Introduction to Optimisation using Metaheuristics

Prof Ender Özcan

School on Advanced Topics in Computational Mechanics

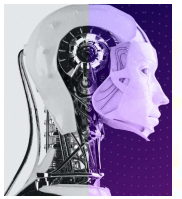
Computational Optimisation & Learning Lab

UKacm
UK Association for Computational Mechanics
2022

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Outline

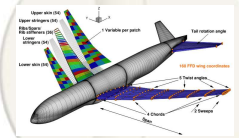
- Preliminaries
  - Search spaces, search paradigms, search methods
- Metaheuristics (Design issues)
  - A classification of metaheuristics, main components: representation, evaluation function, neighbourhoods, escaping from local optima, termination criteria
  - Tabu Search
  - Evolutionary/Genetic Algorithms



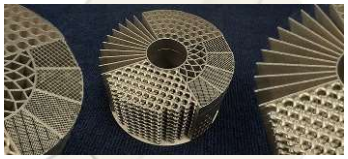
Optimisation Problems

- Optimisation problems appear in various forms across industry, leisure and the public sector
- Majority of them are computationally hard to solve (NP hard)

Sustainable Energy

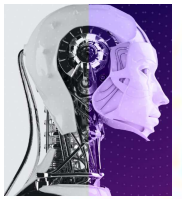


Engineering Design



Additive Manufacturing

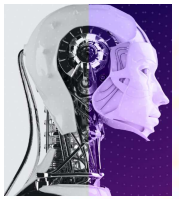
Optimisation



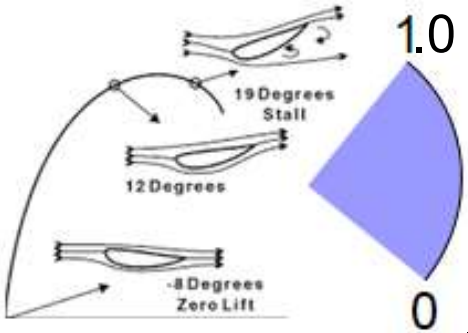
maximise/minimise $z = f(\boldsymbol{X})$,
subject to $g_i(\boldsymbol{X}) \leq \boldsymbol{b}_i\ ,\ (= \mid \geq)$
where $\boldsymbol{X}$ is a vector of variables < x_1 , x_2 ,…, x_n>, i=1..m

- Solving an optimisation problem requires search for optimum
- Fundamental problem of optimisation is to arrive at the **best** possible solution (**optimal** – $\boldsymbol{X}_{\text{opt}}$=argmin z) in any given set of circumstances.
- In most cases **optimal** is unattainable

Search in Continuous vs Discrete Space



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

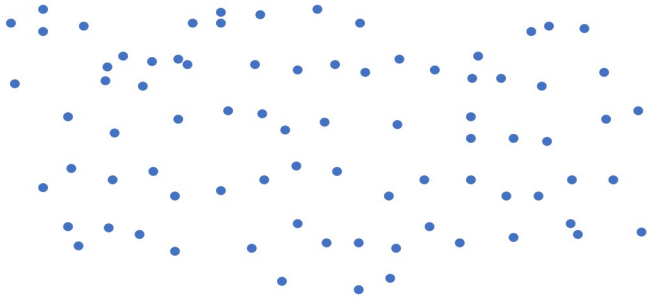


1.0 19.0 Degrees

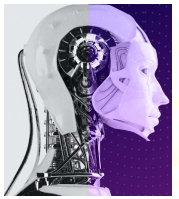
…

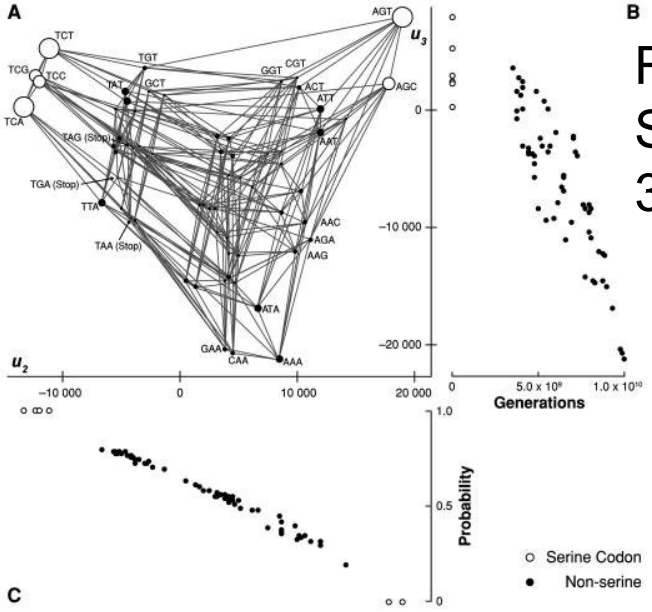
0 -8.0 Degrees

- Find the tour (visiting sequence) with the optimum (minimum) travelling distance, given 81 cities.

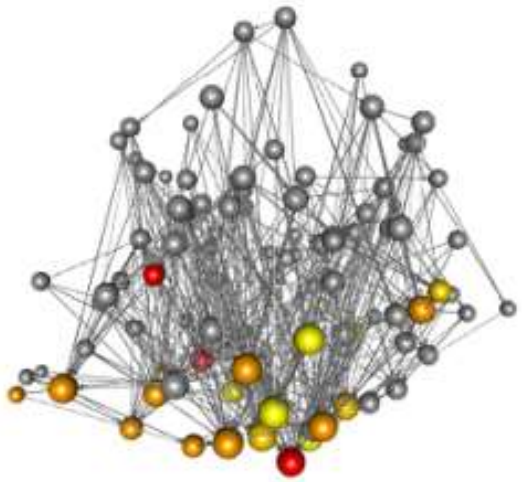


Search for an Optimal Solution





RNA
Sequencing:
3 nucleotides



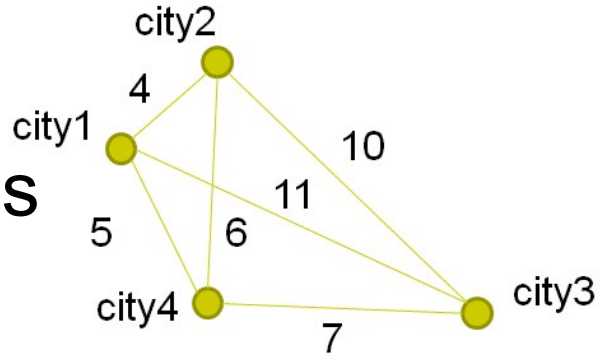
Permutation
flowshop scheduling:
10 jobs and 7 machines

TSP:
pr1002

http://www.cs.stir.ac.uk/~goc/papers/joh-2017-global.pdf
http://www.cs.stir.ac.uk/~goc/papers/flowshop-LON-cec2017.pdf

Search Paradigms I

- Single point (trajectory) based search vs. Multi-point (population) based search
- Constructive
  - Search on partial candidate solutions



<2> → <2, 1> → <2, 1, 4> → <2, 1, 4, 3> (26)

vs.

- Perturbative
  - Search on complete solutions

<2, 4, 1, 3> (32)→ <1, 4, 2, 3> (32) → <1, 2, 4, 3> (28) → <1, 2, 3, 4> (26)

Search Paradigms II

deterministic ⟷ **stochastic**

systematic ⟷ **local search**

sequential ⟷ parallel

---

single objective ⟷ multi-objective

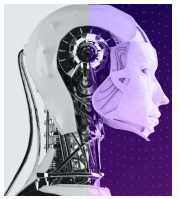
[See http://www.cs.ubc.ca/~hoos/SLS-Internal/ch1.pdf pp.23-30]

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,…

Need for (meta)heuristic optimisation methods

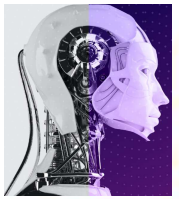


- Travelling salesman problem with *N* cities
- *N*=4, 24
- *N*=5, 120
- *N*=7, 5 040
- *N*=10, 3 628 800
- *N*=81, 5.797×10^{120}
- Number of configurations

to search from is *N*! (combinatorial explosion in the search space)

- Number of particles in the universe is in between 10^{72} – 10^{87}
- Japanese Fugaku (Since 2020): ~442.01 petaFLOPS (one thousand million million/quadrillion (10^{15}) floating-point operations per second) – ~4.16×10^{95} years (from TOP500 project).

Metaheuristics

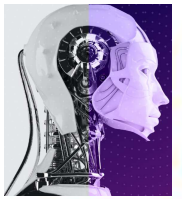


A **metaheuristic** is a high-level problem independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms

Local Search Population-based Constructive

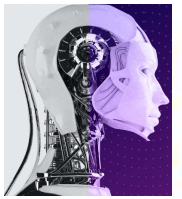
K. Sörensen and F. Glover. Metaheuristics. In S.I. Gass and M. Fu, editors, Encyclopedia of Operations Research and Management Science, pp 960–970. Springer, New York, 2013.

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)
	• [Storn & Price1997]	Differential Evolution
	• [Hansen, 1998]	CMA-ES
Constructive	• [Dorigo, 1992]	Ant Colony Optimisation (ACO)
	• [Resende, 1995]	Greedy Randomized Adaptive Search Procedure (GRASP)

Metaheuristic frameworks and software libraries



- HeuristicLab
  - Wagner and M. Affenzeller (2015), https://dev.heuristiclab.com/trac.fcgi/wiki/
- ECJ
  - Luke (2017), https://cs.gmu.edu/~eclab/projects/ecj/
- FOM
  - Parejo et al. (2003), http://www.isa.us.es/fom/
- Opt4J,
  - Lukasiewycz et al. (2011), http://opt4j.sourceforge.net/
- jMetal
  - Durillo and Nebro (2011), http://jmetal.sourceforge.net/
- JAMES
  - De Beukelaer et al. (2017), http://www.jamesframework.org/
- PISA
  - Bleuler et al. (2003), http://www.tik.ee.ethz.ch/pisa/

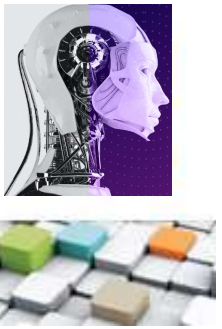
Applications of Metaheuristics to Real-world Problems

- Biological Sciences and Bioinformatics
- Computer Science
- Earth Sciences
- Engineering
- Finance and Economics
- Industry and Management
- Logistics
- Machine learning/data science
- Mathematics
- Natural Sciences
- Social Sciences
- Telecommunication, …
- Optimal design of laminated composites
- Topology optimisation of porous materials
- Solar cell design
- VLSI design
- Optimization of heat exchangers/chemical reactors
- Engineering of conducting polymers
- Improving biopolymer functions
- Optimal control of fermentation
- Cutting and packing in manufacturing
- Scheduling and routing, …

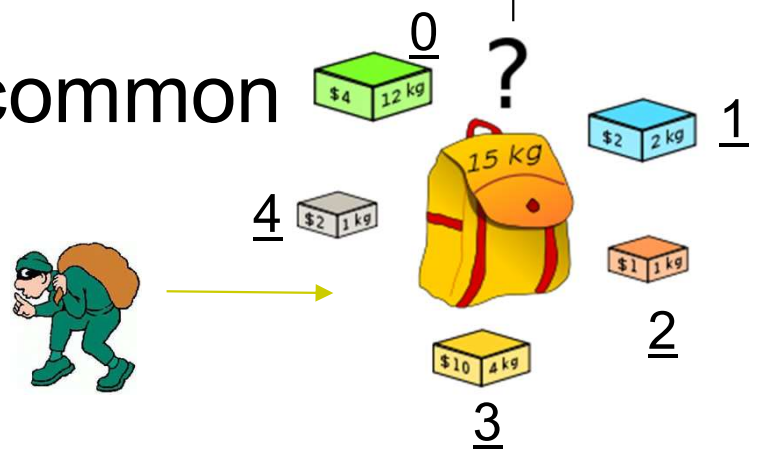
Main Components of Metaheuristics

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

Representation



- **Binary encoding** is the most common
  - 10110010110010…1011



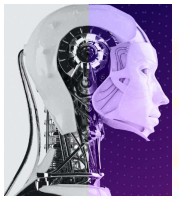
- E.g.: 0/1 Knapsack problem

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

  0 1 2 3 4
  1 0 0 1 1 pack items {0,3,4}

  0 1 2 3 4
  1 1 1 1 0 pack items {0,1,2,3}

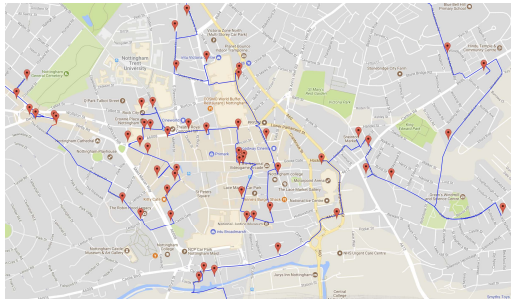
- Given a binary string of length *N* (representing *N* items), search space size is 2^N



Representation (cont.)

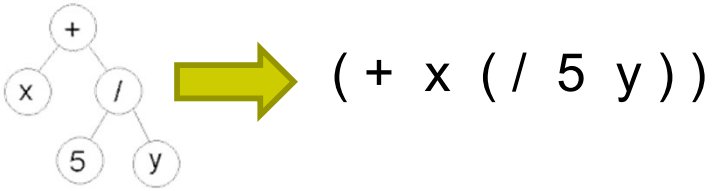
- E.g.: Travelling salesman problem, some sequencing problems
- **Permutation encoding**
  - 1 5 3 2 6 4 7 9 8

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



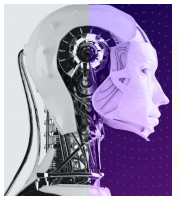
- Given *N* cities (pubs), search space size is
  - *N*!

Other Representation Schemes

- **Integer encoding**
  - 1 9 4 5 5 5 4 12 8 … 10 10 4
- **Value Encoding**
  - 1.2324 5.3243 0.4556 2.3293 2.4545
  - ATGCTTCGGCAAGACTCAAAAAATA
  - <(back), (back), (right), (forward), (left)>
- **Nonlinear Encoding**
  - 

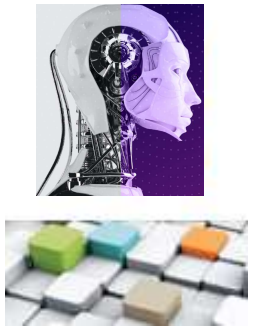
Evaluation Function

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



- Also referred to as objective, cost, fitness, penalty, etc.
  - Indicates the quality of a given solution, distinguishing between better/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.

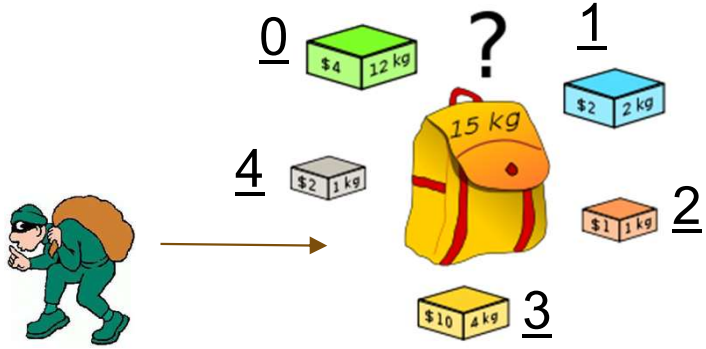
Evaluation Function (cont.)



- Evaluation functions could be computationally expensive
- Exact vs. approximate
  - Common approaches to constructing approximate 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

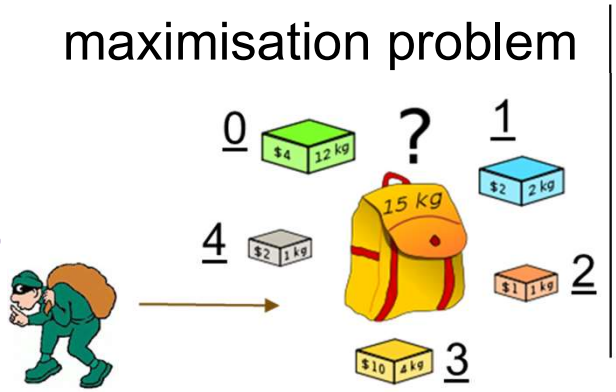
0/1 Knapsack Problem – Evaluation function

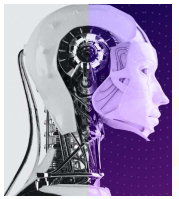
- Fill the knapsack with as much value in goods as possible (i.e., maximise "profit") **without exceeding the capacity** (as a constraint) – which items to take?



0	1	2	3	4	$f(s)$ profit		
1	1	0	0	1	\$8 (15 kg)	→	feasible solution
0	1	1	1	1	\$15 (8 kg)	→	feasible solution
1	0	0	1	0	~~\$14~~ (**16 kg**)	→	not a feasible solution
1	1	0	1	0	~~\$16~~ (**18 kg**)	→	not a feasible solution

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., $f'(s) = -2$:

    1 1 0 1 0: \$-2 (**18 kg**), 1 0 0 1 0: \$-2 (**16 kg**)
  - Distinguish the level of infeasibility of a solution with the penalty: e.g., $f'(s)$=min_profit/(2*(*total_weight*-**capacity**)):

    1 1 0 1 0: \$0.167 (**18 kg**), 1 0 0 1 0: \$0.5 (**16 kg**)

Evolutionary Algorithms for Constrained Parameter Optimization Problems Zbigniew Michalewicz and Marc Schoenauer: https://cs.adelaide.edu.au/~zbyszek/Papers/p30.pdf

Example Neighbourhood Relation Binary Representation

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

  Neighbourhood size: 8, Hamming distance: 1

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

Delta Evaluation for 0-1 Knapsack

weight of the object is ε and its profit is Δ

- 0 1 0 … 0 **0** 0… 1 1 $\rightarrow$ 0 1 0 … 0 **1** 0… 1 1

$f(s) = \Sigma$
totalWeight(s) = T

$f(s') = \Sigma + \Delta$
totalWeight(s') = $T + \varepsilon$

Example Neighbourhood Relation 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)

Example Neighbourhood Relation 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 Relation Permutation Representation II

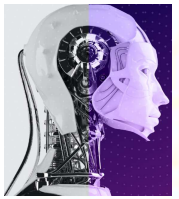
- Exchange operator: arbitrarily selected two entries are swapped
  - Example: **5** 4 3 **1** 2 → **1** 4 3 **5** 2

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

Mechanisms for Escaping from Local Optima I

- 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



- 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

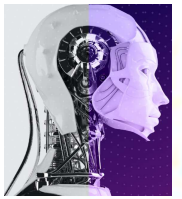
- Stop if
  - a fixed maximum number of iterations, or moves, objective function evaluations), or a fixed amount of CPU time is exceeded.
  - consecutive number of iterations since the last improvement in the best objective function value is larger than a specified number.
  - evidence can be given than 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)

Tabu Search

- Proposed independently by Glover and Hansen 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
  - 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.

Tabu Search Algorithm



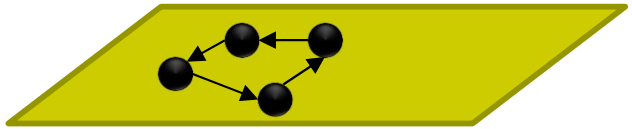
```
s = generateInitialSolution();
Repeat
    s' = findBestNeighbour(s);
    if (notTabuMove(tl, s') or aspirationCriteriaCheck(s'))
            s = s' ;
            s_best = updateBestSolution(s');
            updateTabuList(tl, s');
Until (termination conditions are satisfied)
return s_best
```

An Iteration of Tabu Search for a Minimisation Problem

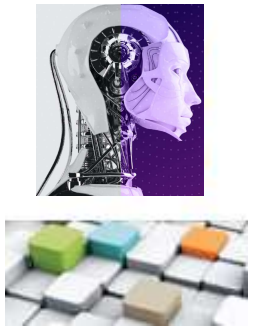
- Neighbourhood: 1-bit flip
- Tabu tenure: 2
- Current iteration (*ci*) is 1207
- Current solution is $s = 101100; f(s) = 43$
- Current tabu list content: <1,3> (tail)
  - This means that 1st bit was flipped two steps ago, while 3rd bit (c) was flipped in the immediately previous step.
- Consider all 1-bit flip neighbours which are not in the tabu list
  - $f(\mathbf{0}01100)$ **X** $tabu$
  - $f(1\mathbf{1}1100) = 17$
  - $f(10\mathbf{0}100)$ **X** $tabu$
  - $\underline{f(101\mathbf{0}00) = 9}$
  - $f(1011\mathbf{1}0) = 51$
  - $f(10110\mathbf{1}) = 61$
  - $s' \leftarrow 101000$
- 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)

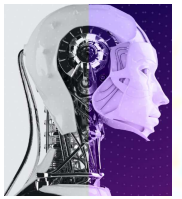


Evolutionary Algorithms – Evolutionary is Revolutionary

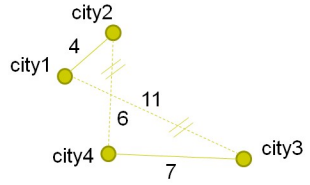


- Nature as a Problem Solver: 4.55 Billion years of evolution can't be wrong.
- Beauty-of-nature argument: Complexity achieved in *short* time in nature.
- Can we solve complex problems as quickly and reliably on a computer?

Evolutionary Algorithms (EAs): Terminology

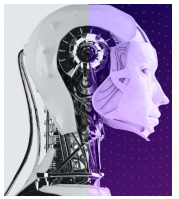


- EAs are population-based metaheuristics.
- EAs simulate natural evolution (Darwinian Evolution) of **individual** structures at the genetic level using the idea of ***survival of the fittest*** via processes of **selection**, **mutation**, and **reproduction (recombination)**
- An ***individual*** (***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 (near-)optimal solution

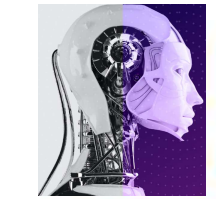
Types of Evolutionary Algorithms (EAs) and History



- **Genetic Algorithms** (evolves (bit) strings) ⇒ Turing 1948, Nils Aall Barricelli 1954, Bremermann 1962, Holland 1975
  - Memetic Algorithms ⇒ Moscato 1989
- **Evolutionary Programming** (evolves parameters of a program with a fixed structure) ⇒ Fogel, Owens, Walsh 1965
- **Evolution Strategies** (vectors of real numbers) ⇒ Rechenberg 1965
- **Genetic Programming** (evolves computer programs in tree form) ⇒ Koza 1992
  - **Grammatical Evolution** (evolves solutions wrt a specified grammar) ⇒ Ryan, Collins and O'Neill 1998
  - **Gene Expression Programming** (computer programs of different sizes are encoded in linear chromosomes of fixed length) ⇒ Ferreira 2001
- **Differential Evolution** (real valued optimisation) ⇒ Storn and Price 1997

...

A Generic Genetic Algorithm



- **First Generation**: Create the initial population of solutions (set of individuals) randomly and evaluate them.
- **Next Generations**: Repeat the following steps until termination
  - *Select* the fittest individuals for reproduction. (parents)
  - Create new solutions (offspring, children) – through *crossover* and *mutation* operations.
  - Evaluate the *fitness* of each new solution
  - *Replace* the least-fit solutions of the population with new solutions.

Initialisation – Binary Encoding

i	Chromosome	Fitness
1:	110000	3
2:	010100	3
3:	001100	2
4:	101110	1

- Assume *population size* is 4
- The individual/*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)).

Parent Selection

- Usually 2 parents (individuals/candidate solutions) are selected using the same method, which will go under the crossover operation ⇒ e.g., roulette wheel selection, *tournament selection,* rank selection, truncation selection, Boltzmann selection, etc.
- *Tournament selection* runs a number of "tournaments" among randomly chosen individuals (of tour size) selecting the one with best fitness at the end

Example – Tournament Selection

tour size = **3**, first parent

i	Chromosome	Fitness
1	110000	3
2	010100	3
3	001100	2
4	101110	1

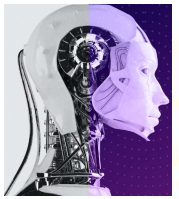
- Throw a random number between 1 and 4 (population size) for **3** times:
  - [3, 1, 2]
- Tournament selection chooses 3 individuals: #1, #2 and #3 at random, then individual#3 with the fitness of 2 is returned as the first parent

Example – Tournament Selection

tour size = **3**, second parent

i	Chromosome	Fitness	
1	110000	3	⇐
2	010100	3	
3	001100	2	⇐
4	101110	1	⇐

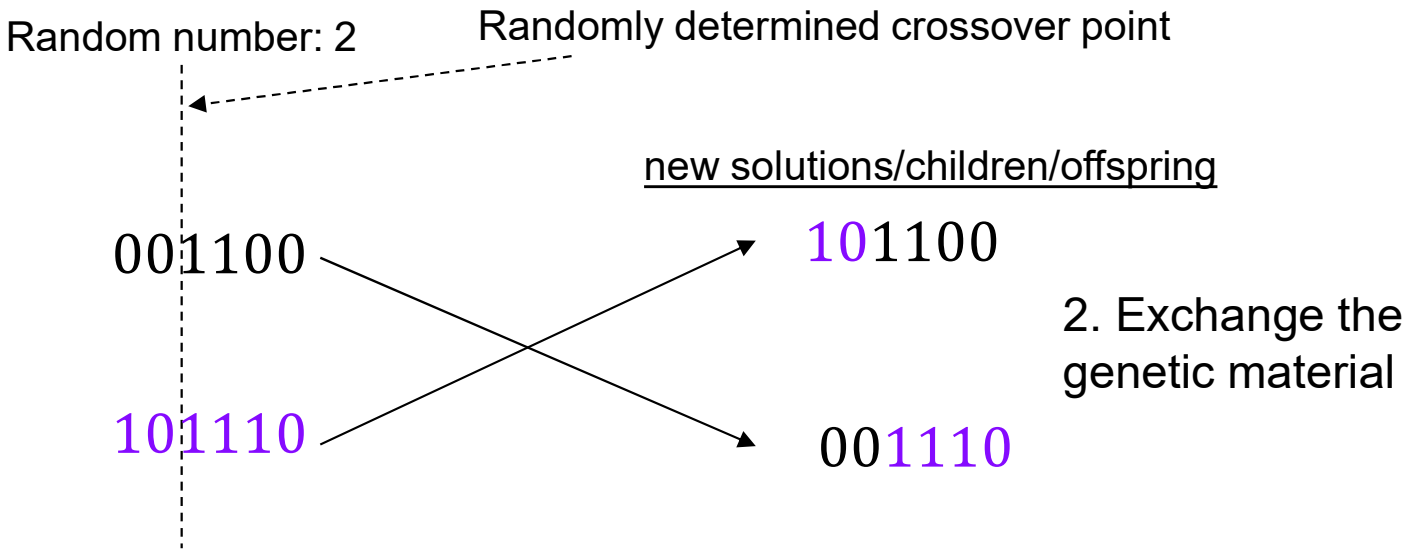
- Throw a random number between 1 and 4 (population size) for **3** times:
  - [4, 1, 3]
- Tournament selection chooses 3 individuals: #1, #3 and #4 at random, then individual#4 with the fitness of 1 is returned as the second parent



Crossover

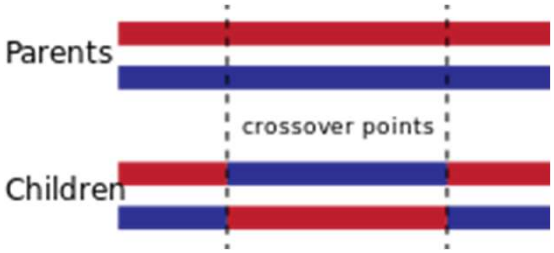
- Selected pairs/mates (*parents*) are recombined to form new individuals (candidate solutions/children/offspring) – exchange of genetic material
- Crossover is applied with a **crossover probability** p_c which in general is chosen close to 1.0

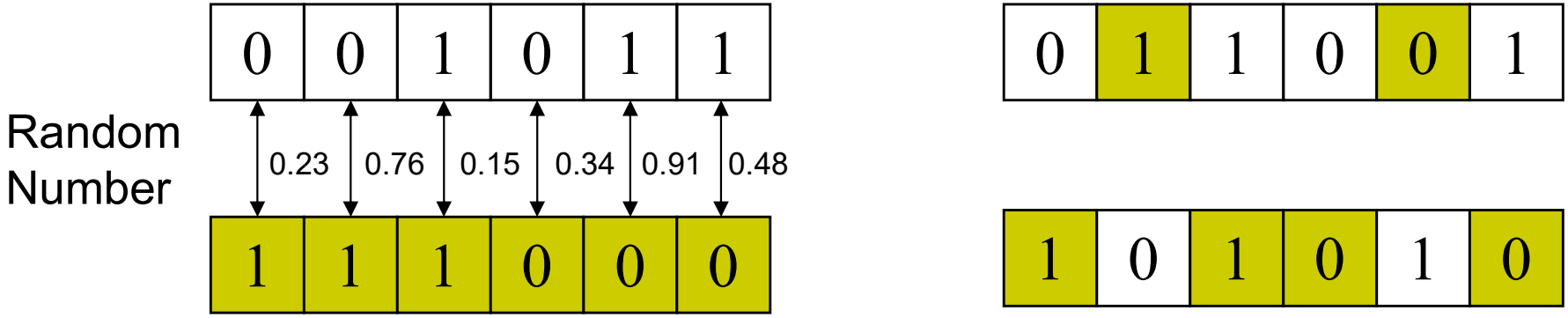
One Point Crossover (1PTX)



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.





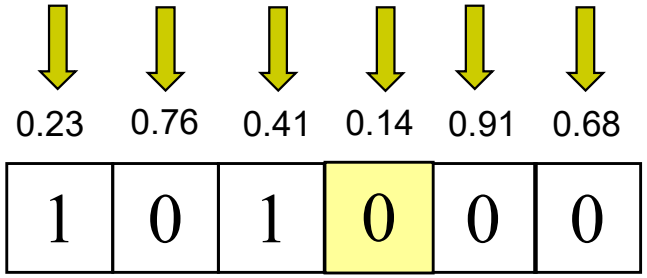
Mutation

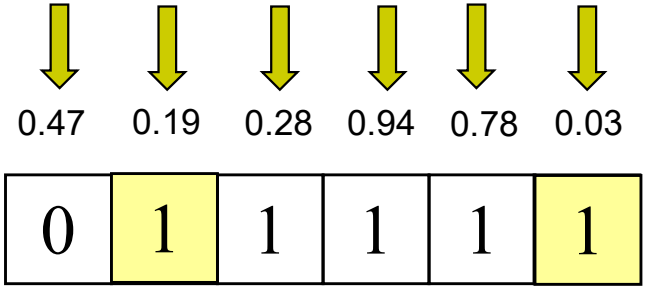
- Any offspring might be exposed to mutation
- Loop through all the alleles of all the individuals one by one, and if that allele is selected for mutation with a given probability p_m, you can either change it by a small amount or replace it with a new value
  - For binary representation mutation corresponds 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/chromosome length)* implies on average a single gene will be mutated for an individual.

Example – Mutation

Random numbers:

<0.23 0.76 0.41 0.14 0.91 0.68 0.47 0.19 0.28 0.94 0.78 0.03 …>



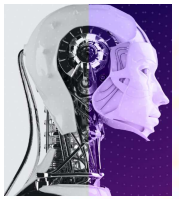


- A loop is performed on each individual
  - If a random value in [0,1) is < p_m =0.2 (1/5), then the allele value is flipped, otherwise kept same.

Replacement – Steady-State GA

- Two *o*ffspring replace two individuals from the old generation.
  - Method#1: two *o*ffspring replace two parents
  - Method#2: two *o*ffspring 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)

Replacement – Transgenerational GA Replacement (with elitism)



i	Chromosome	Fitness
1	001100	2
2	101110	1
3	101000	0
4	011111	0

New Population/ Generation

Form the new generation by

Selecting the best 4 individuals among both old population and offspring

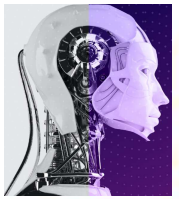
i	Chromosome	Fitness
1	100100	3
2	010100	3
3	101000	0
4	011111	0

Offspring

i	Chromosome	Fitness
1	110000	3
2	010100	3
3	001100	2
4	101110	1

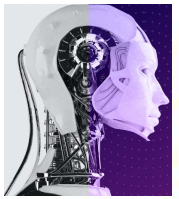
Old Population

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)
	• [Storn & Price1997]	Differential Evolution
	• [Hansen, 1998]	CMA-ES
Constructive	• [Dorigo, 1992]	Ant Colony Optimisation (ACO)
	• [Resende, 1995]	Greedy Randomized Adaptive Search Procedure (GRASP)

Metaheuristics

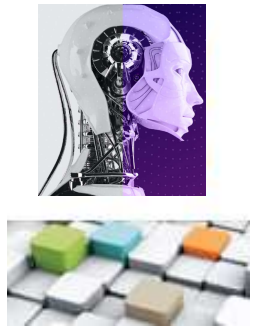


		Examples of Parameters
Local Search	SA	T_0 (initial temperature), α (cooling rate)
	TS	Tabu list size
	GLS	λ (intensification control), a (coefficient)
	ILS	mutation strength
	VNS	k_{min}, k_{max} (smallest, largest neighbourhood size)
Population-based	GA	population size, mutation probability
	GP	
	EC	
	MA	
	DE	scale factor, crossover rate, population size
	CMA-ES	population size, number of parents
Constructive	ACO	weight of pheromone, evaporation rate
	GRASP	restricted candidate list parameter

Parameter Setting

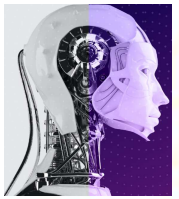
- **Parameter tuning**: Finding the best initial settings for a set of parameters before the search process starts (*off-line*). E.g., fixing the mutation strength in ILS, mutation probability in genetic algorithms, etc.
  - The initial parameter setting influences the performance of a metaheuristic
- **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

Parameter Tuning Methods



- Traditional approaches
  - Use of an arbitrary setting
  - Trial&error with settings based on intuition
  - Use of theoretical studies
  - A mixture of above
- Sequential tuning: fix parameter values successively
- Design of experiments (E.g., Taguchi method)
- Meta-optimisation: use a metaheuristic to obtain “optimal” parameter settings

(Automated) Parameter Tuning Methods



- Sampling methods
  - Efficient Global Optimisation (1998), Calibra (2006),…
- Screening methods
  - I/F-Race (2002) [download: http://iridia.ulb.ac.be/irace/],…
- Meta-optimisation
  - meta-GA (1986), linear GP (2005), Relevance Estimation and Value Calibration of Parameters (2006), ParamILS (2007) [download:http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/], Gender-based GA (2009),…
- Model-based
  - Sequential Parameter Optimization (2005) [download: https://cran.r-project.org/web/packages/SPOT/index.html], Sequential Kriging Optimisation (2006), Sequential Model-Based Algorithm Configuration (2010),…

The Art of Searching

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

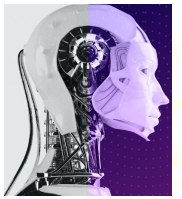
Summary

- There are various search paradigms and metaheuristics provide guidelines for heuristic optimisation based on those search paradigms
- There are three main classes of metaheuristics: local search metaheuristics, population based metaheuristics and constructive metaheuristics
- Move acceptance is a crucial component of local search metaheuristics
  - Simulated Annealing would be a good initial choice for solving an unseen problem
- Each component of a metaheuristic influences its performance: careful implementation and analysis required

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

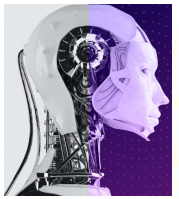
Metaphors and New Nature Inspired Metaheuristics



- “It is not acceptable to use a metaphor to promote an algorithm as interesting or novel just because the metaphor is interesting or novel for the author.” [3]
- The *Journal of Heuristics* fully endorses Sörensen’s view that metaphor-based “novel” methods should not be published if they cannot demonstrate a contribution to their field. [Policies on Heuristic Search]

[1] Dennis Weyland. A rigorous analysis of the harmony search algorithm: How the research community can be misled by a “novel” methodology, Int J Appl Metaheuristic Comput, 1 (2) (2010), pp. 50-60. [PDF1] [PDF2]
[2] Metaheuristics – the metaphor exposed, K. Sörensen (2013) [PDF]
[3] Camacho-Villalón, C.L., Dorigo, M. & Stützle, T. The intelligent water drops algorithm: why it cannot be considered a novel algorithm. *Swarm Intell* **13,** 173–192 (2019). [PDF]

Resources



1. Search methodologies: introductory tutorials in optimization and decision support techniques - Edmund Burke, Graham Kendall c2014 [copy found over the internet in PDF]
2. Stochastic local search: foundations and applications - Holger H. Hoos, Thomas Stützle 2005 [Public access to an old version]
3. Metaheuristics: From Design to Implementation, El-Ghazali Talbi, DOI: 10.1002/9780470496916, John Wiley, ISBN: 9780470278581 [PDF from ResearchGate] (this version is publically available now)
4. J. Swan, S. Adraensen, C. G. Johnson, A. Kheiri, F. Krawiec, J.J. Merelo, L. L. Minku, E. Özcan, G. L. Pappa, P. García-Sánchez, K. Sörensen, S. Voss, M. Wagner, D. R. White, Metaheuristics "In the Large", European Journal of Operational Research, Vol. 297, Issue 2, pp. 393-406, 2022, DOI:10.1016/j.ejor.2021.05.042 (available online [PDF]), to appear. [PDF]
5. J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, Recent Advances in Selection Hyper-heuristics, European Journal of Operational Research, vol. 285, no. 2, pp. 405-428, 2020, DOI:10.1016/j.ejor.2019.07.073 (invited review) (Open Access - available online [PDF]).

Q&A

Thank you.

Prof 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