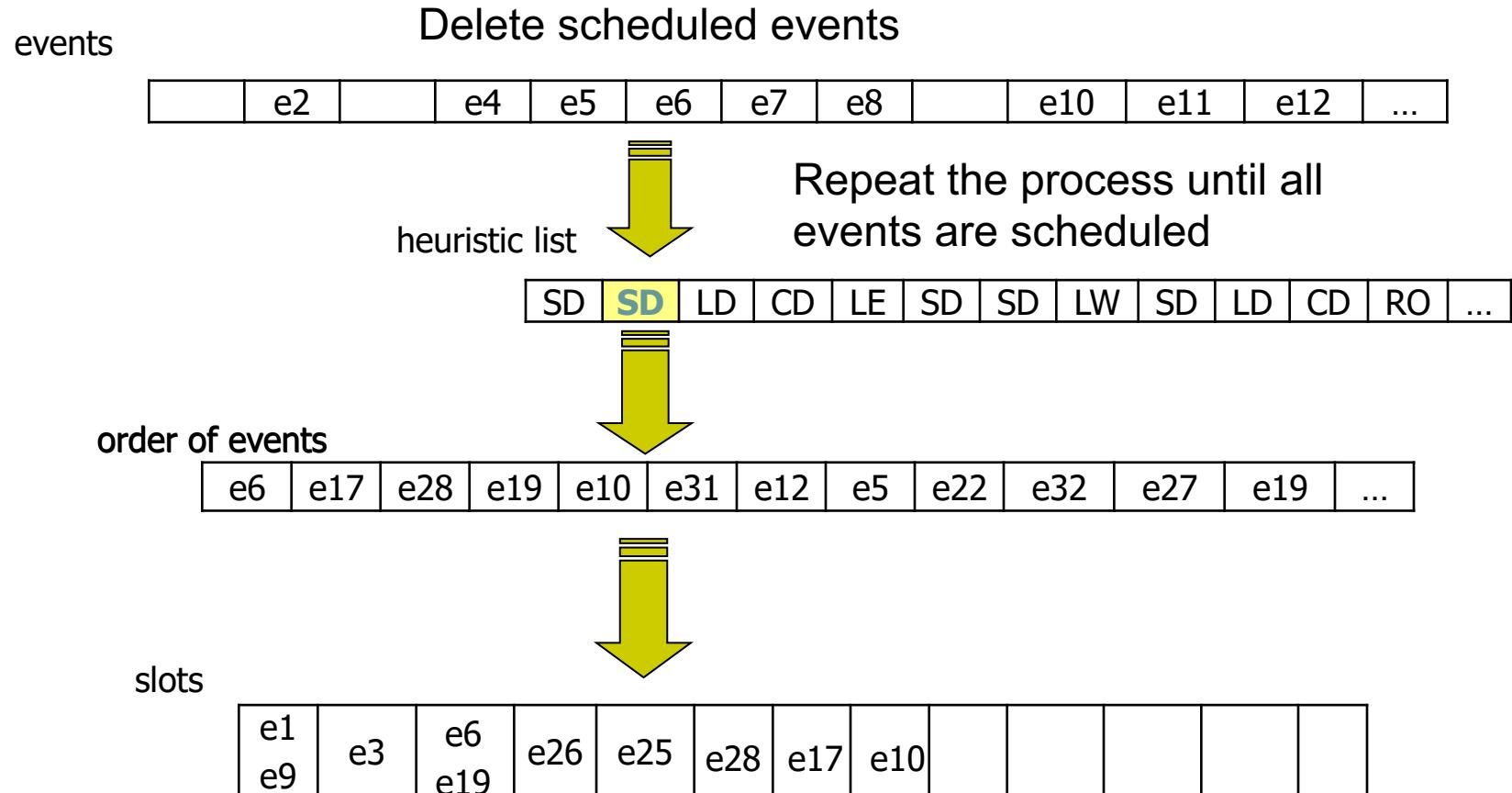
slots

e1	e3		e26	e25							
e9											

$N=5$



Graph-based hyper-heuristics





Graph-based hyper-heuristics

events

	e2		e4	e5		e7	e8			e11	e12	...
--	----	--	----	----	--	----	----	--	--	-----	-----	-----

heuristic list

SD	SD	LD	CD	LE	SD	SD	LW	SD	LD	CD	RO	...
----	----	----	----	----	----	----	----	----	----	----	----	-----

order of events

e5	e32	e19	e22	e13	e31	e12	e7	e2	e15	e27	e12	...
----	-----	-----	-----	-----	-----	-----	----	----	-----	-----	-----	-----

slots

e1 e9	e3	e6 e19	e26	e25	e28	e17	e10 e13	e5 e13	e32 e19	e13		
----------	----	-----------	-----	-----	-----	-----	------------	-----------	------------	-----	--	--



Graph-based hyper-heuristics

- Tabu Search at the high level
 - ▶ Neighbourhood operator: randomly change two heuristics in the heuristic list
 - ▶ Objective function: quality of solutions built by the corresponding heuristic list
 - ▶ Tabu list: visits to the same heuristic lists forbidden
- Other high-level search strategies tested
 - ▶ Steepest Descent
 - ▶ Variable neighbourhood search → best performing
 - ▶ Iterated Steepest Descent



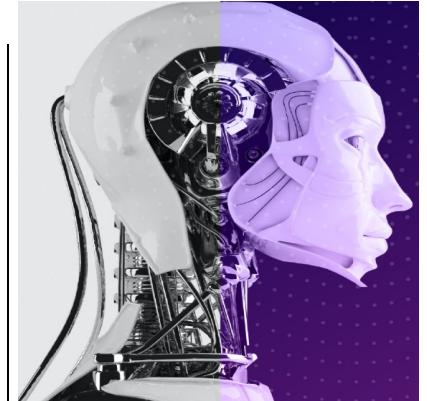
Results

Carter's benchmark

Problem	SL	SLR	SCL	SCLR	SCLx	SCLxR	SCLxR(10*)
car91	5.78	5.67	5.52	5.36	5.65	5.43	5.39
car92	4.76	4.68	4.21	4.14	4.53	4.78	4.63
ear83	38.8	38.57	38.68	38.5	37.92	38.22	38.03
hec92	12.35	12.27	12.30	12.81	12.39	12.25	12.11
kfu93	16.21	15.79	15.37	15.23	15.67	15.2	15.12
lse91	12.17	11.36	12.09	11.93	11.56	11.33	11.33
sta83	164.01	163.5	164.06	163.31	158.19	160.19	159.32
tre92	9.15	9.13	8.87	9.08	8.75	9.03	8.97
ute92	29.49	29.17	28.27	28.19	28.01	28.21	28.11
uta93	4.12	4.03	4.05	3.98	3.88	3.95	3.78
york83	44.54	42.67	42.44	42.37	41.37	42.01	41.52

Costs of solutions obtained by GHH upon a different number of heuristics (S: saturation degree, L: largest degree, C: color degree; R: random ordering, Lx: largest weighted)

Generation Hyper-heuristics



Computational
Optimisation &
Learning Lab



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA



Genetic Programming (GP)

- Challenge:
“Get a computer to do what needs to be done, without telling it how to do it.”
- GP provides a method for automatically creating a working computer program from a high-level problem statement of the problem (i.e., program synthesis or program induction)
- GP iteratively transforms a population of computer programs into a new generation of programs via evolutionary process



Why Genetic Programming?

- “It is difficult, unnatural, and overly restrictive to attempt to represent hierarchies of dynamically varying size and shape with fixed length character strings.” “For many problems in machine learning and artificial intelligence, the most natural representation for a solution is a computer program.” [Koza, 1994]
- A parse tree is a good representation of a computer program for Genetic Programming

Some selected real-world applications of Genetic Programming



- Automated design of mechatronic systems (NSF)
- Climatology: Estimation of heat flux between the atmosphere and sea ice, modelling global temperature changes
- Clinical decision support in ophthalmology
- Container loading optimisation
- Scheduling, timetabling
- Machine learning
- Vehicle routing ...



A Computer Program in C

```
int foo (int time)
{
    int temp1, temp2;
    if (time > 10)
        temp1 = 3;
    else
        temp1 = 4;
    temp2 = temp1 + 1 + 2;
    return (temp2);
}
```

What would be the output of this code as time changes from 0 to 14?

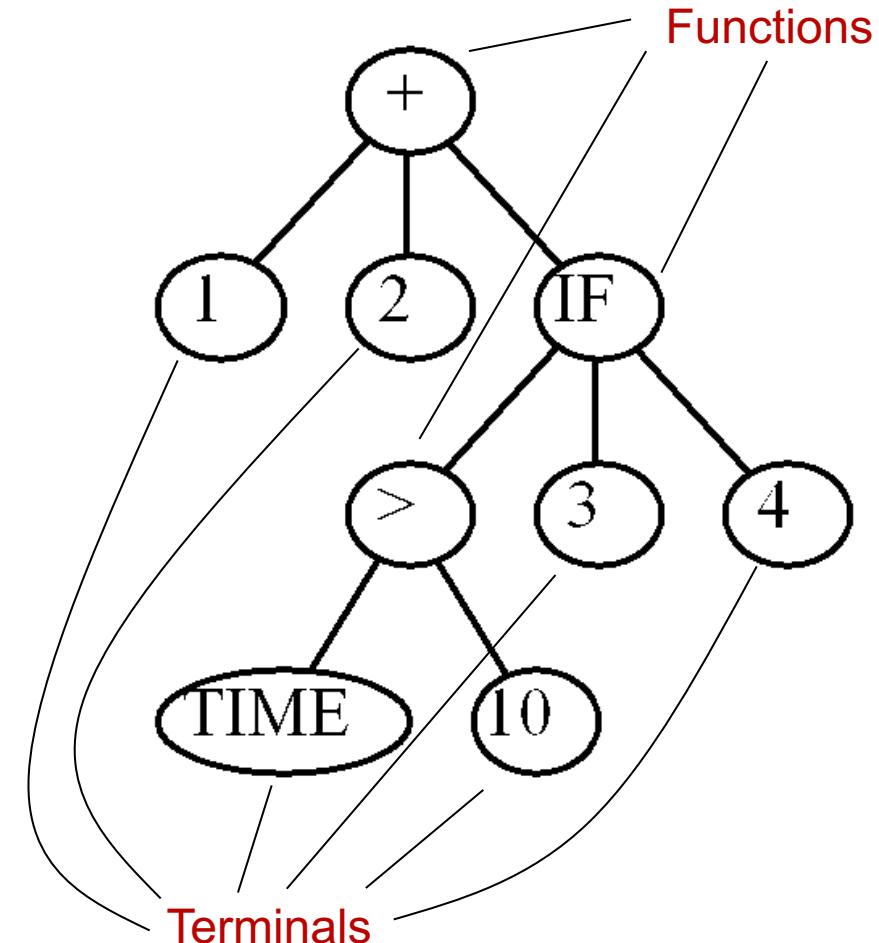


A Computer Program in C

- If we have the observations on the right, can we reverse engineer the mapping function?

Time	Output
0	7
1	7
2	7
3	7
4	7
5	7
6	7
7	7
8	7
9	7
10	7
11	6
12	6
13	6
14	6

Using Trees To Represent Computer Programs



```
(+ 1 2 (IF (> TIME 10) 3 4))
```



Genetic Operations

- GP is an evolutionary algorithm containing the same algorithmic components, including:
 - ▶ Random generation of the initial population of possible solutions (programs)
 - ▶ Genetic crossover of two promising solutions to create new possible solutions (programs)
 - ▶ Mutation of promising solutions to create new possible solutions (programs)



Randomly Generating Programs

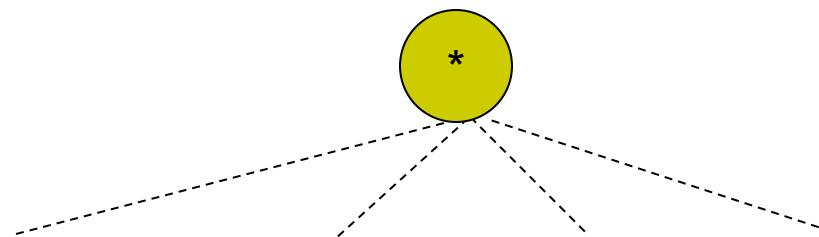
- Randomly generate a program that takes two (or more) arguments and uses basic arithmetic to return an answer
 - ▶ Function set = {+, -, *, /}
 - ▶ Terminal set = {integers, X, Y}
- Randomly select either a function or a terminal to represent our program
- If a function was selected, recursively generate random programs to act as arguments



Randomly Generating Programs

Random item: *

(* ...) – assume multiplication with four arguments: arg1 + arg2 + arg3 + arg4
randomly create each argument one by one

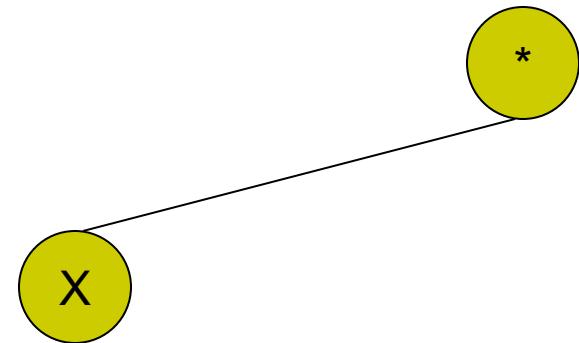




Randomly Generating Programs

arg1 is a terminal, so stop
(* X ...)

Random item: X

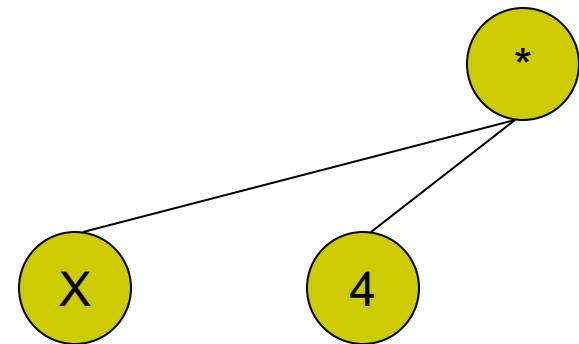




Randomly Generating Programs

arg2 is terminal stop
(* X 4 ...)

Random item: 4



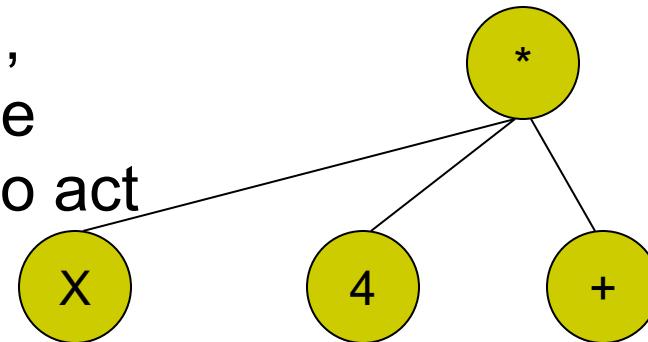


Randomly Generating Programs

arg3 is a function
(* X 4 (+ ...) ...)

assume addition requires
two arguments, since a
function is selected,
recursively generate
random programs to act
as arguments

Random item: +



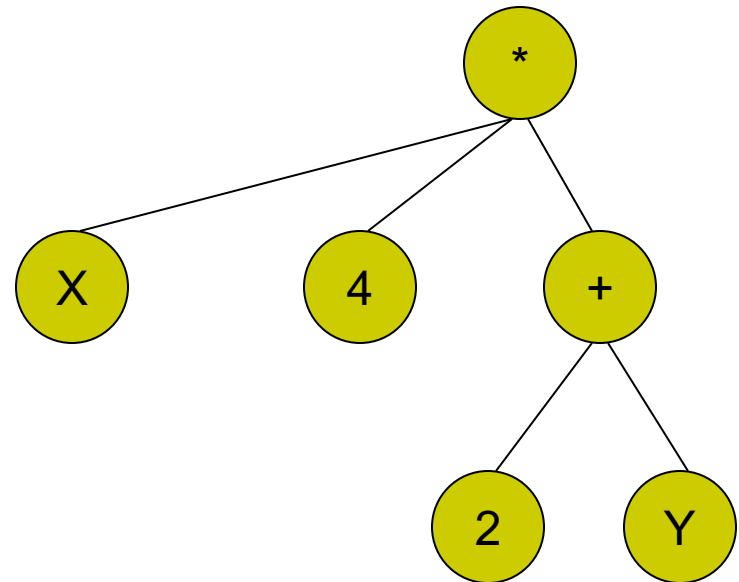


Randomly Generating Programs

(* X 4 (+ 2 Y) ...)

arguments of addition are terminals, so stop recursion for each

Random items: 2, Y





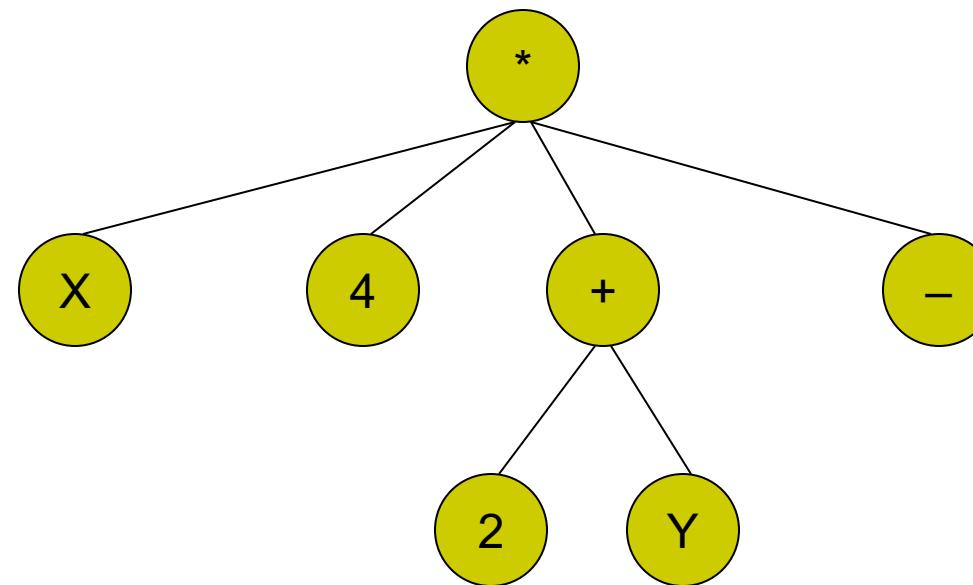
Randomly Generating Programs

arg3 is a function

(* X 4 (+ 2 Y) (- ...))

assume subtraction requires two arguments, since a function is selected,
recursively generate random programs to act as arguments

Random item: -



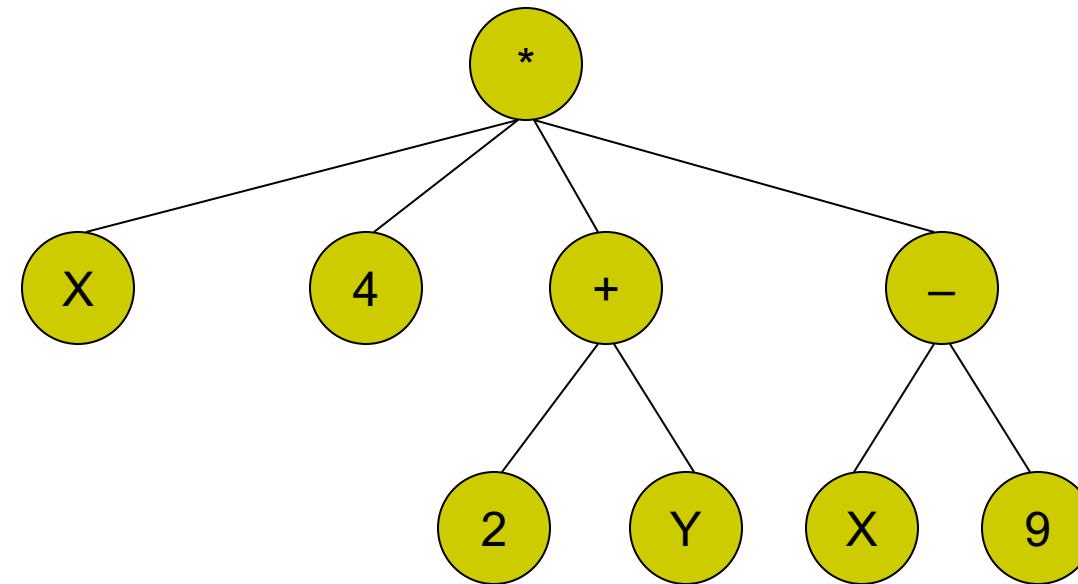


Randomly Generating Programs

(* X 4 (+ 2 Y) (- X 9))

arguments of subtraction are terminals, so stop recursion for each

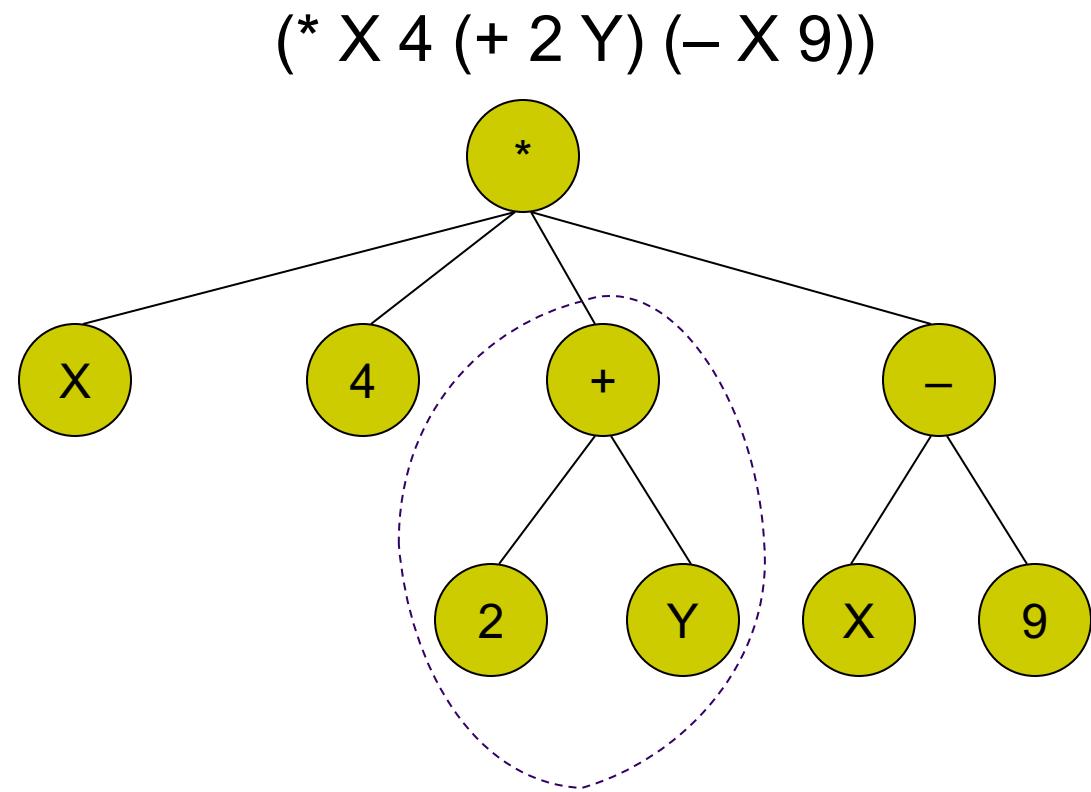
Random items: X, 9



Mutation



First pick a random node (e.g., node(+))

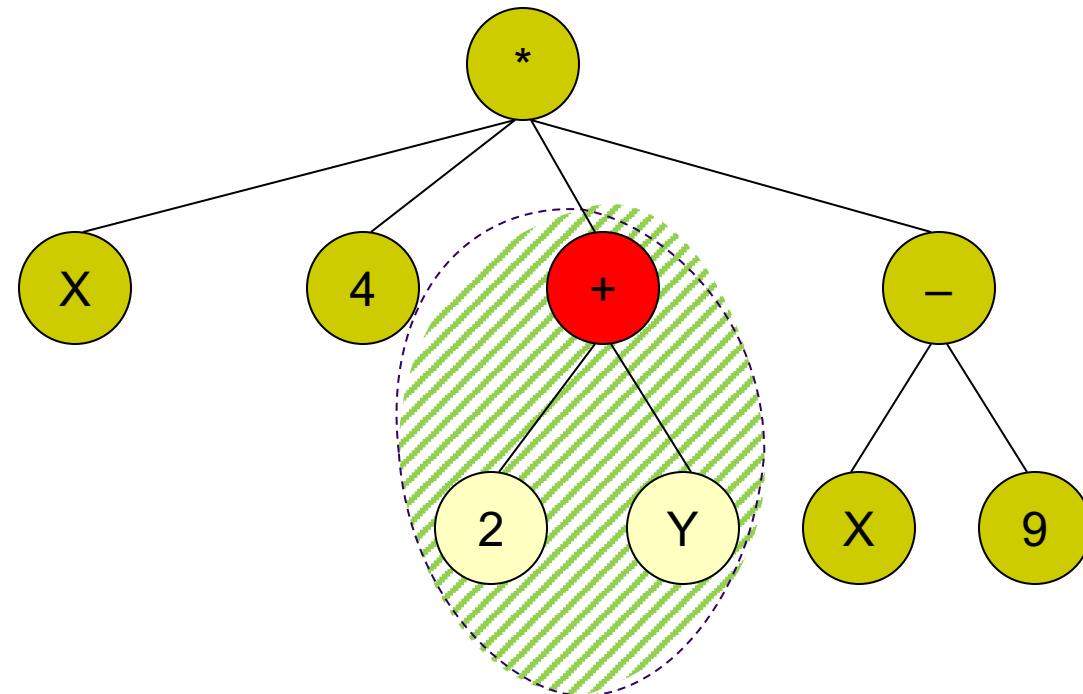


Mutation



First pick a random node (e.g., $\text{node}(+)$),
Delete that node and its children

(* X 4 (+ 2 Y) (- X 9))

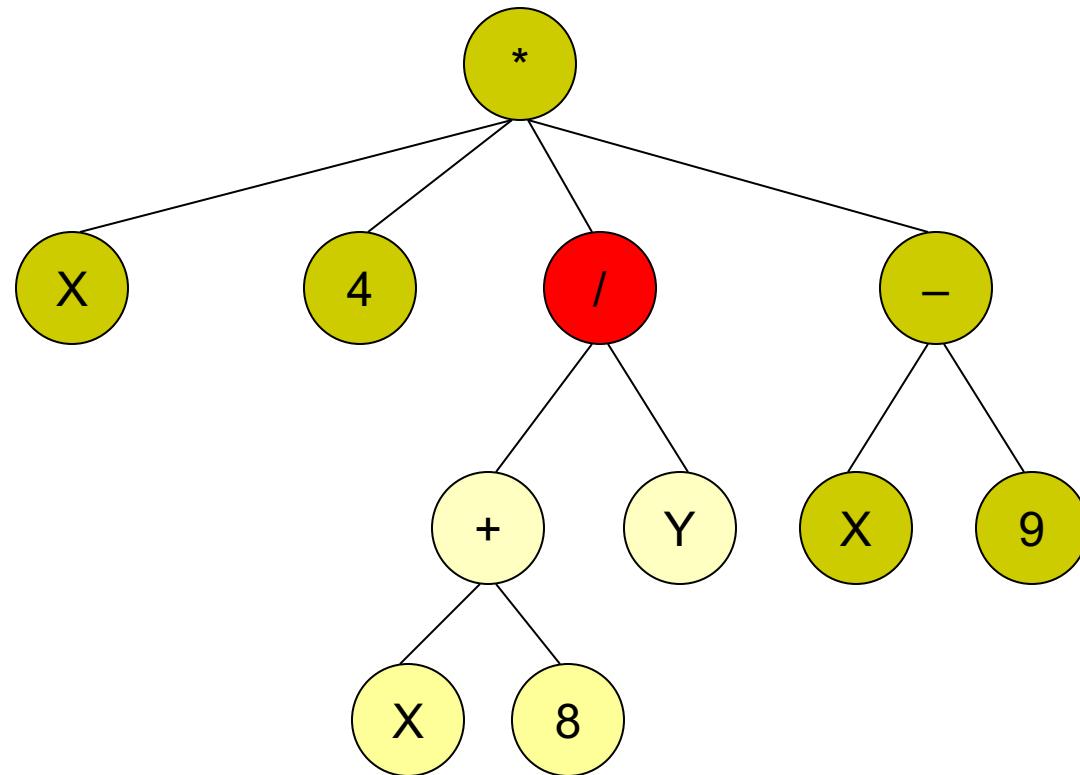


Mutation



Insert a randomly generated program in place of the deleted node (e.g., $(\backslash (+ X 8) Y)$)

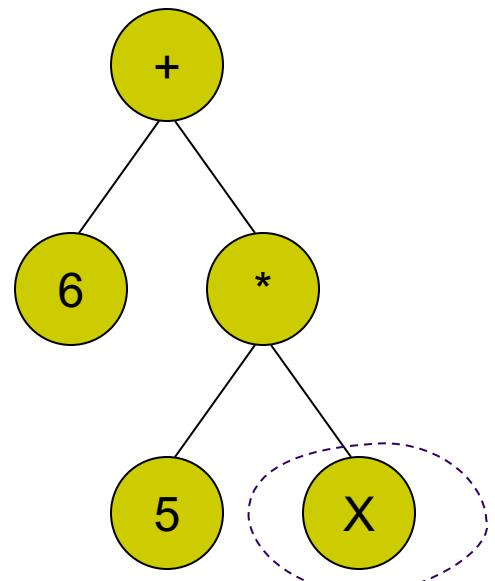
$(* X 4 (\backslash (+ X 8) Y) (- X 9))$





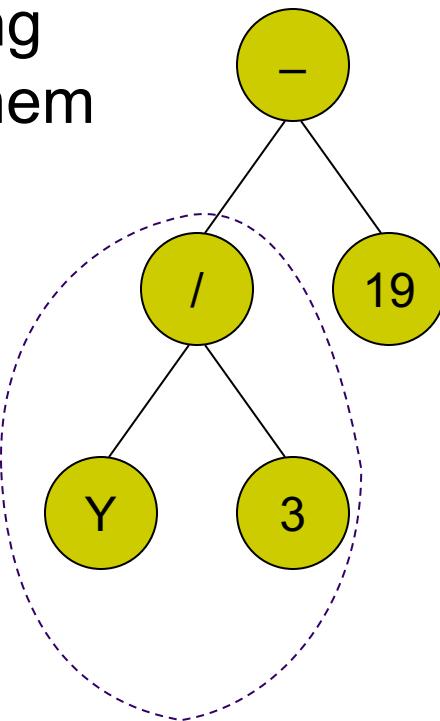
Crossover

(+ 6 (* 5 X))



Pick a random node in each program including the subtrees having them as the root nodes
(e.g., X and /)

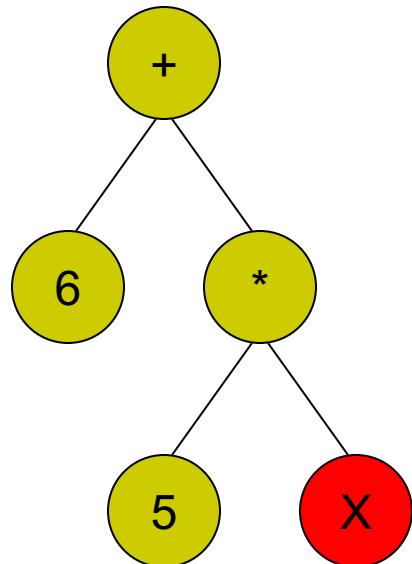
(- (/ Y 3) 19)





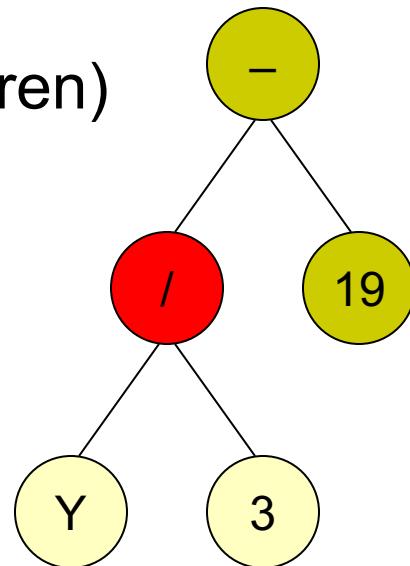
Crossover

(+ 6 (* 5 X))



(- (/ Y 3) 19)

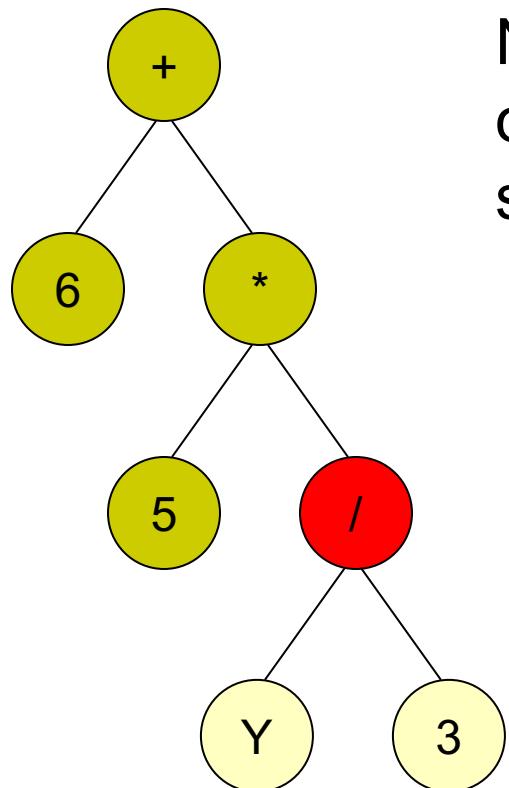
Swap the two nodes
(including their all children)





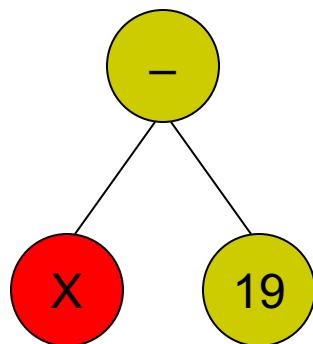
Crossover

(+ 6 (* 5 (/ Y 3)))



Now we have two offspring/new candidate solutions

(- X 19)

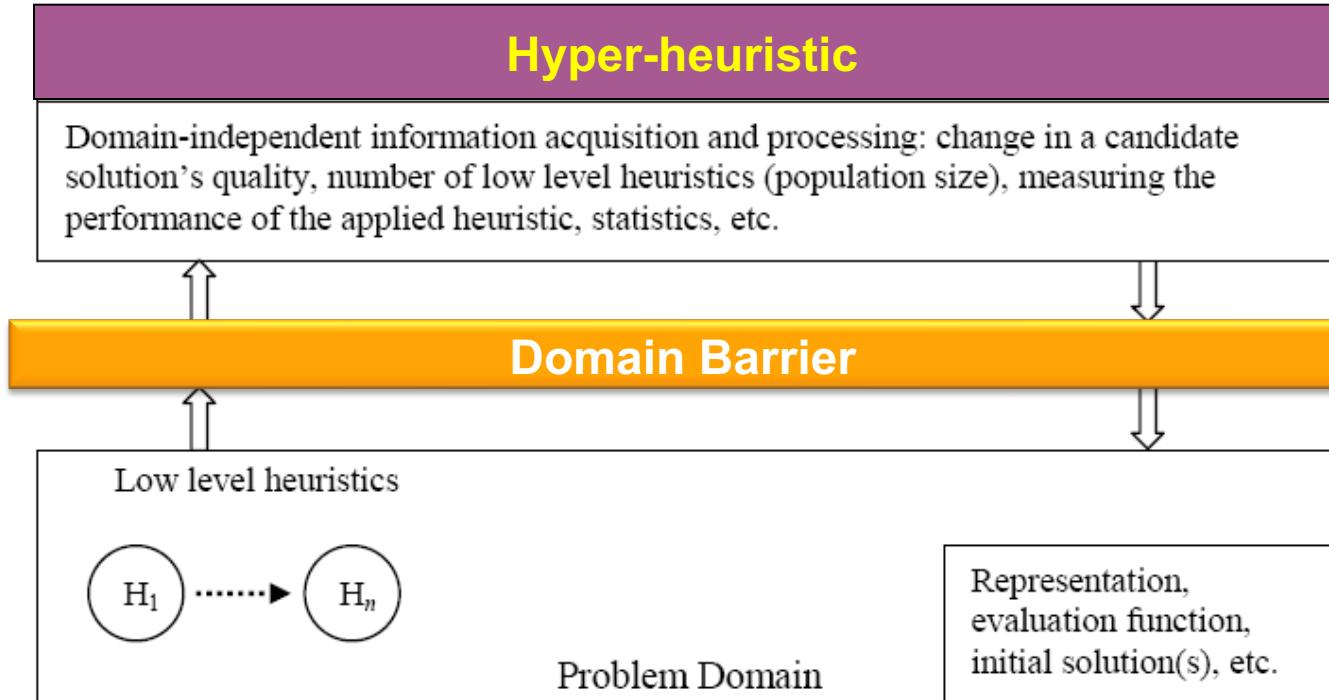


Some Java based Software Libraries



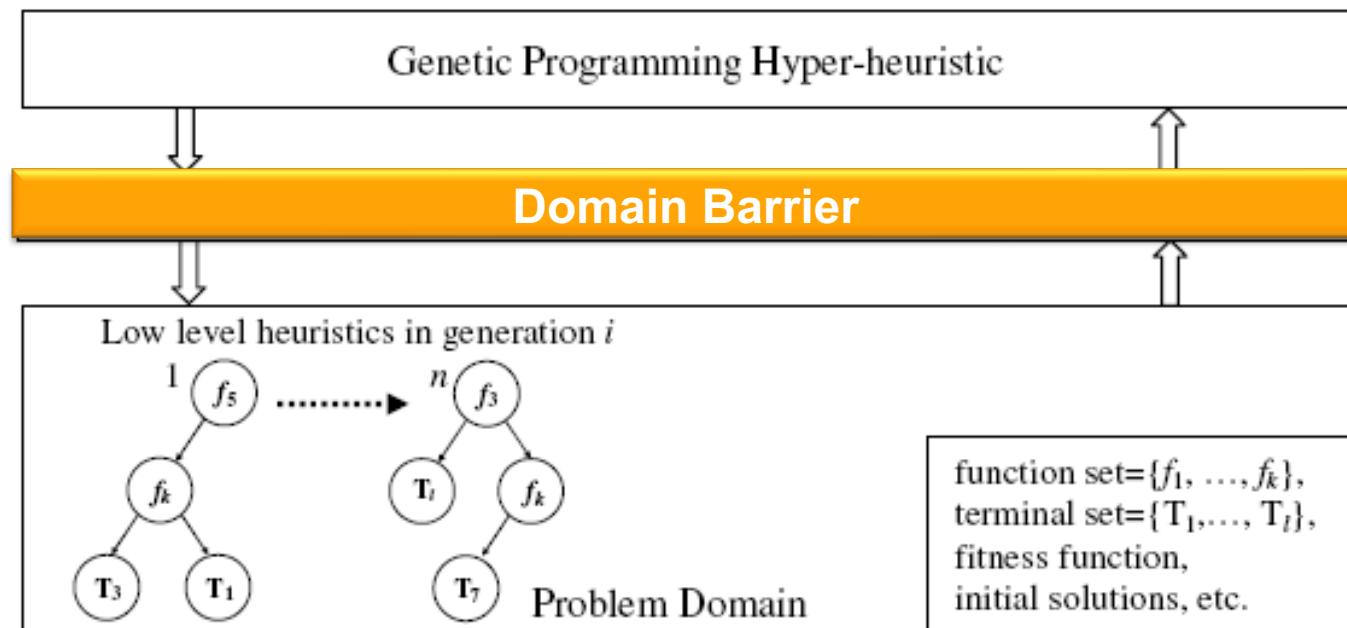
- ECJ: <http://cs.gmu.edu/~eclab/projects/ecj/>
- TinyGP: <http://cswww.essex.ac.uk/staff/rpoli/TinyGP/>
- GEVA (grammatical evolution):
<http://ncra.ucd.ie/Site/GEVA.html>
- Cartesian GP resources:
<http://www.cartesiangp.co.uk/resources.html>

A Hyper-heuristic Framework – revisited



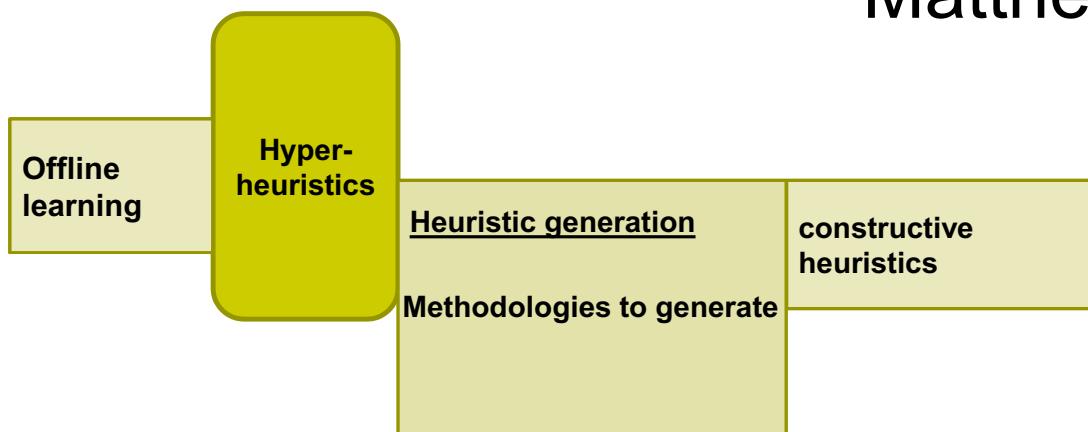


A Generation Hyper-heuristic Framework



Genetic Programming for Packing

from the PhD Thesis (2010) of
Matthew Hyde



Computational
Optimisation &
Learning Lab



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA





1D Offline Bin Packing

Pack a **set** of items of sizes s_i for $i = 1, \dots, n$

- ▶ Sizes are integer values and $s_i \in [1, C]$
- ▶ C is the fixed capacity of each bin

in such a way that

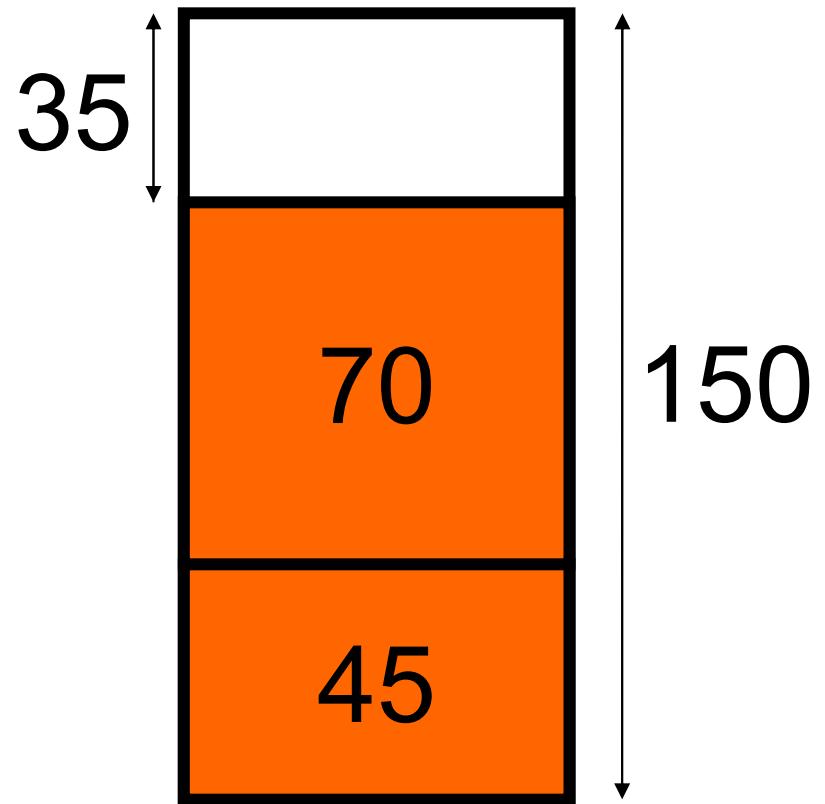
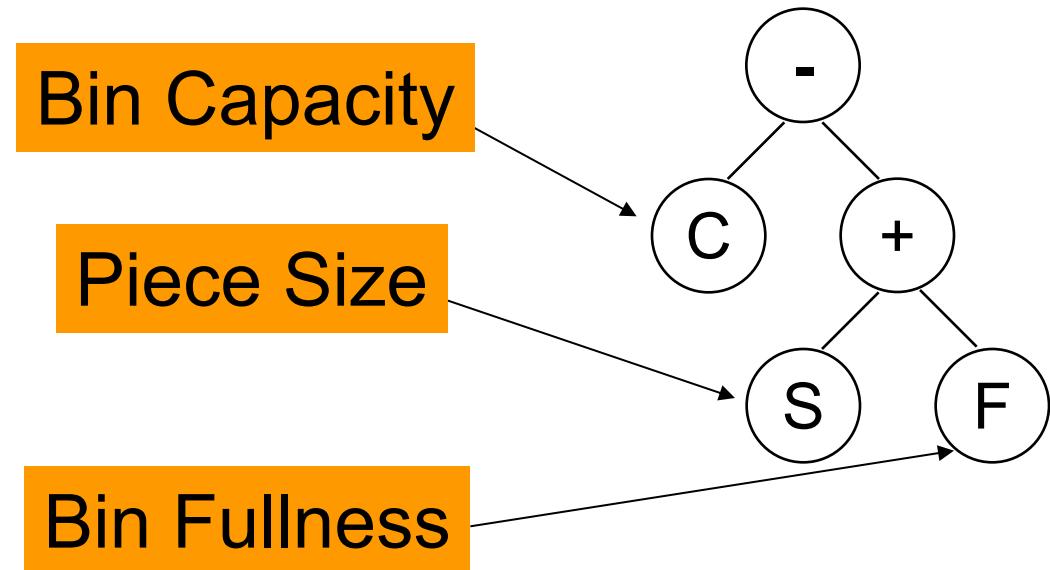
- ▶ Never exceed bin capacity
- ▶ Minimise number of bins used

Standard NP-hard problem

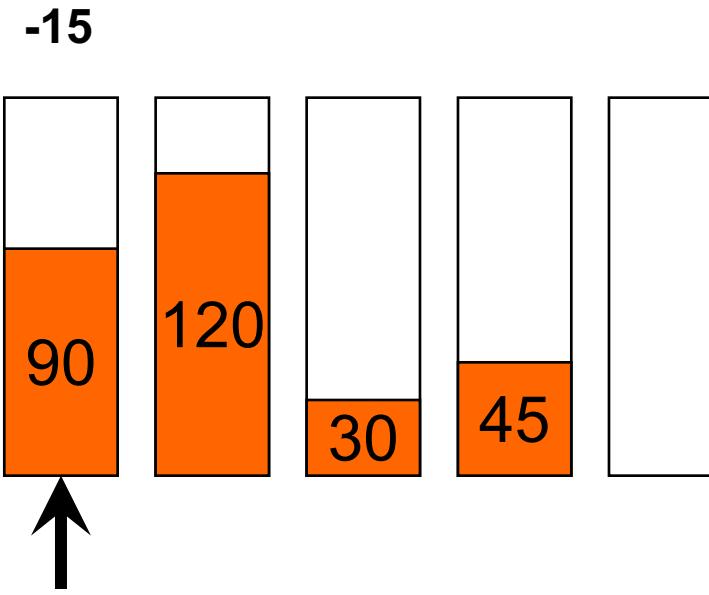
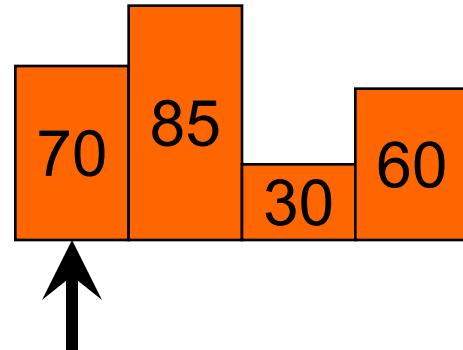
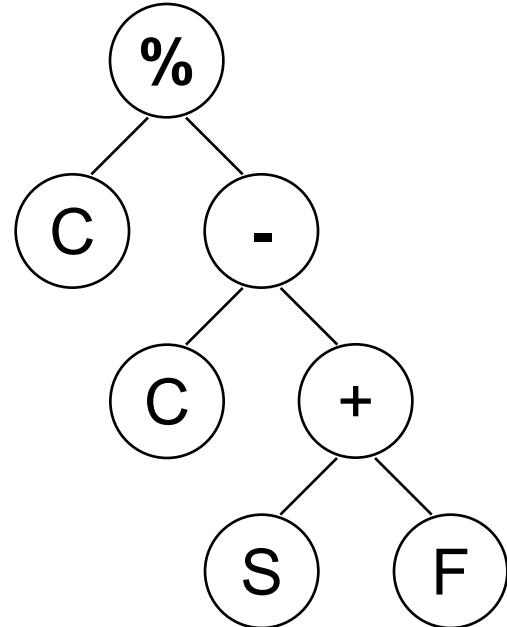
Genetic Programming



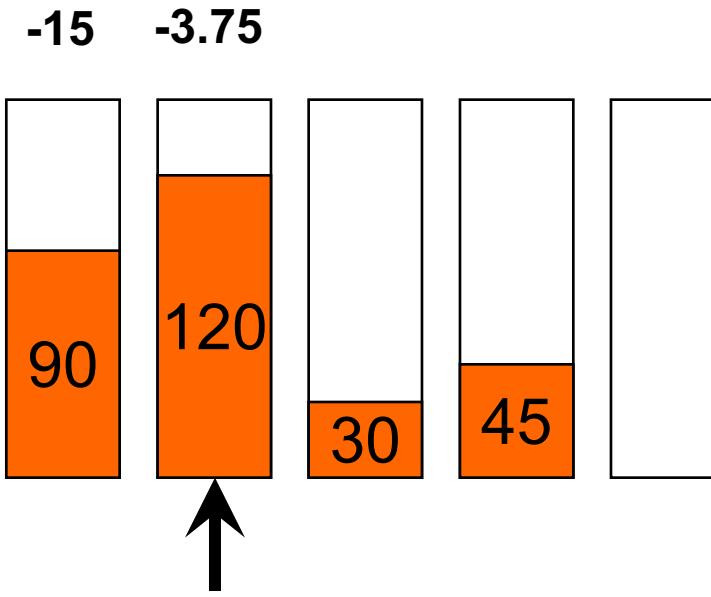
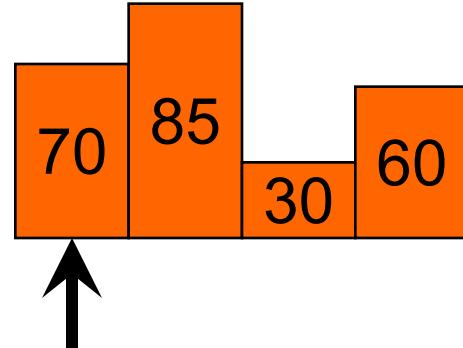
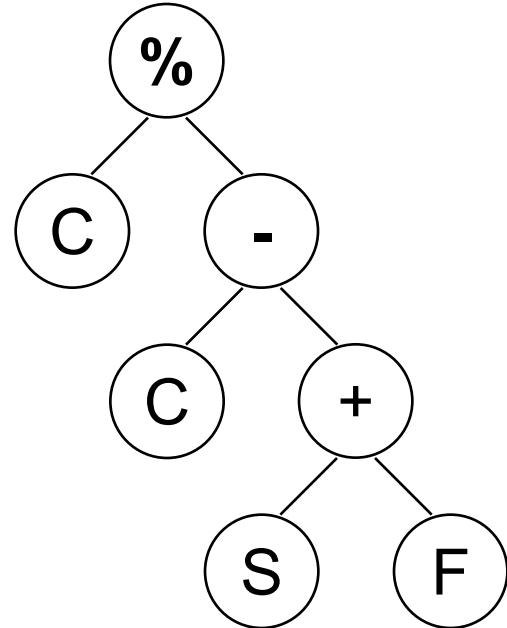
- Evolves trees representing a program
- Following tree is a program that calculates the space left at the top of the bin
- Train and test



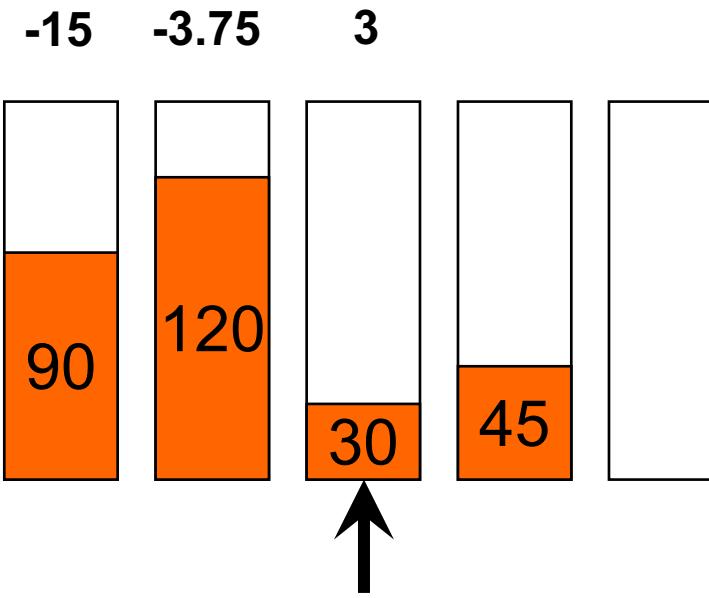
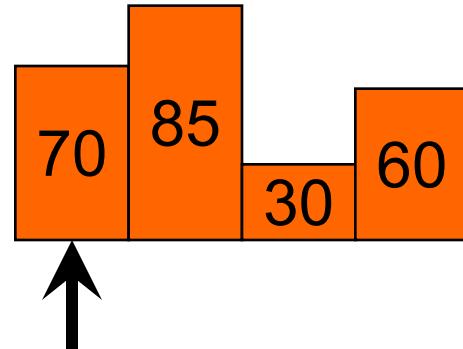
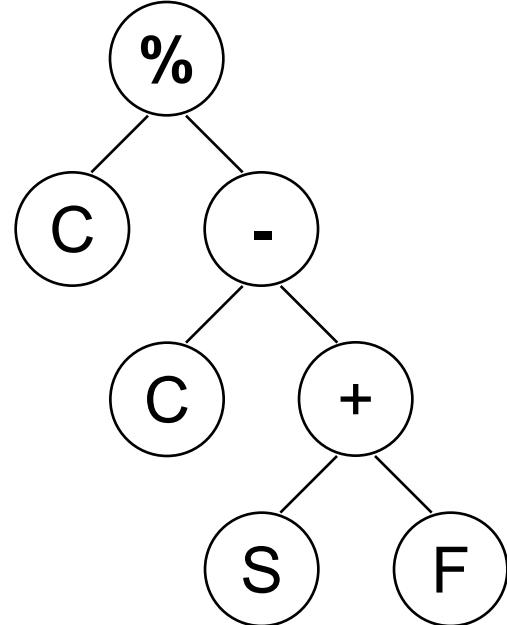
Genetic Programming Heuristics – Bin Packing



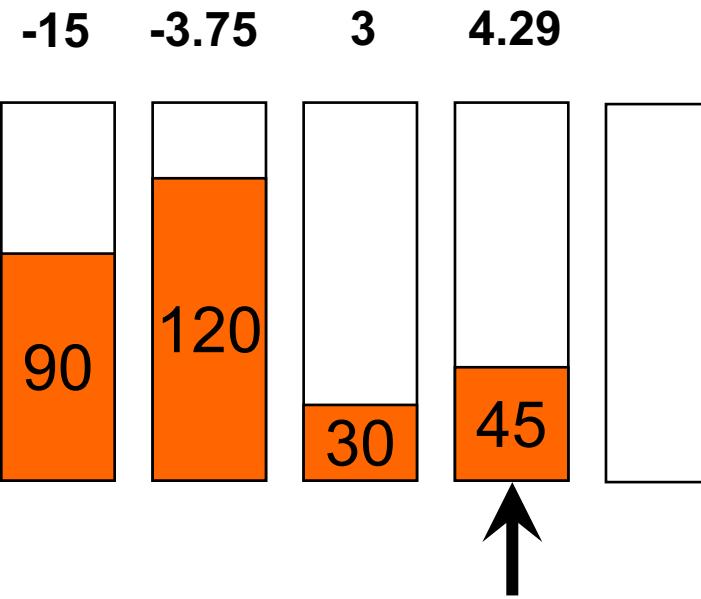
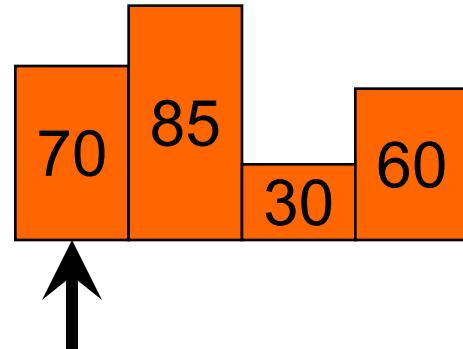
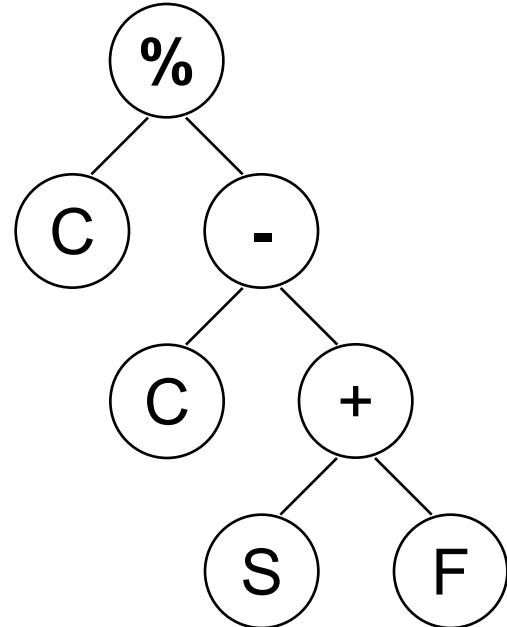
Genetic Programming Heuristics – Bin Packing



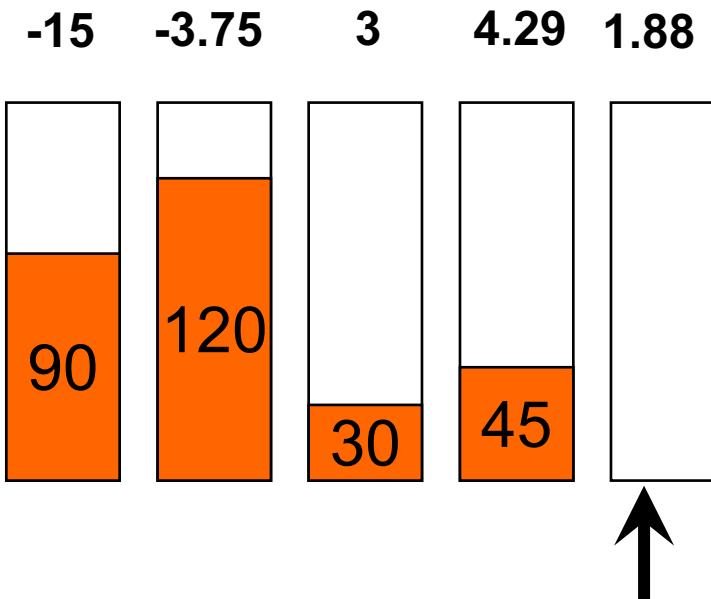
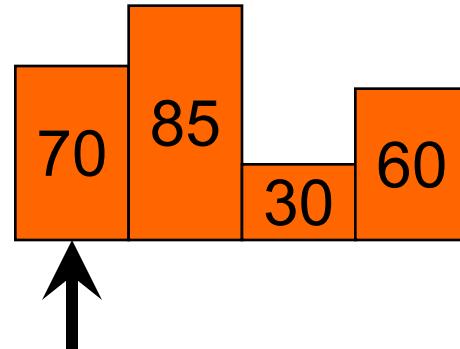
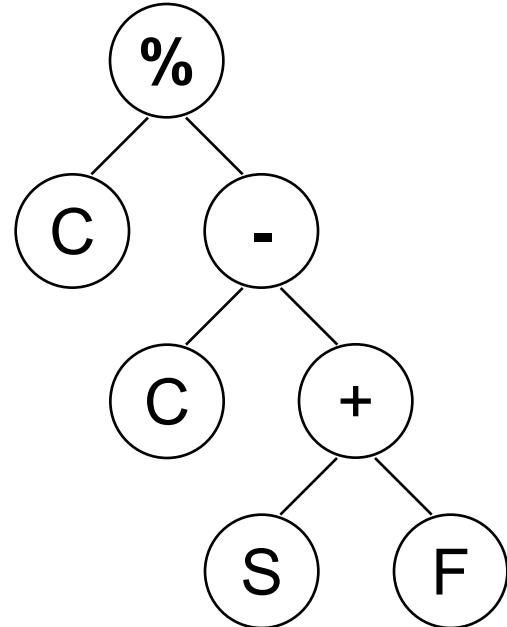
Genetic Programming Heuristics – Bin Packing



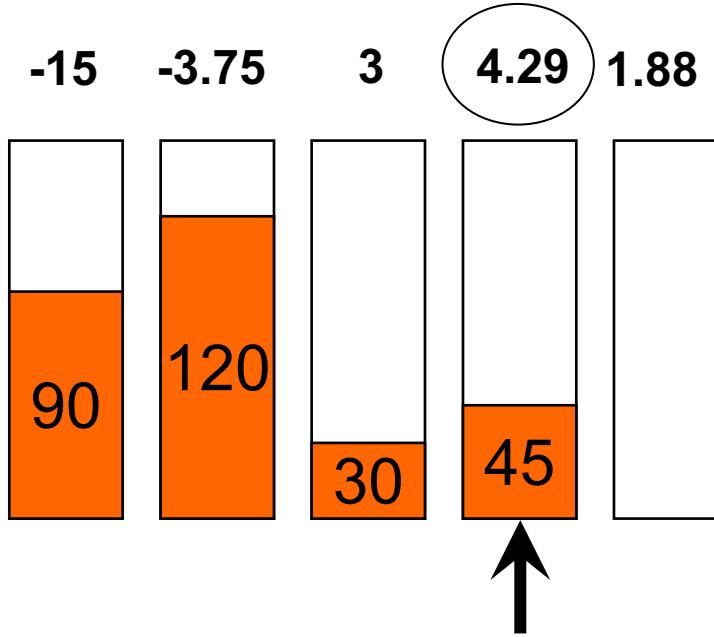
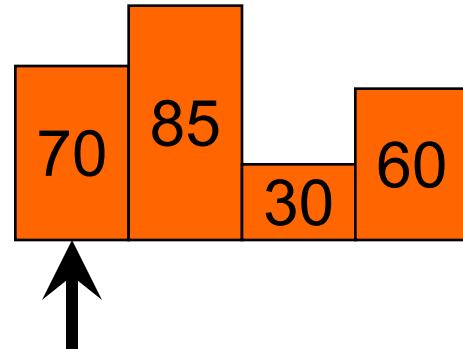
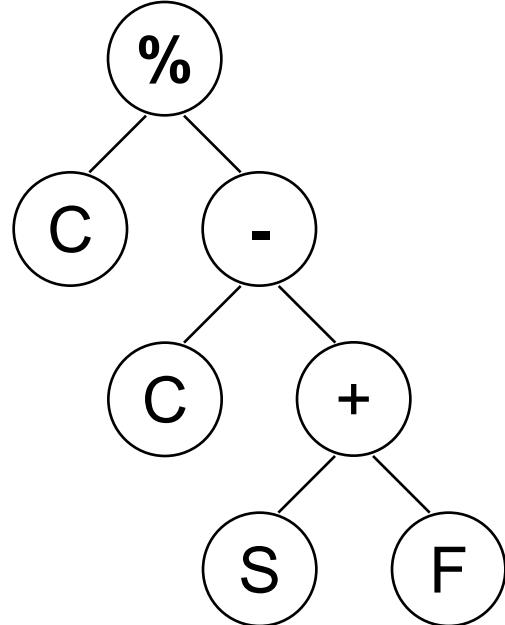
Genetic Programming Heuristics – Bin Packing



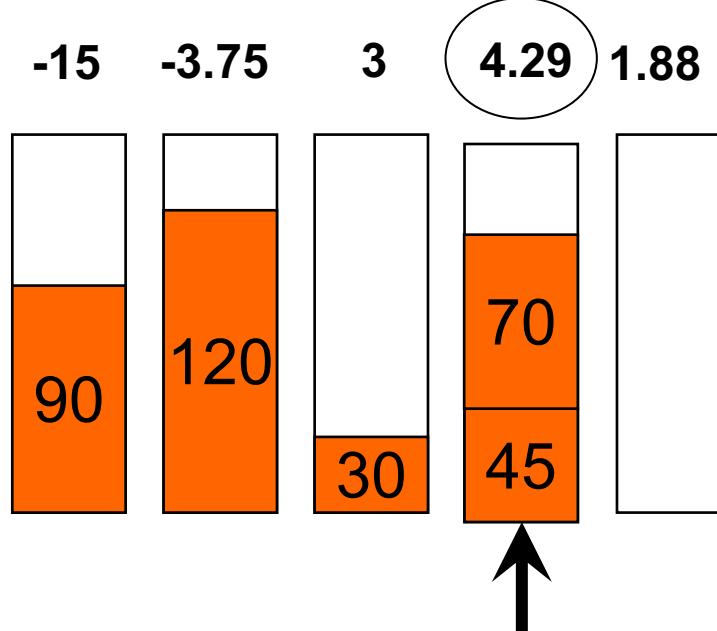
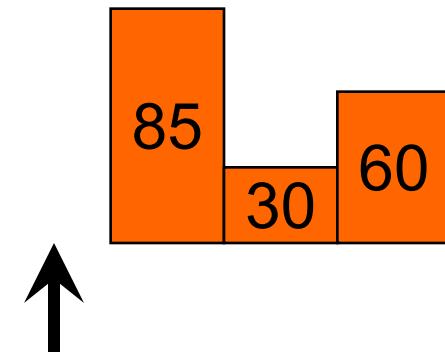
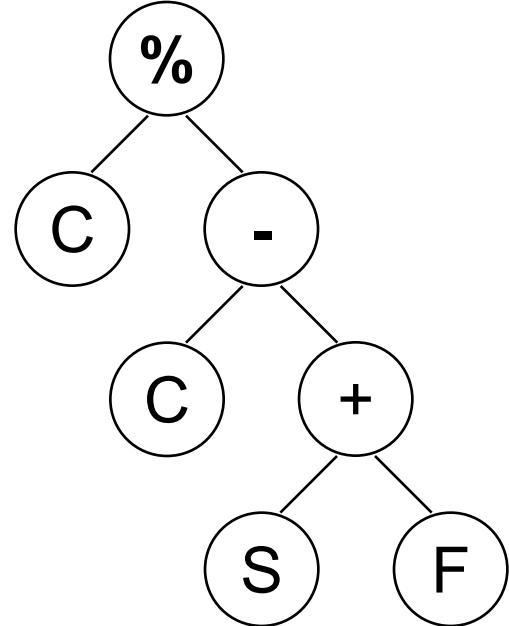
Genetic Programming Heuristics – Bin Packing



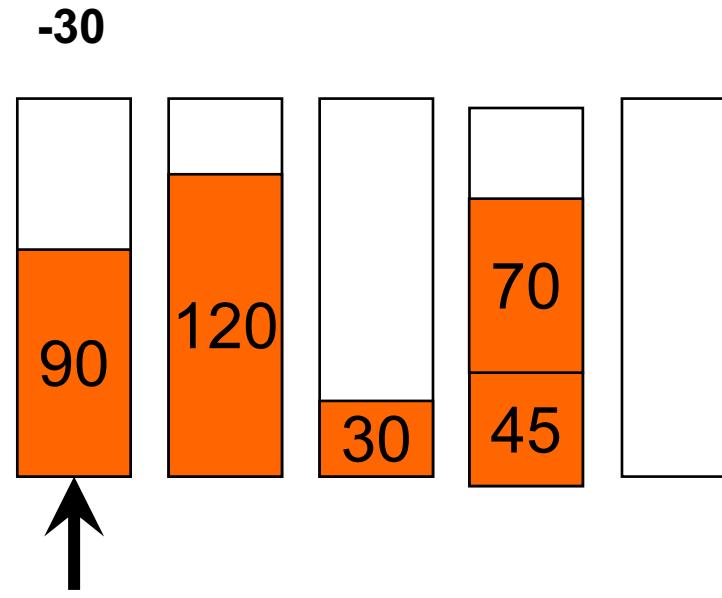
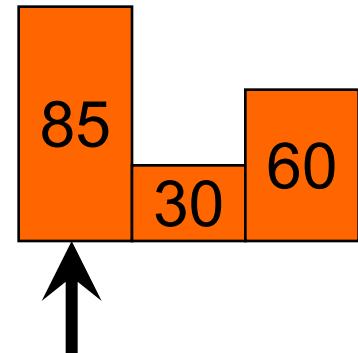
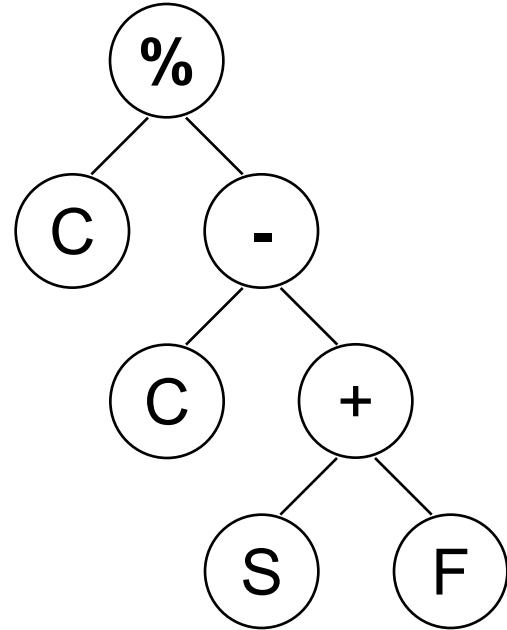
Genetic Programming Heuristics – Bin Packing



Genetic Programming Heuristics – Bin Packing



Genetic Programming Heuristics – Bin Packing





Experiments

- Compared against results from 18 Papers
- 1D Bin Packing
 - ▶ 2 Instance Sets
- 2D Knapsack and Bin Packing
 - ▶ 10 Instance Sets
- 3D Knapsack and Bin Packing
 - ▶ 6 Instance Sets

Results



Bin Packing Problem

Dimensions	Instance Name	Percent Improvement
1D	Uniform	0
	Hard	-0.4
2D	Beng	0
	Ngcut	0
	Gcut	-1.2
	Cgcut	0
3D	Thpack9	-2.3

Knapsack Problem

Dimensions	Instance Name	Percent Improvement
2D	Okp	+1.2
	Wang	0
	Ep30	-0.9
	Ep50	+0.4
	Ep100	-2.6
	Ep200	+2.4
	Ngcut	-4.3
	Gcut	+1.2
	Cgcut	-1.7
3D	Ep3d20	+13.0
	Ep3d40	+10.2
	Ep3d60	+6.0
	Thpack8	-0.5
	Thpack9	+0.7
	BandR	-2.9

GP hyper-heuristic for packing



- A **more general** packing methodology for 1D, 2D and 3D bin packing and knapsack problems
- **Achieved generality without the loss of solution quality**



Summary

- A search method with different components, algorithmic configurations and/or parameter settings often performs differently
- A metaheuristic performs search over space of solutions while a hyper-heuristic (which can be a metaheuristic) performs search over the space of (low level) heuristics
- Selection hyper-heuristics can be used to mix perturbative as well as constructive low level heuristics



Summary II

- The choice of low level heuristics used in a hyper-heuristic approach influences its performance
- Genetic programming hyper-heuristic can be used to build heuristics or heuristic components
 - ▶ Often they operate in a train and test fashion and
 - ▶ Training on selected sample instances could take long time while application to unseen instances is generally fast
 - ▶ Each tree generated by GP can be evaluated using an indicator showing how good it is in building high quality solutions to the sample problem instances, such as, mean quality of solutions over the sample instances

Q&A



Thank you.
Ender Özcan

ender.ozcan@nottingham.ac.uk

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
<http://www.cs.nott.ac.uk/~pszeo>

Advanced Topics

Lecture 9

Ender Özcan



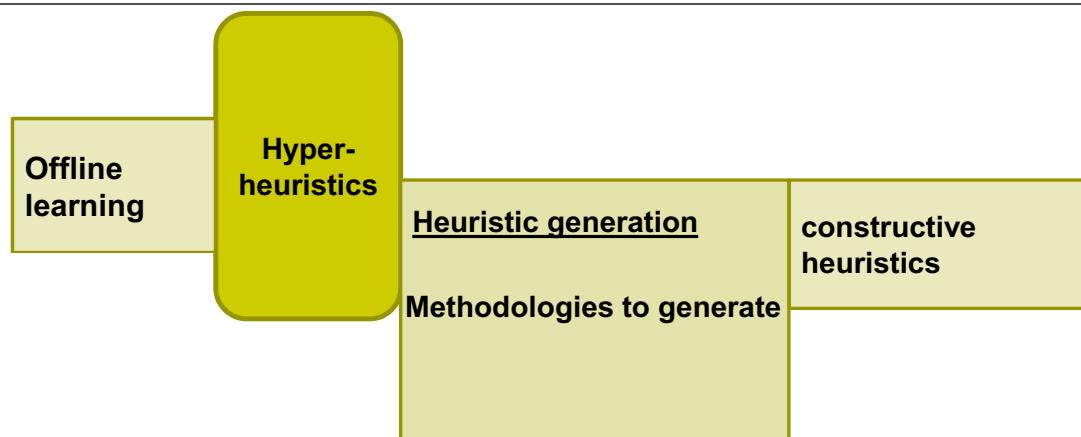
The University of
Nottingham



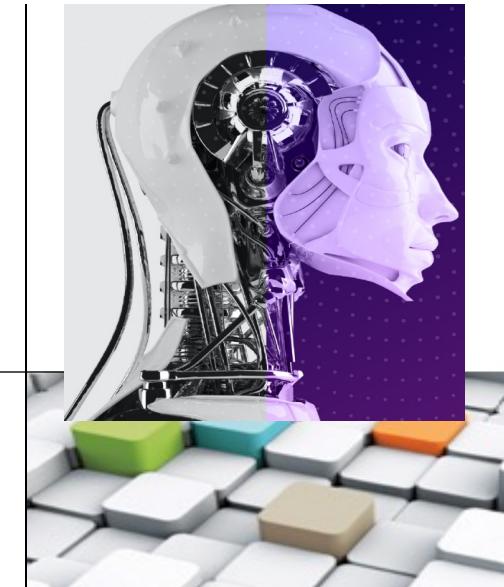
UNITED KINGDOM • CHINA • MALAYSIA

509

Policy Matrix Evolution for Generation of Heuristics



S. Asta, E. Özcan, and A.J. Parkes, CHAMP: Creating Heuristics via Many Parameters for Online Bin Packing, Expert Systems With Applications, vol. 63, pp. 208-221, doi:10.1016/j.eswa.2016.07.005, 2016



The University of
Nottingham





1D Offline Bin Packing

Pack a **set** of items of sizes s_i for $i = 1, \dots, n$

- ▶ Sizes are integer values and $s_i \in [1, C]$
- ▶ C is the fixed capacity of each bin

in such a way that

- ▶ Never exceed bin capacity
- ▶ Minimise number of bins used

Standard NP-hard problem



1D Online Bin Packing

Pack a **stream** of items of sizes s_i for $i = 1, \dots$

- ▶ Sizes are integer values and $s_i \in [1, C]$
- ▶ C is the fixed capacity of each bin

upon their arrival (one item at a time)

in such a way that

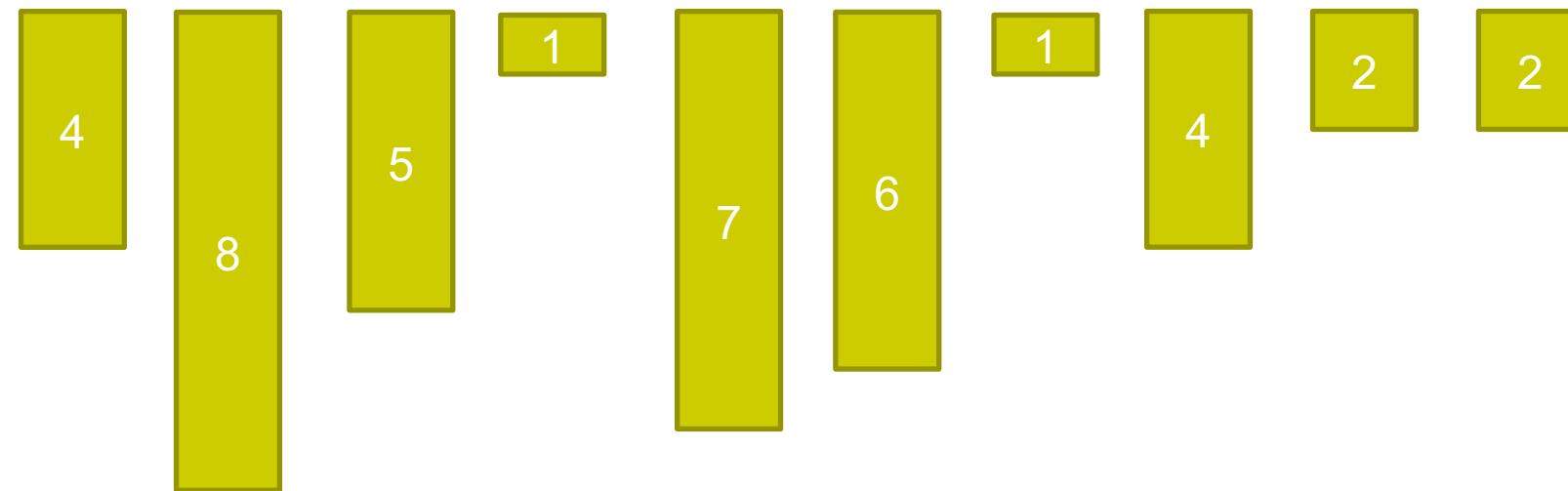
- ▶ Never exceed bin capacity
- ▶ Minimise number of bins used (Maximise the average bin-fullness) **at the end**



Standard Heuristics: First-fit and Best-fit

- **First Fit Heuristic (BP-FF).** Place the items in the order in which they arrive. Place the next item into the lowest numbered (left most) bin in which it fits. If it does not fit into any open bin, start a new bin.
- **Best Fit Heuristic (BP-BF).** Place the items in the order in which they arrive. Place the next item into that bin which will leave the least room left over after the item is placed in the bin. If it does not fit in any bin, start a new bin.

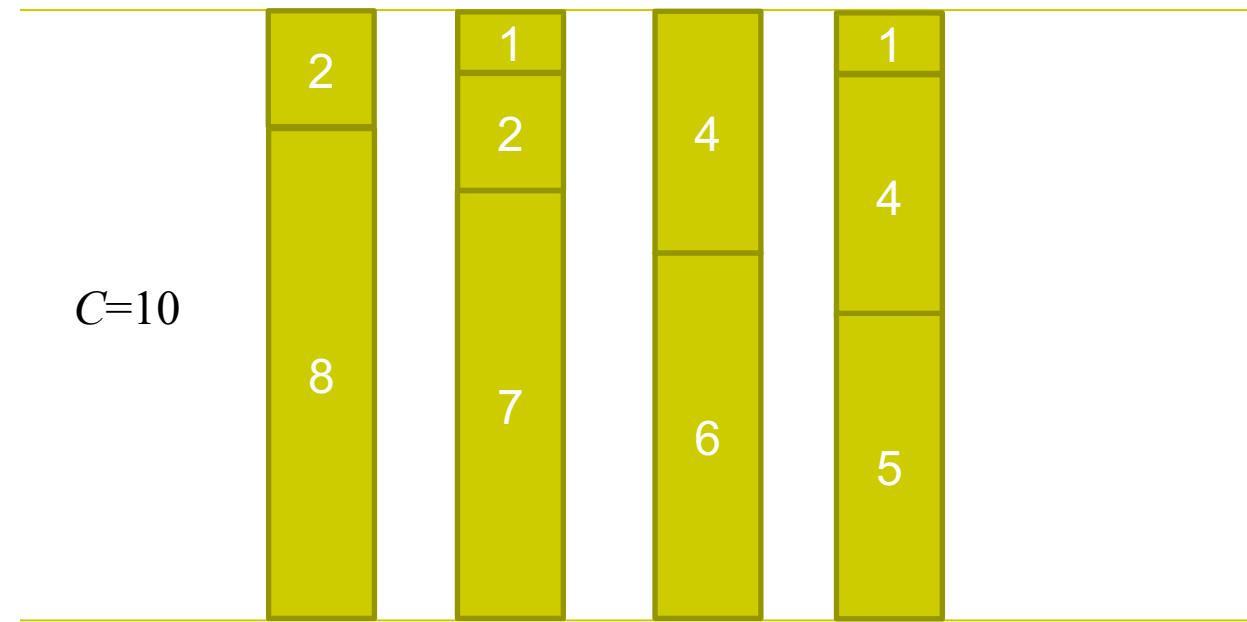
$$S = \{4, 8, 5, 1, 7, 6, 1, 4, 2, 2, \dots\} \quad C=10$$





Standard Heuristics: First-fit and Best-fit

$S = \{8, 7, 6, 5, 4, 4, 2, 2, 1, 1\}$ $C=10$ Offline Case

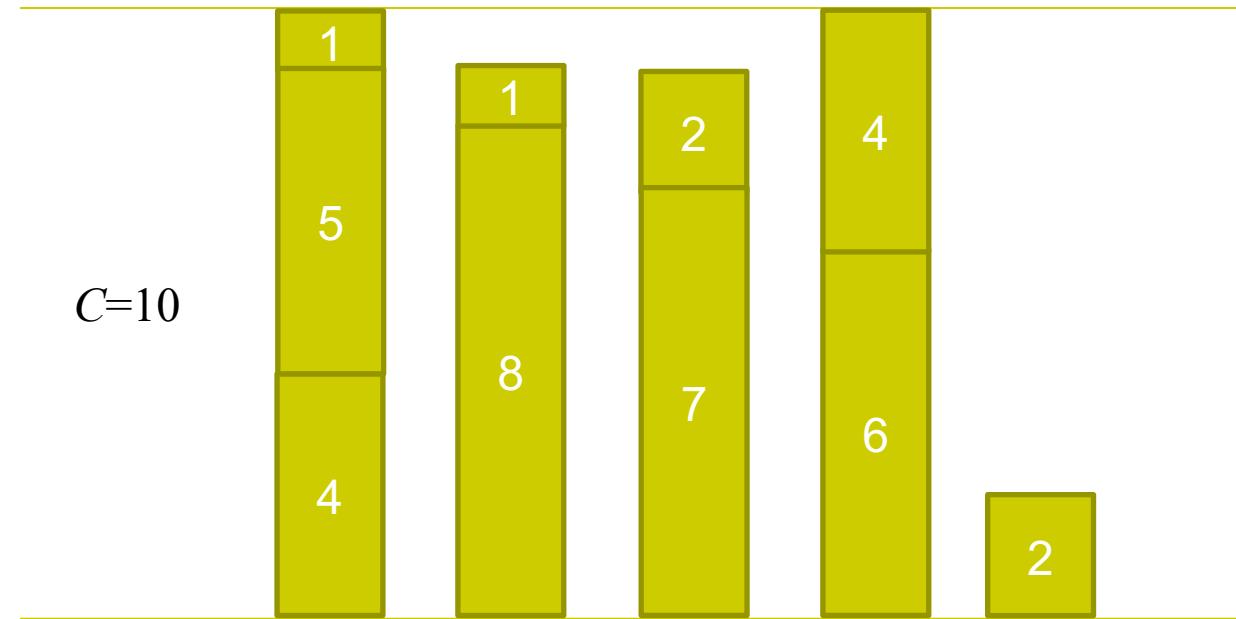


4 bins (average bin-fullness: 100%)



Standard Heuristics: First-fit and Best-fit

$S = \{4, 8, 5, 1, 7, 6, 1, 4, 2, 2, \dots\}$ $C=10$ Online Case



5 bins (average bin-fullness: 80%)



“Index Policies”

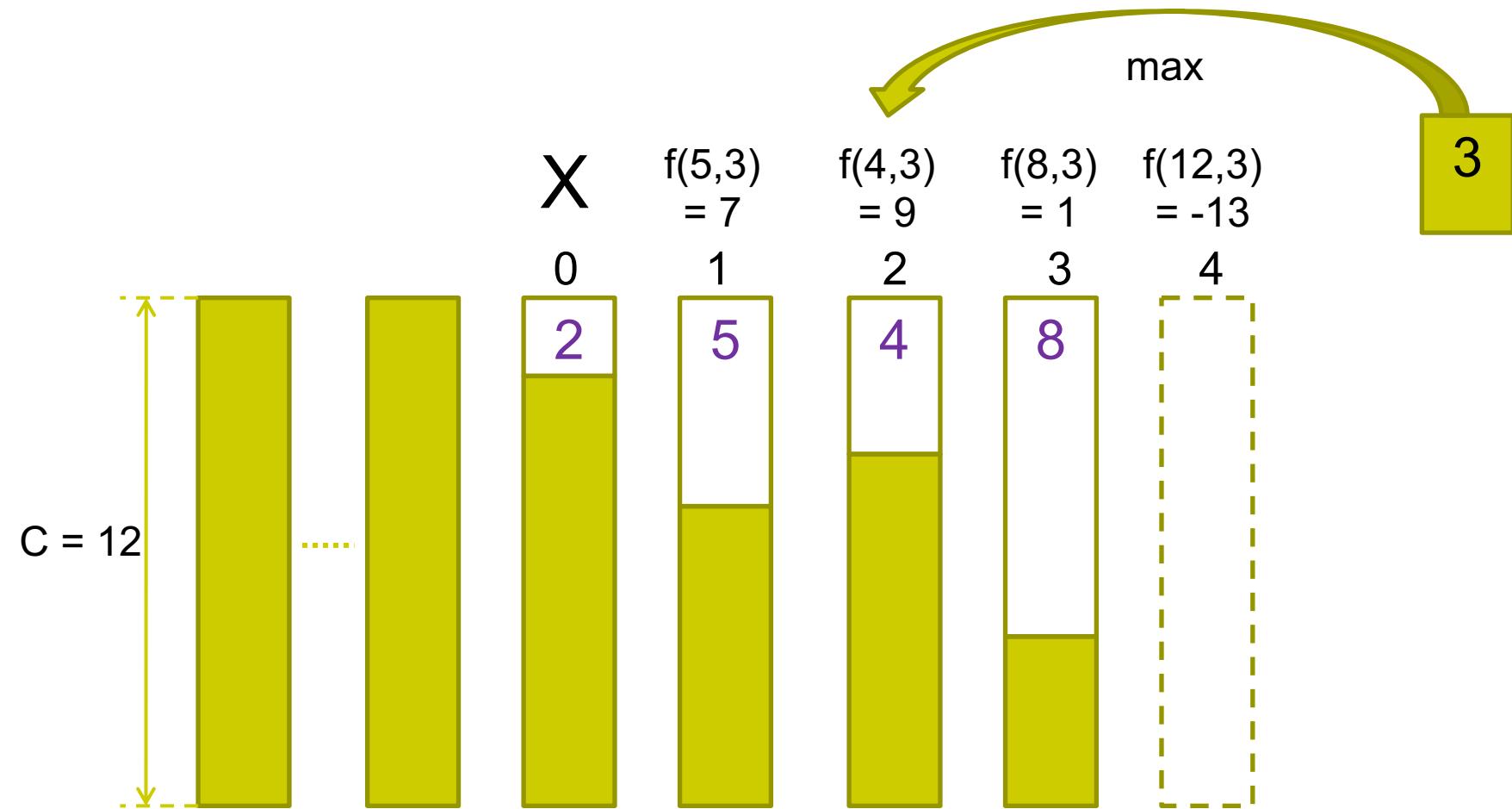
- “index policy”: each choice option is given a score, or “index value” independently of the other options
 - ▶ The option with the highest index value is taken
 - ▶ Also need a rule to break ties
- Although index policies are a special case, in some situations, they can be optimal, or at least very good

Index Policies for 1D Online Bin Packing



- Given
 - ▶ r : remaining capacity of bin (residual space)
 - ▶ s : item size
- score of bin is $f(r,s)$ for some function f
- Given a new item of size then place into bin with largest value of $f(r,s)$
- We will break any ties using FF:
 - ▶ **place item in earliest bin with the best available score**

1D Bin Packing – Intermediate Step





1D Online Bin Packing

- A new empty bin is always available (**open**)
- A bin is **closed** if it can take no more items
 - ▶ e.g. if **residual space** is smaller than size of any item
- We need a *good* “policy”, i.e. a method to assign a new item upon its arrival to one of the *open* bins

Potential General Method for 1D Online Bin Packing



- On arrival of new item of size s_i
 - ▶ Inspect the current set of open bins
 - ▶ Simultaneously use the entire set of residual spaces in the open bins to pick where to place the new item
- This is difficult and expensive (in general)



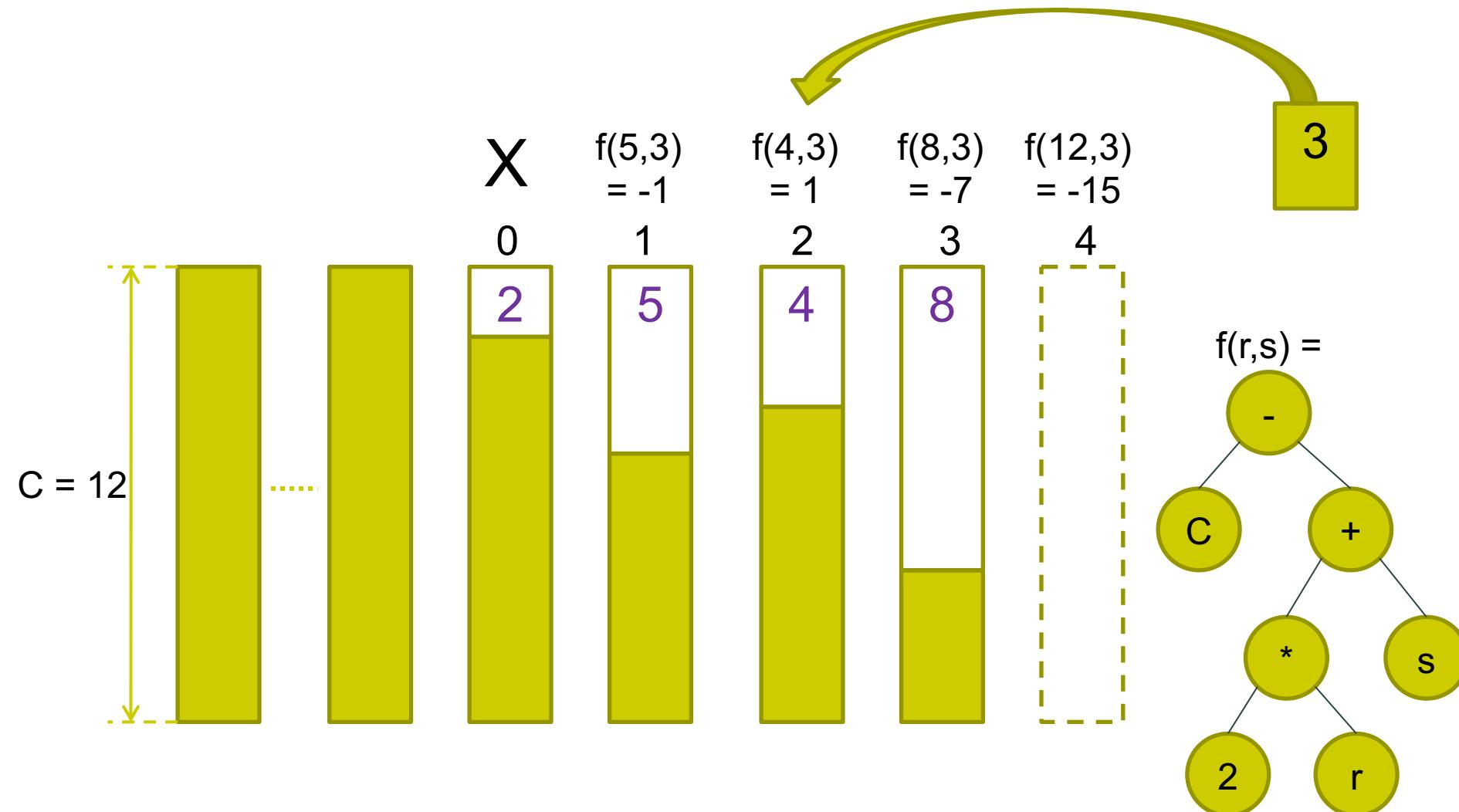
Generating Heuristics

- Within search methods, often have score functions, “index functions” to help make some choice
 - ▶ difficult to invent successful ones; want to automate this
- GP approach: evolve arithmetic score functions

E. K. Burke, M. R. Hyde, G. Kendall, J. R. Woodward: Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. GECCO 2007: 1559-1565 [[PDF](#)]

- ▶ Use Genetic Programming to learn $f(r,s)$
- ▶ $f(r,s)$ is represented as arithmetic function tree
- ▶ Automatically creates functions that at least match FF, BF

GP – 1D Online Bin Packing





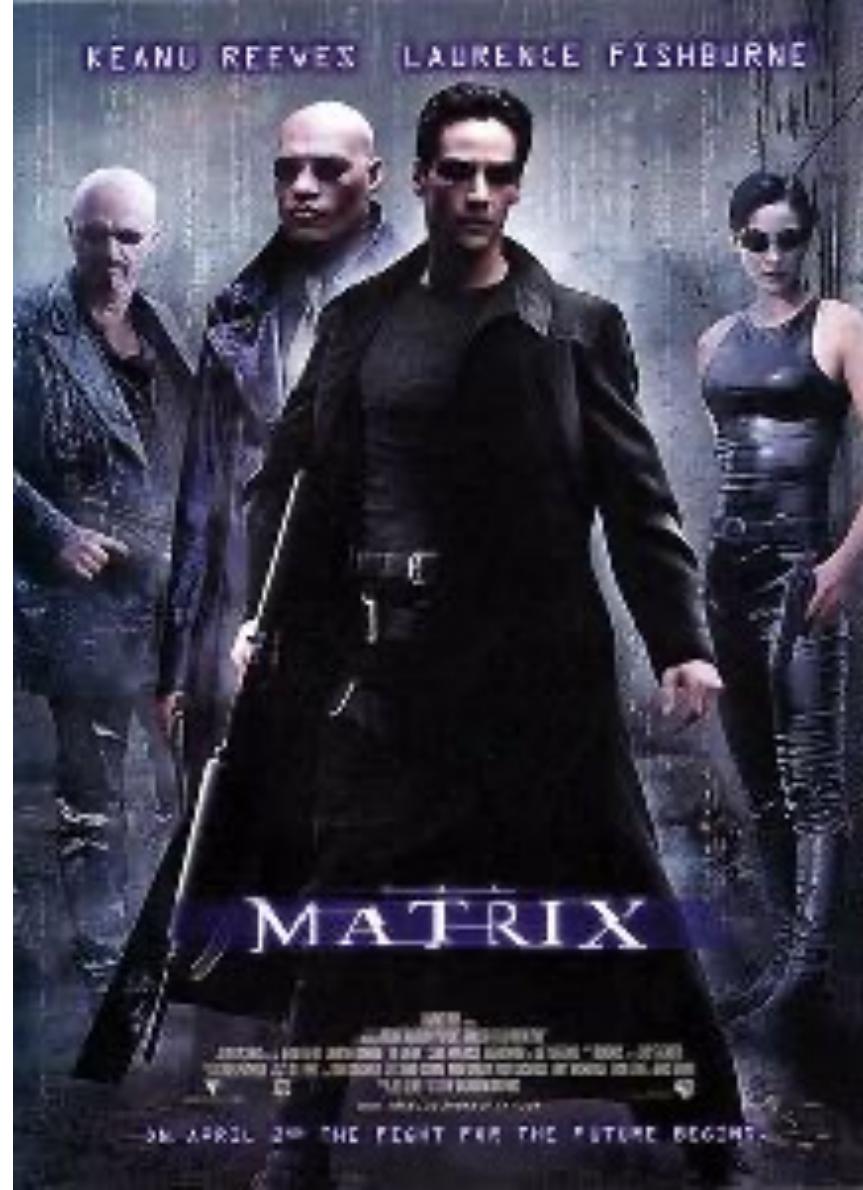
Generating Heuristics

Challenge:

- Space of functions, as used in GP,
 - ▶ is hard to understand
 - ▶ potentially biased because of the choice of representation
 - ▶ some perfectly good functions might have “bloated” representations

Can one do even better?

Is there another way to find policies?



Source: http://en.wikipedia.org/wiki/The_Matrix





Matrix View of Policies/Heuristics

- Since all item sizes (s) and residual capacities (r) are integer, then $f(r,s)$ is simply a large ($C \times C$) matrix $M(r,s)$ of **parameter values**
 $M(r,s)$

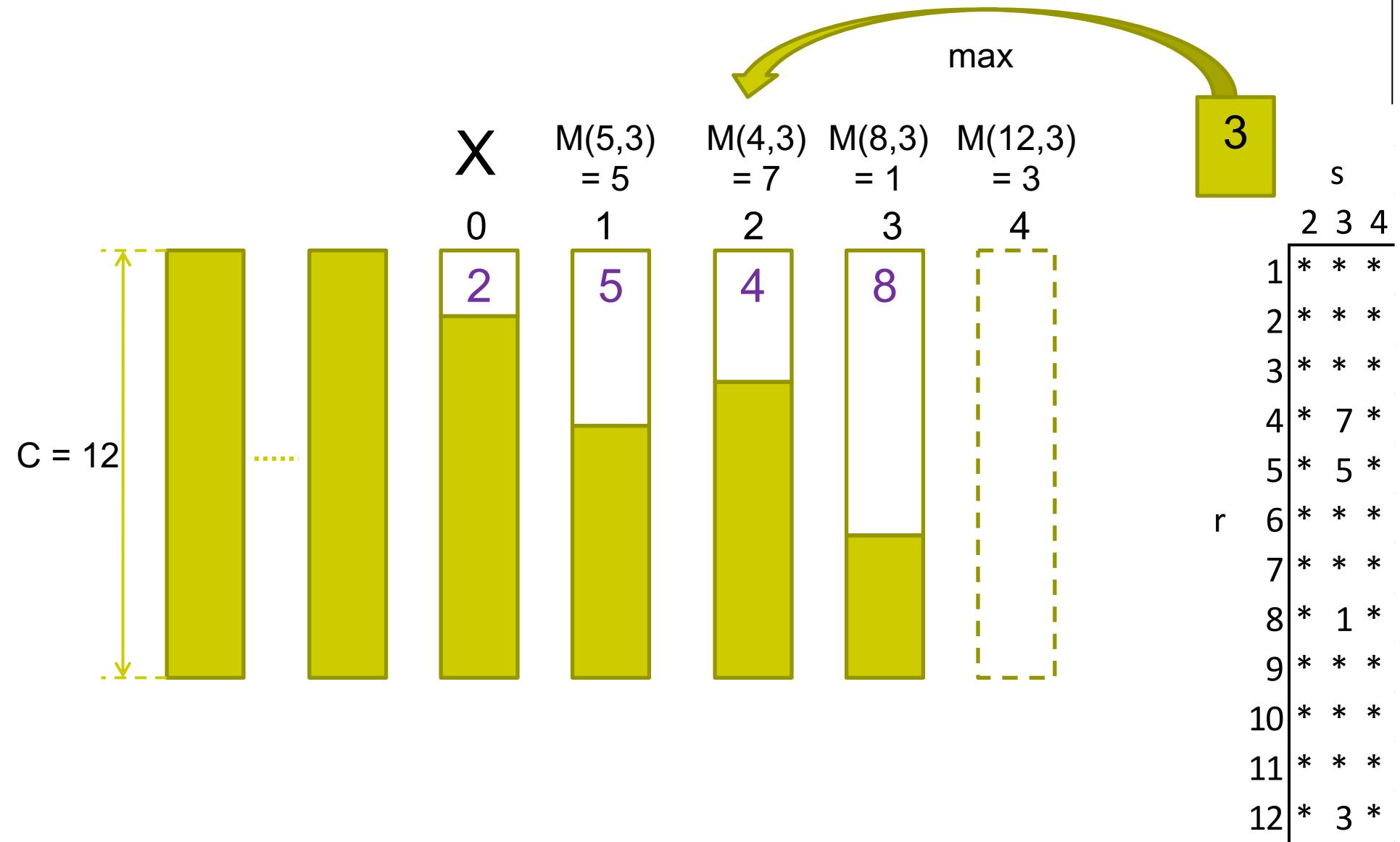
$r \setminus s$	1	2	3	4	5	6
1
2	.	2
3	.	1	2	.	.	.
4	.	2	1	.	.	↑
5
6	.	2	2	.	.	.

$r \geq s$

Diagram annotations:

- A blue arrow points from the bottom-left to the bottom-right corner of the matrix, labeled c .
- A blue arrow points from the top-right corner of the matrix to the top-right corner of the matrix, labeled c .
- A dotted line connects the bottom-right corner of the matrix to the bottom-right corner of the matrix.
- Yellow arrows point upwards from the bottom of the matrix towards the right edge, indicating rows where $r < s$.
- The text "NOT USABLE" is written vertically next to the yellow arrows.
- The text " $r < s$ " is written vertically next to the yellow arrows.

Policy Matrix – 1D Online Bin Packing





Uniform (random) Instances

We empirically studied matrix policies on Uniform Bin Packing problems

$\text{UBP}(C, s_{\min}, s_{\max}, N)$ (problem generator)

- Bin capacity C
- N items are generated with integer sizes independently taken uniformly at random from the range $[s_{\min}, s_{\max}]$
 - ▶ N is usually taken to be large: 100k



UBP(6,2,3)

- (Bin capacity 6, items are size 2 or 3 only.)
- The only perfect packings are
 - ▶ 2+2+2
 - ▶ 3+3
- Hence the ‘obvious’ policy is ...
- ... “never mix even and odd sizes”



UBP(6,2,3)

- ... “never mix even and odd sizes”
- Index policy as a matrix:
 - ▶ rows: residual capacity of the bin
 - ▶ columns: item size
- Ties are broken using First-Fit (FF)
- Grey entries “.” are never usable

resid \ item	1	2	3	4	5	6
1
2	.	2
3	.	1	2	.	.	.
4	.	2	1	.	.	.
5
6	.	2	2	.	.	.

Creating Heuristics via Many Parameters

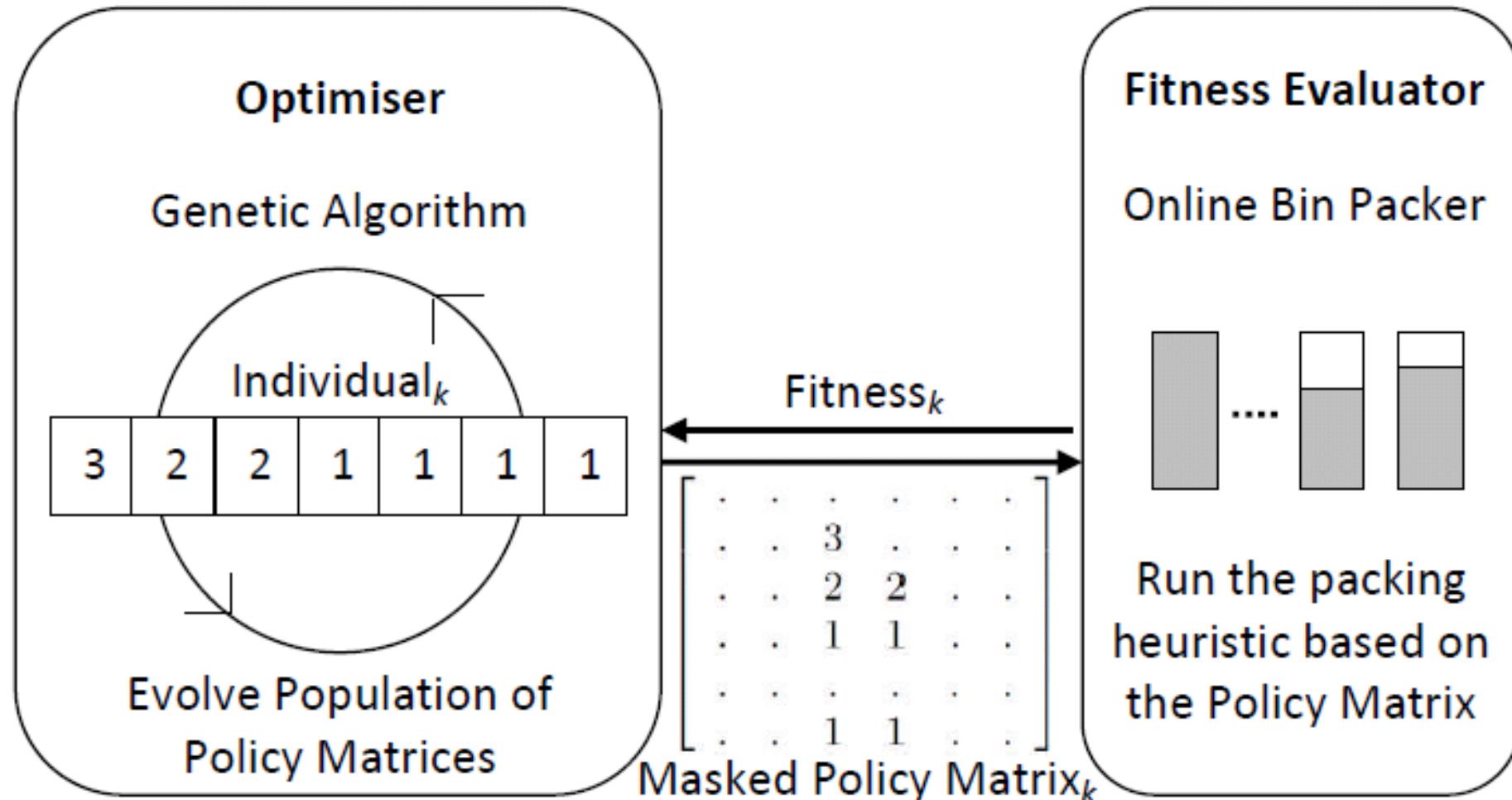
- CHAMP



- Basic idea:
 - ▶ Take values in matrix $M(r,s)$ to be integers
 - ▶ Do (meta-)heuristic search to find good choices for $M(r,s)$: Evaluation is by simulation
- Our Original Expectation:
 - ▶ the matrix will tweak the functions from GP and might slightly improve performance
- Potential expected disadvantages:
 - ▶ matrices can be much more verbose than functions
 - ▶ they fail to take into account of the good structure captured by functions



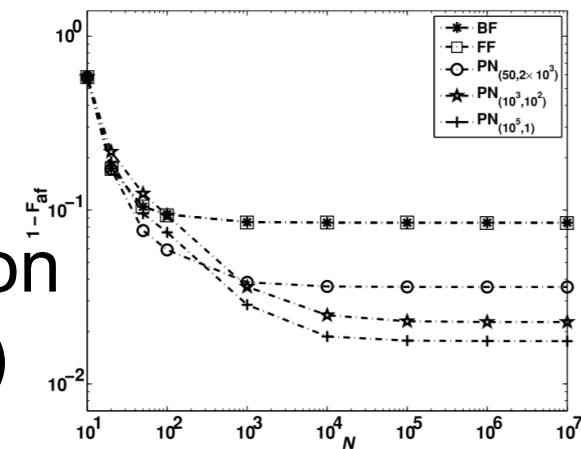
CHAMP-GA Architecture



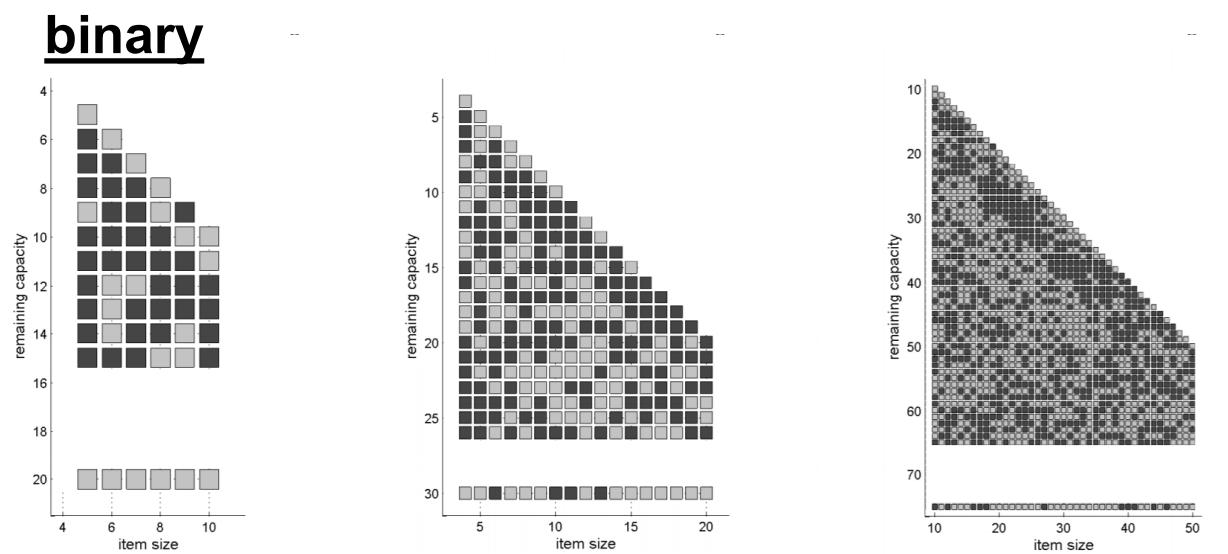
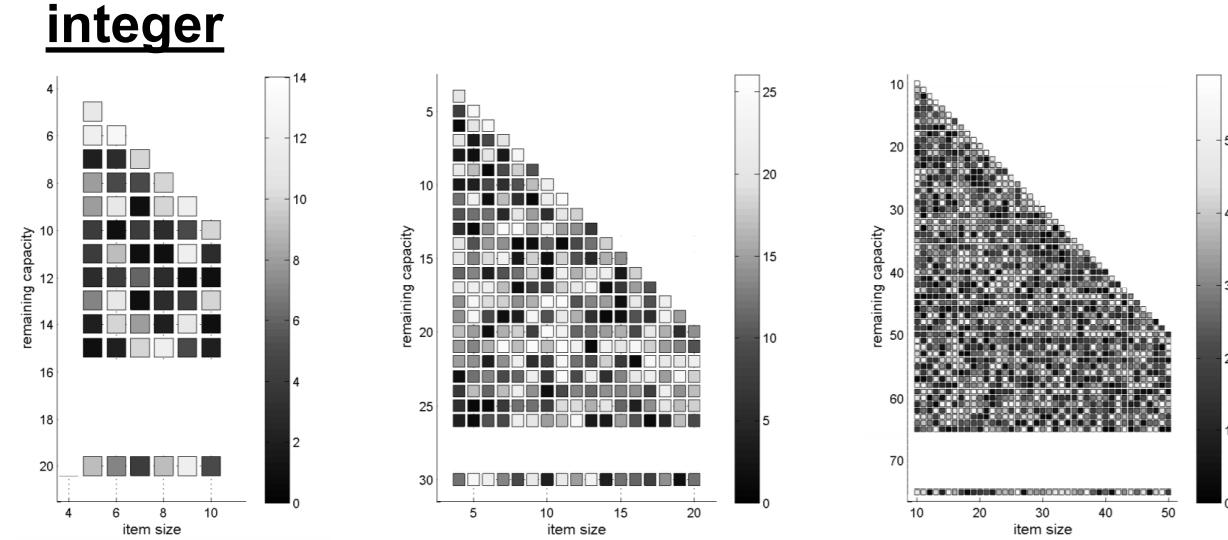
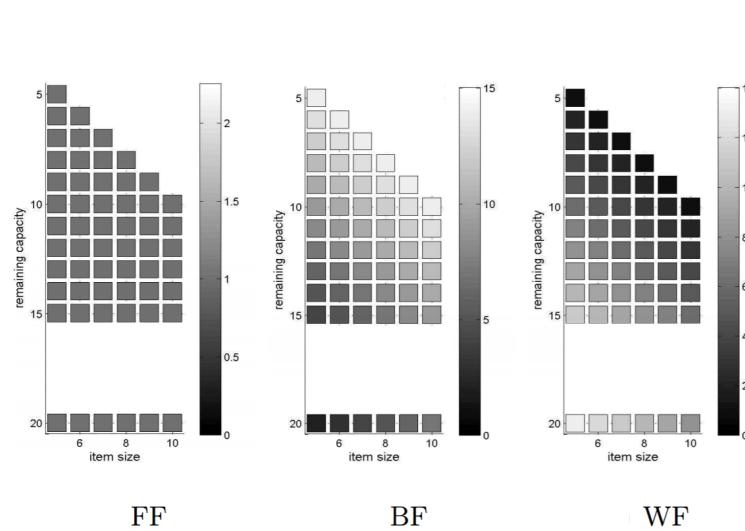


Implementation Details

- Apply a standard GA for training
 - ▶ Trans-generational (with weak elitism, population size: $C/2$), tournament selection (tour size:2), Uniform Crossover, standard mutation (with probability $1/L$), termination after 200 iterations, number of trials: 1.
- Only the active members of the matrix are stored as integer/binary values ($GA_{Original}/GA_{Binary}$) in the chromosome (GA_{FFinit} : $GA_{Original}$ where initial population contains -seeded with- an individual representing FF)
- Evaluation:
 - ▶ write matrix to a file
 - ▶ use matrix as input for a program that packs many items



Examples of good evolved matrices



- ▶ Does not look like a smooth function
 - “Weird”
 - Seems to have spikes

(a) UBP(20, 5, 10, 10^5)

(b) UBP(30, 4, 20, 10^5)

(c) UBP(75, 10, 50, 10^5)

Results – Best of runs for GA



Method	UBP(6, 2, 3, 10^5)	UBP(15, 5, 10, 10^5)	UBP(20, 5, 10, 10^5)	UBP(30, 4, 20, 10^5)	UBP(30, 4, 25, 10^5)	UBP(40, 10, 20, 10^5)	UBP(60, 15, 25, 10^5)	UBP(75, 10, 50, 10^5)	UBP(80, 10, 50, 10^5)	UBP(150, 20, 100, 10^5)
BF	92.30	99.62	91.55	96.84	98.38	90.23	92.55	96.08	96.39	95.82
FF	92.30	99.55	91.54	96.68	97.93	90.22	92.55	95.91	96.29	95.64
WF	91.70	86.58	90.54	88.61	84.10	88.66	90.80	87.94	89.25	87.73
Harmonic	-	74.24	90.04	73.82	74.21	89.10	85.18	71.59	72.96	71.97
<i>GA_{Original}</i>	99.99	99.63	98.18	99.41	98.39	96.99	99.68	98.22	98.54	97.88
<i>GA_{FFinit}</i>	99.99	99.61	98.15	99.10	99.25	96.05	98.28	98.43	97.87	96.92
<i>GA_{Binary}</i>	99.99	99.61	98.42	99.58	99.55	96.75	96.96	98.45	98.46	97.63



Conclusions

- Can use standard metaheuristics to create policies expressed in matrix representation
 - ▶ Policies exist that out-perform standard heuristics
 - ▶ Finding the policies is easier than expected
 - ▶ There are many different policies with similar performance
 - ▶ The policies are “weirder” than expected
 - The good policies could have “random” structures
 - Not necessarily easy to capture with an algebraic function of GP
 - ▶ The results can be “analysed” (inspected) to produce simple policies that out-perform standard ones
 - and that then scale to larger problems



References

- E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward (2009). Exploring hyper-heuristic methodologies with genetic programming. In C. Mumford and L. Jain (eds.), Computational Intelligence, Intelligent Systems Reference Library, pp. 177-201. Springer [[PDF](#)]
- E. Özcan, and A. J. Parkes, Policy Matrix Evolution for Generation of Heuristics, Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11), Natalio Krasnogor (Ed.). ACM, New York, NY, USA, pp. 2011-2018 (won the best paper award in the Self-* track), 2011 [[original-PDF](#)]. [[PDF](#)]
- A. J. Parkes, E. Özcan, M. Hyde, Matrix Analysis of Genetic Programming Mutation, EuroGP 2012, Lecture Notes in Computer Science 7244, pp. 158-169, 2012. [[PDF](#)]
- S. Asta, E. Özcan and A. J. Parkes, Dimension reduction in the search for online bin packing policies, Proceedings of the 2013 Genetic and Evolutionary Computation Conference Companion, pp. 65-66, 2013. [[PDF](#)]
- S. Asta, E. Özcan, and A.J. Parkes, CHAMP: Creating Heuristics via Many Parameters for Online Bin Packing, Expert Systems With Applications, vol. 63, pp. 208-221, doi:10.1016/j.eswa.2016.07.005, 2016 [[original PDF](#)]. [[PDF](#)]

Data Science (Machine Learning) Improved Hyper-heuristic Optimisation



Computational
Optimisation &
Learning Lab



The University of
Nottingham

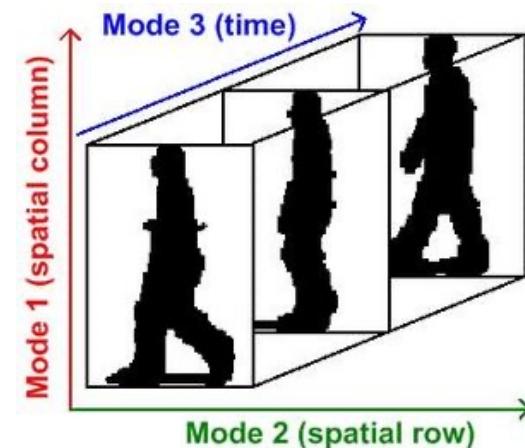
UNITED KINGDOM • CHINA • MALAYSIA

Single Objective Hyper²-heuristic: A Data Science Improved Hyper-heuristic



- Many real-world data are multidimensional
 - ▶ Very high-dimensional (big) with a large amount of redundancy
- Multi-dimensional arrays representing such data describe a tensor

Many applications in signal processing, psychometrics, social network analysis, genomics and more



S. Asta and E. Özcan, A Tensor-based Selection Hyper-heuristic for Cross-domain Heuristic Search, Information Sciences, vol. 299, pp. 412-432, 2015

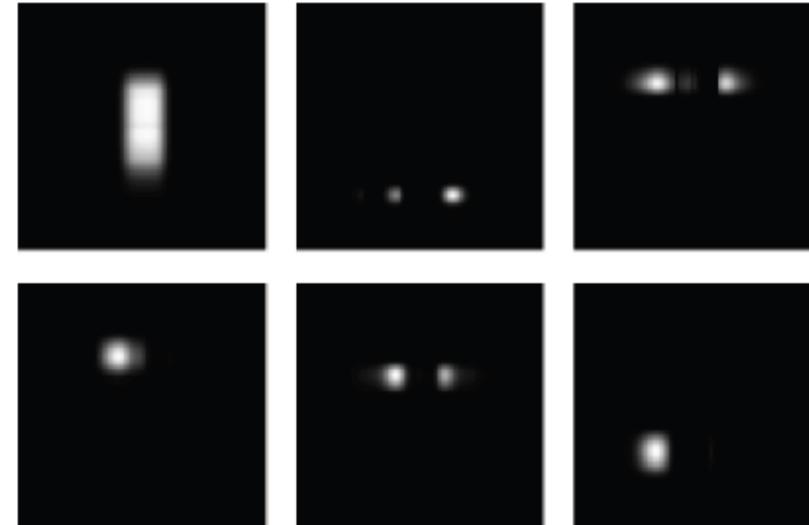


Tensor Factorisation

- There are different decomposition methods, we use Canonical Polyadic (CP) factorisation

$$\hat{\mathcal{T}} = \sum_{k=1}^K \lambda_k \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k$$

- ▶ This gives a projection of 3D data onto 1D vectors
- ▶ Helps to discover latent structures in data, quantifying the relationship between pairs of different components

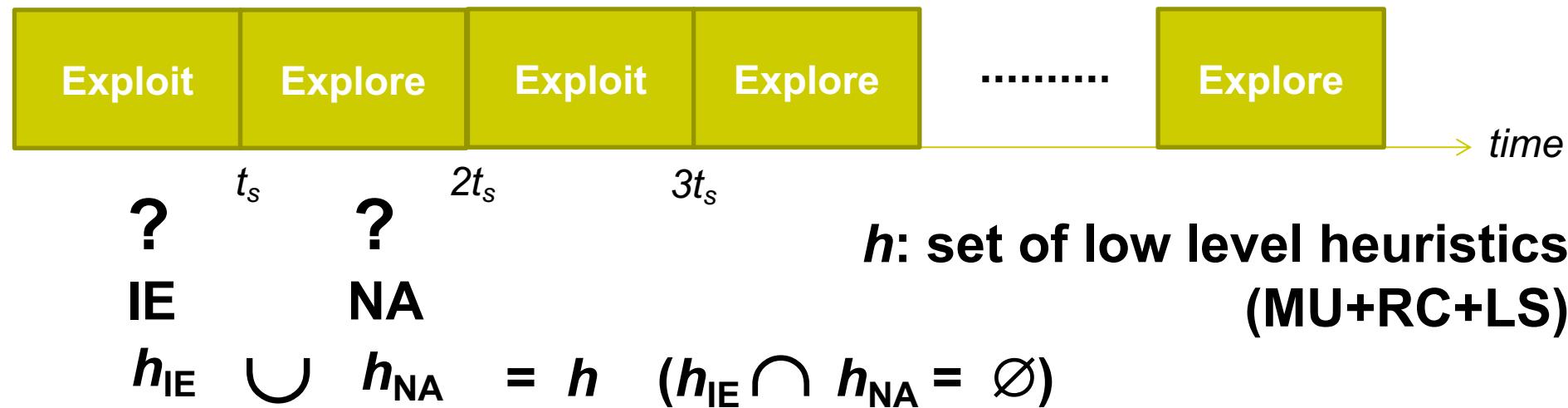


SOURCE: B. Krausz, C. Bauckhage, Action recognition in videos using nonnegative tensor factorization., in: ICPR, IEEE, 2010, pp. 1763–1766.



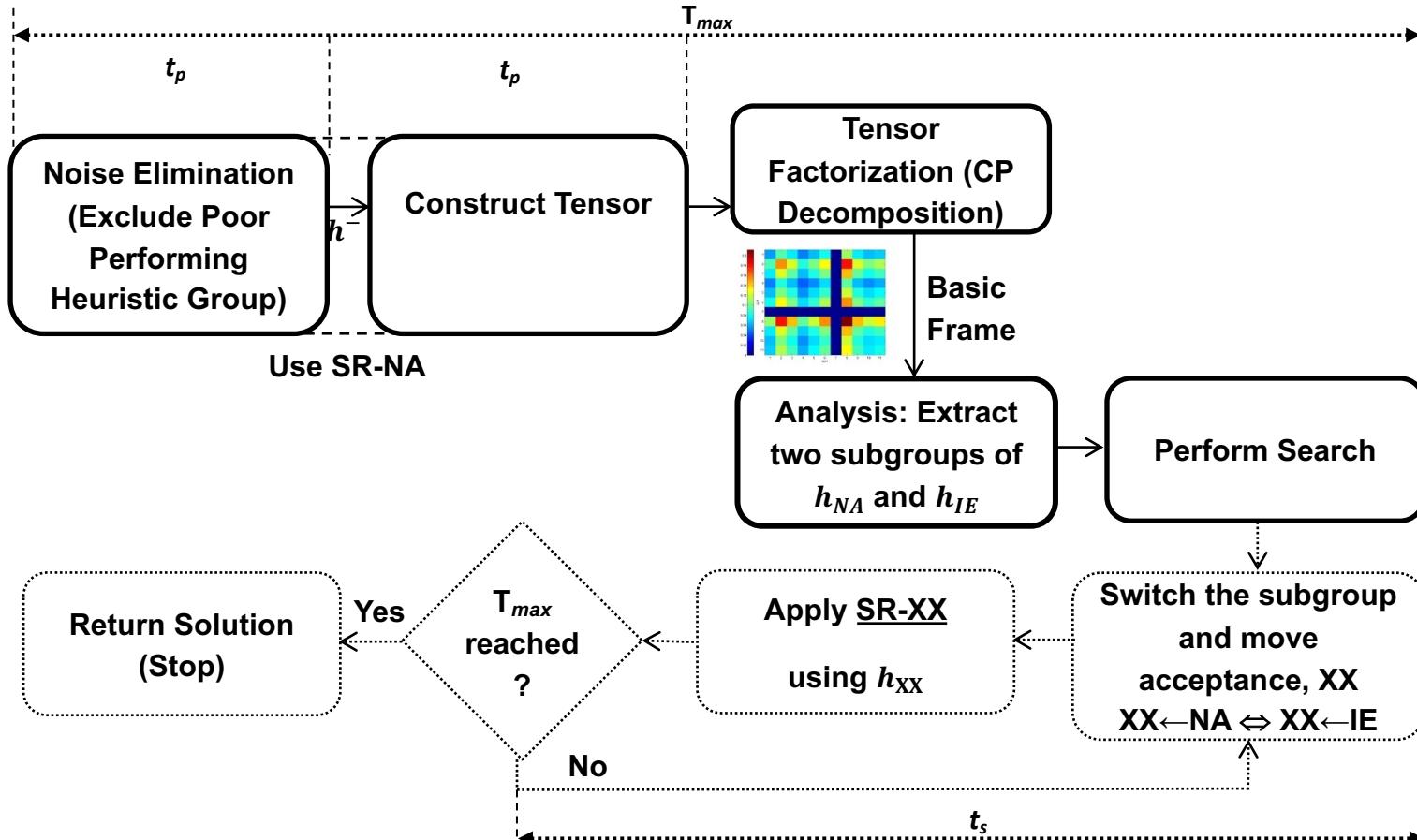
Proposed Approach – Ideas

- The balance between exploration and exploitation is crucial (e.g. ILS)

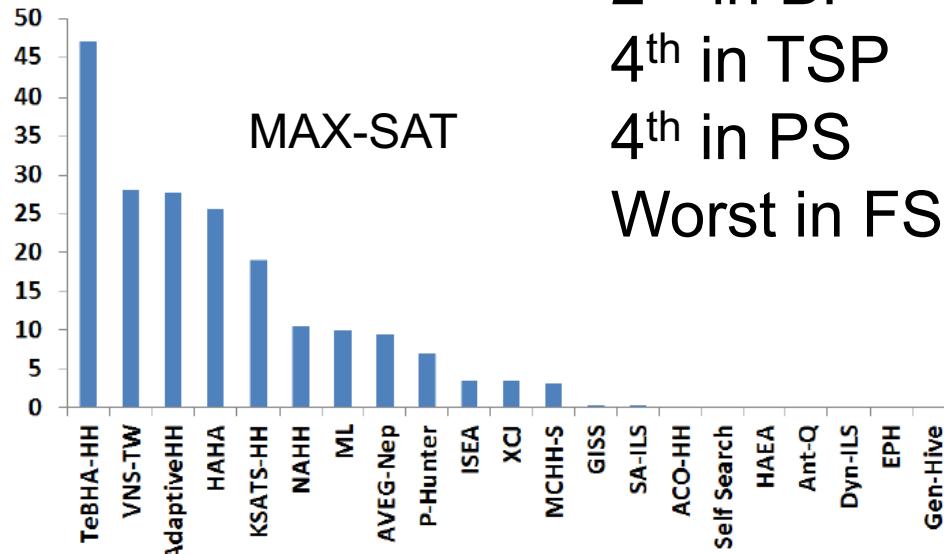


- Mix move acceptance methods
- Use machine learning to partition the low level heuristics associated with each method

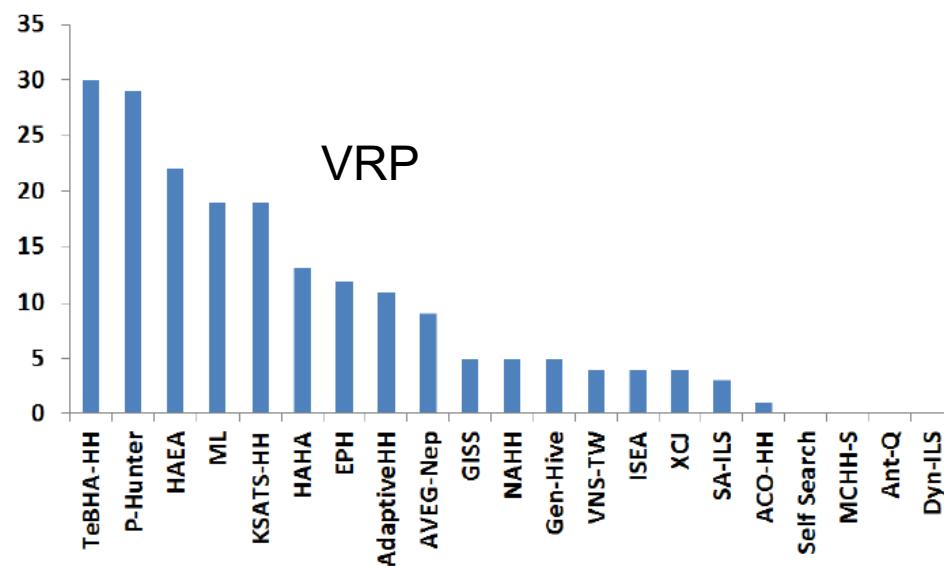
Proposed Approach – TeBHA-HH



Results

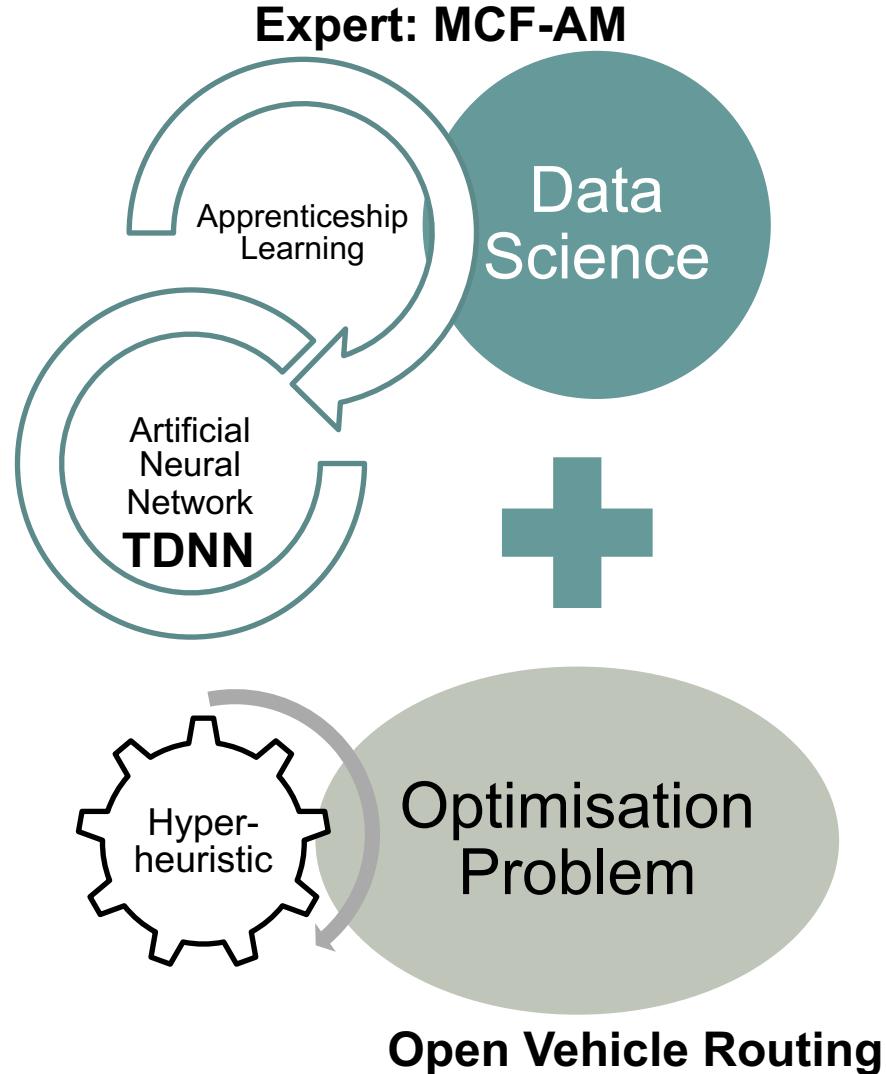


2nd in BP
4th in TSP
4th in PS
Worst in FS



Rank	Name	Score
1	AdaptiveHH	162.83
2	TeBHA-HH	148.85
3	VNS-TW	118.83
4	ML	117.50
5	P-Hunter	84.75
6	EPH	83.25
7	NAHH	68.50
8	HAHA	65.58
9	ISEA	62.50
10	KSATS-HH	52

A Single Objective Hyper²-heuristic – Apprenticeship Learning



<http://www.calinon.ch/images/hoap2-xsens01.jpg>

Automated generation of a new hyper-heuristic by observing an expert hyper-heuristic in operation



Results on Open VRP

	C1	C2	C3	C4	C5	C6	C7
(1)	5432.4	10727.2	8728.9	12934.1	18054.1	5551.6	10671.2
(2)	5560.5	10944.4	8925.6	13216.4	18168.5	5543.7	10937.9
$W(2)$	<	<	<	<	<	\geq	<
(3)	5534.3	10856.6	8838.5	13091.2	17502.2	5524.4	10860.8
$W(3)$	<	<	<	<	>	\geq	<
	C8	C9	C10	C11	C12	C13	C14
	8945.4	14094.3	18131.6	7976.8	10699.9	11579.9	10963.9
	8925.4	14229.3	18578.7	8279.1	10901.8	11539.2	10935.3
	\geq	<	<	<	<	\geq	\geq
	8958.4	13135.4	17511.1	8068.7	10827.6	11417.2	10839.9
	<	>	>	<	<	\geq	\geq

(1) vs (2)

≤ 10

> 0

≥ 4

(3) vs (2)

≤ 8

> 3

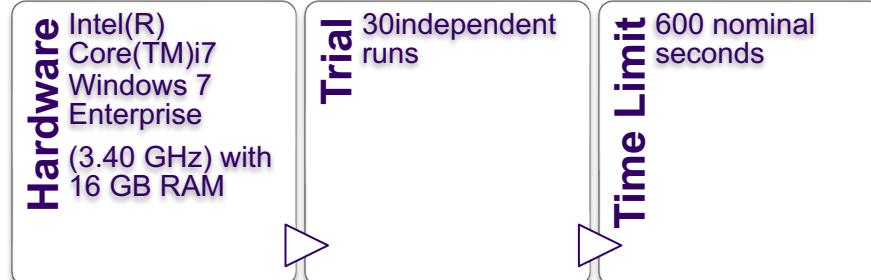
≥ 3

R. Tyasnurita, E. Özcan and R. John, Learning Heuristic Selection using a Time Delay Neural Network for Open Vehicle Routing, Proc. of the 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 1474-1481.

(1) TDNN-ALHH (objective values and distance)

(2) EXPERT (MCF-AM)

(3)TDNN-ALHH (objective values)





Summary I

- Hyper-heuristic research originated from a job shop scheduling application and has been rapidly growing since then.
- Generation hyper-heuristics are commonly used in the area
 - ▶ Train and test fashion
 - Does the selected subset of training instances is sufficiently representative of the test set?
 - Training is time-consuming (delta/incremental evaluation, surrogate functions)
 - ▶ The generated/evolved heuristics might not be easy to interpret, yet they can outperform human designed heuristics



Summary II

- There is empirical evidence that machine learning/analytics/ data science help to improve the hyper-heuristic search process
 - ▶ Problem features vs solution/state features
 - ▶ Offline versus online learning – Life long learning
- There is still a lack of benchmarks
- Automated design of search methodologies is extremely challenging
 - ▶ Addressed in almost complete absence of a mathematical and theoretical understanding

Q&A



Thank you.

Ender Özcan ender.ozcan@nottingham.ac.uk

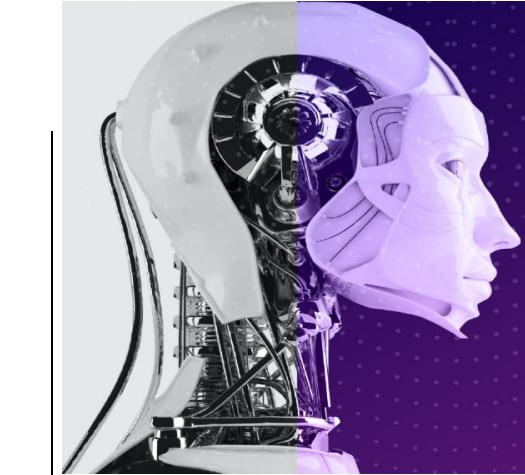
University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK

<http://www.cs.nott.ac.uk/~pszeo>

COMP2001/COMP2011 Artificial Intelligence Methods

Ender Özcan

Revision Lecture



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA



What did we cover?

- Introduction to heuristic search / optimisation / decision support, heuristic types, search paradigms, case studies: bin packing, TSP, graph colouring, classroom assignment, pseudo-random numbers.
- Components of heuristic search: representation, evaluation, neighbourhoods
- Basic hill climbing/local search methods: First Improvement, Best Improvement, Davis's bit hill climbing, Random mutation hill climbing



What did we cover? II

- Metaheuristics
 - ▶ Components: representation, evaluation, initialisation, move operators, guideline, stopping conditions, mechanism for escaping the local optima
 - ▶ Single point based search (Local Search Metaheuristics): Iterated Local Search, Tabu Search
 - ▶ Move acceptance methods: Taxonomy, Late Acceptance, Great Deluge, Simulated Annealing,...
 - ▶ Parameter setting: parameter tuning vs parameter control.
- Performance Comparison of Algorithms



What did we cover? III

- Evolutionary Algorithms: GAs, MAs, MMAs
- Hyper-heuristics, classification
 - ▶ Selection Hyper-heuristics: heuristic selection and move acceptance, methods controlling constructive heuristics, HyFlex
 - ▶ Generation Hyper-heuristics: genetic programming, genetic algorithm for generating online bin packing heuristics
 - ▶ Advanced topics
- Fuzzy systems and planning for robotics



Examinable Material

- Materials provided over Moodle and in lecture
 - ▶ What's written on the slides
 - ▶ What's in the papers provided for reading
 - ▶ What's in the videos
 - ▶ What's in the engagement activities
 - ▶ What's said in lectures
 - ▶ What's written on the whiteboard
 - ▶ Asynchronous/synchronous exercises solved (forums and quizzes)
 - ▶ Questions: Asked during a lecture
 - ▶ Answers: Said or written during a lecture

Bloom's Taxonomy (a subset)



- The form of knowledge needed to answer an exam question:
 - ▶ Knowledge – for what we often term ‘bookwork’ based upon recall of factual information’
 - ▶ Comprehension – or ‘understanding’, where students are asked to perform such actions as to describe, explain, classify ideas or concepts
 - ▶ Application – for the situations where a student is asked to undertake an ‘unseen’ task by using knowledge in a new way (or a variant upon one previously shown in class)



SAMPLE PAST EXAM QUESTIONS



Warning

- Wherever you provide your reasoning or explanation do not return lengthy answers.
 - ▶ Providing a lengthy answer may increase the chances that you make a mistake
 - ▶ Every erroneous comment in a solution/answer will get penalised
- Be concise and do not exceed the word limitation.
- There is no solution where your answer should exceed half a page.

What is/are ...?

Define...

Name X (e.g., 2) ...



- Heuristic
- Graph (colouring) heuristics
- Types of search methods
- Tardiness
- Hill climbing
- Metaheuristic
- Hyper-heuristic

- Combinatorial optimisation problem
- Single point based search
- Evolutionary algorithm
- Delta/incremental evaluation
- Fuzzy set, ...



True/False

- Which one of the following statements are TRUE?
 - (i) A seed value will correspond to a specific sequence of generated values for a given random number generator.
 - (ii) A heuristic search algorithm might result with a poor solution.
 - (iii) A search algorithm based on the nearest neighbor constructive heuristic/operator can always find a better solution than a search algorithm based on the pairwise exchange perturbative heuristic/operator for solving a given instance of the Travelling Salesman Problem.

(i), (ii) only



Multiple Choice

- Please indicate which algorithm below is a local search metaheuristic:
 - a. Ant colony optimisation
 - b. Particle Swarm Optimisation
 - c. Iterated Local Search
 - d. Greedy Randomized Adaptive Search Procedure

What is/are ...?

Define...

II

Name X (e.g., 2) ...

- Mechanisms for escaping local optima
- Population based metaheuristics
- Local search metaheuristics
- Components of metaheuristics

What is (are) the difference(s)/similarity (-ies) between...?

- Memetic algorithm and multimeme memetic algorithms
- Genetic algorithm metaheuristic and genetic algorithm hyper-heuristic



What is (are) the difference(s)/ similarity(-ies) between...?



- System effectiveness and system efficiency
- Optimisation and decision support
- Mutational and hill climbing heuristics
- Constructive and perturbative heuristics
- Simulated annealing and choice function
- Iterated local search and memetic algorithms
- Exploration and exploitation in search



Computation Question

- Consider 4 jobs having the processing times p_j , the due-dates d_j and the weights w_j of the jobs $j=1, \dots, 4$, given in the table:

jobs	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

What is the weighted tardiness for the schedule:

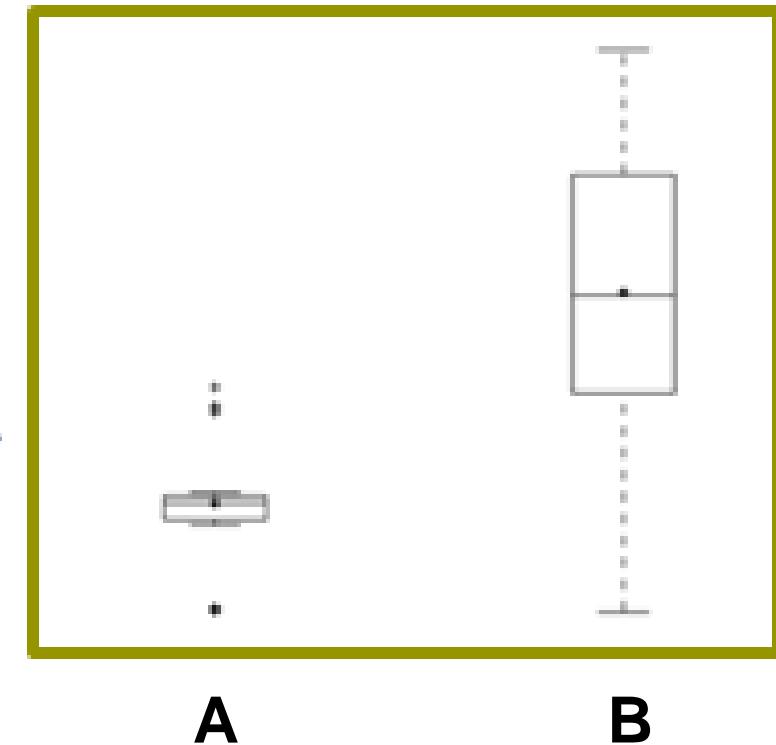
$\langle 2, 1, 4, 3 \rangle$?



Performance Comparison I

- Given two algorithms A and B which are run 100 times on an instance of MAX-SAT and box plot of all objective values from all runs for each algorithm are provided on the right,
- Which algorithm performs better on the instance with a statistically significant performance difference?

Assume a minimisation objective function

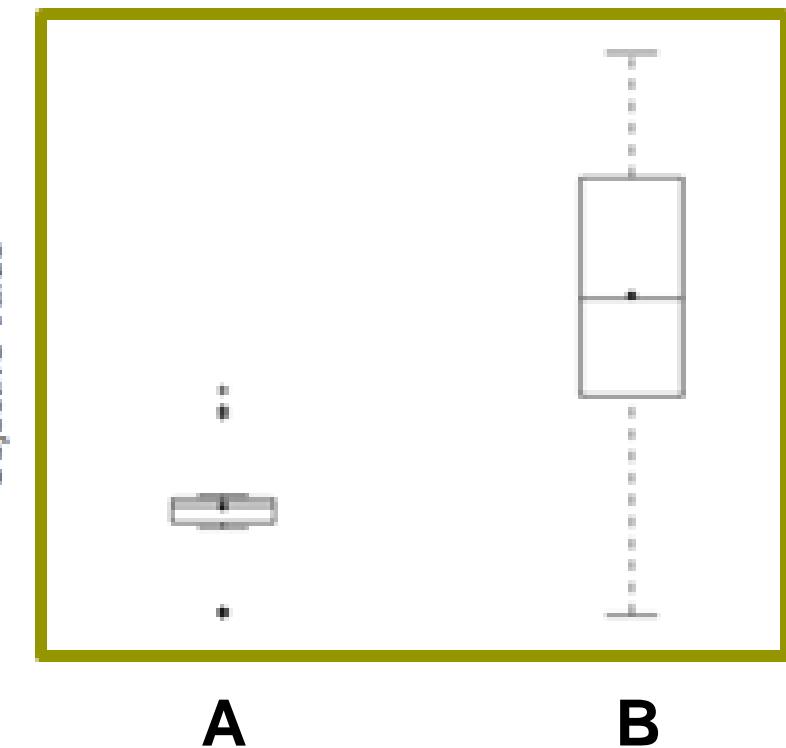




Performance Comparison II

- Given two algorithms A and B which are run 100 times on an instance of MAX-SAT and box plot of all objective values from all runs for each algorithm are provided on the right,
- Which algorithm performs better on MAX-SAT with a statistically significant performance difference?

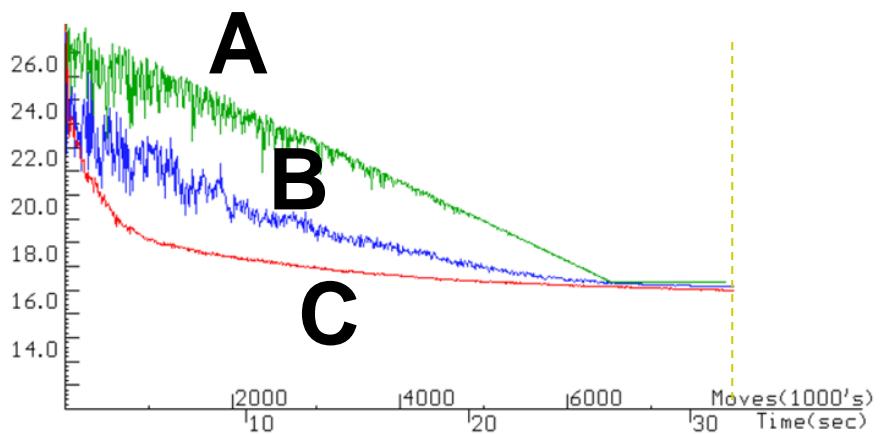
Assume a minimisation objective function





Performance Comparison III

- Given three algorithms A, B and C which are run 100 times on an instance of a MAX-SAT problem and the mean objective value progress plots in time are in the right,
- Which algorithm performs the best with significance in the given duration?



Computation Question – Applying an Algorithm



- A bus company has 50 busses, each with 52 seats.
- There will be a concert outside a city and groups of people will arrive at the city centre to attend it. The company will be responsible for the transportation of all groups to the concert site from the city centre. Each group of people has to be placed into the same bus. Different groups are allowed in the same bus. The capacity of a bus cannot be exceeded.
- The company aims to reduce the number of busses used for transportation.
- 33 groups arrive each having the following number of people: 8, 44, 10, 10, 10, 10, 10, 10, 25, 25, 25, 25, 25, 25, 25, 9, 9 , 12, 12, 12, 13, 13, 11, 11, 11, 12, 12, 12, 44, 46, 37
- By using an appropriate packing heuristic, place all groups into busses. Please give details in each step of your heuristic.

Computation Question –Executing an Algorithm



```
1 long seed = 123456789;  
2 Random generator = new Random(seed);  
3 double num;  
4 for (int trial=0; trial<4; trial++)  
5     num = generator.nextDouble() ;  
6 System.out.print(num); // print out num
```

Given the pseudo-code above and assuming that the seed 345678912 produces the following sequence of double values in the given order

<0.13, 0.14, 0.75, 0.39, 0.234, 0.005, 0.81,>

which value would be printed out after the for loop?

0.39

Computation Question – Representation Design



- **Maximum Satisfiability** is the problem of determining the maximum number of clauses, of a given Boolean formula in conjunctive normal form, that can be made true by an assignment of truth values to the variables of the formula
- Given a MAX-SAT problem with N variables. How many configurations can be encoded using a binary representation?
- How would you evaluate a given solution?



Algorithm Design Problem

- **Maximum Satisfiability** is the problem of determining the maximum number of clauses, of a given Boolean formula in conjunctive normal form, that can be made true by an assignment of truth values to the variables of the formula.
- Design an Iterated Local Search/Tabu Search/Simulated Annealing... algorithm for solving this problem, discussing each algorithmic component and parameter setting.

Computation Question – Applying an Algorithm



- **Maximum Satisfiability** is the problem of determining the maximum number of clauses, of a given Boolean formula in conjunctive normal form, that can be made true by an assignment of truth values to the variables of the formula.
- Given the problem instance: $(x_1 \vee x_2) \wedge (\neg x_4 \vee x_6) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_6) \wedge (\neg x_2 \vee x_3) \wedge (x_3 \vee x_5)$ and current solution of 110100, what is the solution that will be returned by best-improvement (steepest ascent)? Show all your steps.

Exercise Question:

Illustration of a Run of Tabu Search on a Scheduling Problem



Example:

jobs	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
$1 \mid d_j \mid \sum w_j T_j$	14	12	1	12

Schedule four jobs on a machine

$$T_j = \max(C_j - d_j, 0)$$

tardiness of job j

Neighbourhood operator: go through all schedules that can be obtained through adjacent pairwise interchanges, choose the best.

Tabu-list: pairs of jobs (j, k) that were swapped within the last two moves

Run the algorithm for 2 while loop iterations, starting with the initial solution: $S_0 = <2, 1, 4, 3>$

Simulated Annealing with Geometric Cooling



INPUT: $T_0 (> 0)$, α (cooling rate < 1.0)

Generate an initial schedule S_0 using some heuristics

Set $S_k = S_{best} = S_0$, $k=0$;

REPEAT

Select $S_{new} \in \mathcal{N}(S_k)$ // Make a move from S_k to S_{new} based on \mathcal{N}

If $F(S_{new}) < F(S_k)$ **then** $S_{k+1} = S_{new}$ // an improving move is made

else // A worsening solution is obtained

generate a random uniform number in $(0,1]$, $U(0,1)$

$$-\frac{F(S_{new}) - F(S_k)}{T_k}$$

If $U(0,1) < e^{-\frac{F(S_{new}) - F(S_k)}{T_k}}$ **then** $S_{k+1} = S_{new}$ // Accept worsening move

else $S_{k+1} = S_k$ // Reject worsening move

// Keep track of the best solution found so far

If $F(S_{new}) < F(S_{best})$ **then** $S_{best} = S_{new}$

$T_{k+1} = \alpha T_k$; // geometric cooling: multiply previous temp with α (value < 1.0)

$k = k+1$;

UNTIL (*stopping condition* = true)

Exercise Question:

jobs	1	2	3	4
p_j	9	9	12	3
d_j	10	8	5	28
w_j	14	12	1	12



- Consider the scheduling problem $1 \mid d_j \mid \sum w_j Tard_j \ (F)$
 $Tard_j = \max(C_j - d_j, 0)$ **tardiness** of job j
- Apply the simulated annealing to the problem starting out with the $<3, 1, 4, 2>$ as an initial sequence.
- Neighbourhood operator:** perform a random adjacent pairwise interchange
- Choose $\alpha = 0.9$ and $T_0 = 0.9$
- Use the following numbers in the given order as **random numbers where appropriate**
 - for choosing a random job i in $[1..4]$: $<1, 3, 2, 4, \dots>$ to apply adjacent pairwise interchange of (i) th and $(i+1)$ th entries in the permutation
 - for $U(0,1)$: $<0.17, 0.009, \dots>$



Algorithm Design Problem II

- High school timetabling (HST) is the problem of assigning a time-slot to each lesson taught by a teacher and taken by students without any clashes, minimising the number constraints that are violated.
- Design a genetic algorithm using a direct solution representation for solving the HST assigning each lesson to a time-slot, e.g., 5 days x 8 time-slots (per day).
- Your solution must explain all your algorithmic choices, in particular chromosome length, (candidate solution) representation showing how a complete solution can be obtained with respect to the chromosome length, initialisation, genetic operators, replacement, termination and any other relevant parameter settings.

Q&A



Thank you.

Ender Özcan ender.ozcan@nottingham.ac.uk

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
<http://www.cs.nott.ac.uk/~pszeo>