Course: Natural Computing

*3. Computational and Statistical Aspects of Natural Computing

J. Michael Herrmann
School of Informatics, University of Edinburgh
michael.herrmann@ed.ac.uk, +44 131 6 517177

Problem solving

How to find a good solution to a given problem?

- Direct calculation, straight-forward recipe
- Solution by analogy, generalisation
- Iterative solution, continuous improvement
- Cartesian method: Divide and conquer
- Heuristics and meta-heuristic algorithms
- Trial and error, random guessing

specific, exact, reliable

$\Uparrow$

$\Downarrow$

general, sufficing, sloppy

General problem solving

Utility is measurable, and risk affects utility (Milton Friedman, 1948)

Utility theorem: If a utility satisfies a set of axioms, then this utility can be expressed by a function (John von Neumann and Oskar Morgenstern, 1945)

Utility hypothesis: Every problem can be encoded by a utility function whose maxima are the admissible solutions (see e.g. Sutton and Barto, 1999, 2017)

We will not discuss here whether the latter is true, but will assume instead that we have already an objective function or are able to construct one.

see also: D. Silver e.a. (2021) Reward is enough. *Artif. Intell*. 103535

Problem Solving as Minimisation of a Cost Function

Choosing the best option from some set of available alternatives

- Minimise energy, time, cost, risk, ...
- Maximise gains, acceptance, turnover, ...
- Averaging may be necessary (over what timescale?)
- Environmental dynamics may have effects as well
- Discrete cost (incorporating constraints):
    - admissible state: maximal gain
    - anything else: no gain
- Secondary costs for:
    - acquisition of domain knowledge, modelling, determining costs
    - testing alternatives
    - doing nothing

Problem Solving as Optimisation of an Objective Function

Objective function*:

- Cost (min)
- Energy (min)
- Risk (min)
- Quality (max)
- Fitness (max)

May be analytical, observational, relative, qualitative, compositional

*Note of caution:

We will sometimes aim at *minimisation*, sometimes at *maximisation*, sometimes we will speak of *optimisation* or the search for the *best* solution. It will usually be clear from context whether high or low values are to be preferred.

Problem Solving as Minimisation of a Cost Function

Example: Evolution of a walking machine

$$\begin{aligned}
\text{Cost} = & +\alpha && \times && \text{weight} \\
& -\beta && \times && \text{speed} \\
& -\gamma && \times && \text{(number of steps before falling over)} \\
& +\delta && \times && \text{(energy consumed)} \\
& -\varepsilon && \times && \text{(measure of similarity to human gait)}
\end{aligned}$$

- Costs $\rightarrow$ fitness function (low costs = high fitness) can be calculated for each candidate solution
- Use a set (population) of candidate solutions
- Evolution scheme for the candidate solutions to produce more of the good ones and discover better ones
- For different choices of α, β, γ, δ, ε the optimal solutions will be different

Meta-heuristic algorithms

- Similar to stochastic optimisation
- Iteratively trying to improve a possibly large set of candidate solutions
- Few or no assumptions about the problem (need to know what is a good solution)
- Usually finds good rather than optimal solutions
- Adaptable by a number of adjustable parameters
- On-line identification of the structural elements that can be composed into candidate solutions
- General problem: more "greedy" or more "rambling"? Exploitation or exploration?

http://en.wikipedia.org/wiki/Metaheuristic

A Selection of Topics from Natural Computation

Themes

- Evolutionary Computation
- Artificial immune systems
- Neural computation
- Amorphous computing
- Swarm intelligence
- Harmony search
- Cellular automata
- Artificial life
- Membrane computing
- Molecular computing
- Quantum computing

Methods

- Genetic algorithms
- Genetic programming
- Gravitational search
- Central force optimisation
- Particle swarm optimisation
- Evolutionary algorithms
- Spiral dynamics algorithm
- Chemical reaction optimisation
- Artificial physics optimisation
- Ant colony optimisation
- Differential Evolution

Comparison of GA, PSO and ACO

	GA	ACO	EA	PSO	DE
search space	discrete	discrete	continuous	continuous	continuous
repre-sentation	strings	pheromone trail	positions	positions and velocities	positions and differences
stochasticity	stoch. op-erators	sampling a distribution	random mutations	modulation of two forces	modulation of one force
timing of interaction	across generations	across generations	n.a.	when new best was found	across generations
interaction	by crossover	pheromone	none	forces to bests	differences
evolutionary drive	selection	evaporation	selection	update of bests	acceptance
distribution of solu-tions	repres. by population	explicit: pro-bability rule	explicit by mutations	represented by population	represented by population
parameters	p_m, p_c	ρ, α, β, N	C, λ, μ	$\alpha_1, \alpha_2, \omega, N$	F, p, N
build.blocks	schemas	none	none	[sparse]	[differences]

Similarities: population-based (SA, HC aren't), stochastic, termination, can degenerate, hybridisable, inexact, global optimisation, ...

Introduction

- Literature on metaheuristic algorithms is largely organised by algorithm or groups of algorithms, rather than by similarities of features and challenges across algorithms
- Problem-dependent details of the implementation and tricks of the trade are not easy to learn (and not easy to teach, sorry).
- For simple Matlab programs see Xin-She Yang's book, see also chapter 5 in Mitchell and E. Talbi (2009) Metaheuristics: From design to implementation.
- In this unit, we will ask
  - Which metaheuristic algorithm (MHO) to use for a given problem?
  - How to test a MHO algorithm using benchmark problems and scaling analysis?
  - How to determine parameter values?

For what problems should I use metaheuristics?

A typical case for metaheuristics is due to costly fitness evaluations, i.e. when a good solution is to be found with few fitness evaluations (FEs)

- Gradients require several FEs at nearby points (with noise: at many points)
- A *model* can help reducing fitness evaluations (e.g. a linear model can be fitted from a few points),
- however the model may not fit well
- MHO algorithms are also “models”, but their properties are often obscure
- Population-based algorithms often have redundancies

For what problems should I use metaheuristics?

There are other indications for the use of MHO algorithms:

- If no exact, efficient algorithm is available
- In case of discontinuous, nonlinear, ill-conditioned, noisy, multimodal, nonsmooth, nonseparable problems or problem with complex constraints, i.e. where gradients do not work in practice.
- ... or if gradients can be adapted to work, but this requires problem-specific knowledge that is not available
- Optimisation by simulation. Black-box functions (Note that Monte-Carlo is in principle also a metaheuristics)
- Support of exact methods or as meta-optimiser: Setting up parameter for other algorithms, e.g. to find the structure of a neural network

see E. Talbi (2009) Metaheuristics: From design to implementation.

Examples of problem cases for MHO

- AI, agent-based systems, planning ("Potato-in-the-exhaust")
- In linear programming for an $n \times n$ assignment problem, with $2n$ linear and n^2 nonnegativity constraints, the search polytope has $n!$ vertices.
- Gradient descent may fail for strongly heterogeneous parameters (Natural gradient may help but has a cost).

see E. Talbi (2009) Metaheuristics: From design to implementation.

Before deciding about an MHO, ask: What is the goal?

- Solve a problem (this is the typical case)
    - knowing enough about the problem to know when it is solved or to have an intuition that improvements are possible
    - not knowing whether the problem is solvable or the existing solution is improvable at all
- Study a (biological) system by reproducing it function in a computational system
- Design a new algorithm
- Finish coursework
- Study an existing algorithm to uncover its function
- Win an optimisation competition

Which metaheuristic algorithm to use?

General answer: The choice of algorithm is justified by performance.

What is known about the problem?

- discrete or continuous
- multi-modal
- noisy
- heterogeneous
- many local optima
- ...

What effort to be spent?

- creating a new algorithm
- adapting an existing algorithm
- parameter adaptation
- hybridise algorithm
- many FEs
- ...

Properties of representations

Not all of the following properties are required, but MHO algorithms work better if they are present at least to some extent.

- Completeness: The representation must not exclude any solutions associated with the problem. Note that larger search spaces almost always lead to higher complexity
- Connexity: A search path must exist between any two solutions of the search space. Note that it is still possible that the algorithm assigns a very low probability to the path to the global optimum
- Efficiency: The time and space complexities of the operators must be low. Note that this can be guaranteed only if the problem is very well understood
- Topography: Similar solutions should be represented by similar representations, i.e. the metrics in the solution space (target space) should be compatible with the representational space
- Compositionality: Parts of the solution should correspond to parts of the representation

see E. Talbi (2009) Metaheuristics: From design to implementation.

How to deal with constraints?

- Reject produced by the algorithm if constraints are violated
- Repair or edit solutions (by a local heuristics), choose operators that transform admissible solutions into admissible solutions
- Penalise violations
  - number of violated constraints
  - repairing cost
- Adaptive strategies: at first penalise, later reject
- Check also about hard and soft constraints
- Discreteness can be seen as a constraint in a continuous algorithm

see E. Talbi (2009) Metaheuristics: From design to implementation.

How to test a MHO algorithm using benchmark problems?

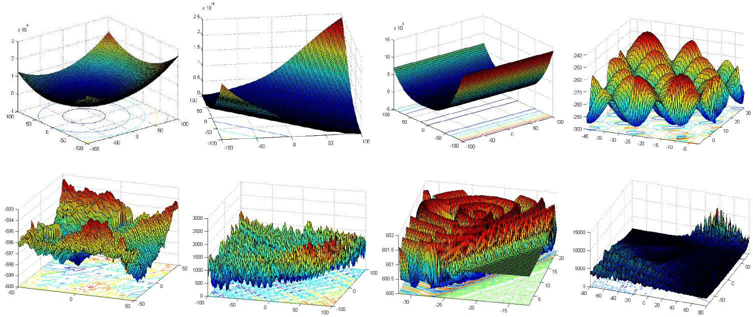
Benchmarks: Fitness at global optimum usually known

Which benchmark problems to use?

- For comparison and thorough test, use the same and the same number of problems that were used in the work you are comparing to
- During design of the algorithm, use problems that seems to be beneficial to learn more about the approach. Continue to serious testing later.

CEC benchmark problems

- Identical optimum ("0") and domain of the functions: $[-100, 100]^d$
- Dimensionality can be chosen $d = 2, 5, 10, 50$
- Random shifts and rotation
- Limited number of fitness evaluations (20,000 FEs)



Liang, J. J. et al. Problem definitions and evaluation criteria for the CEC 2013. Nanyang TU, Singapore, Technical Report. (see also CEC 2014 and CEC 2015)

Evaluation at CEC

- Record fitness after (0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)*MaxFEs
- Present the best, worst, mean, median and standard deviation values of function error values for 51 runs
- Round results in a reasonable way; error values smaller than 10^{-8} are taken as zero
- Include information about
    - Parameters the were adjusted incl. dynamic ranges
    - how the parameters were adjusted
    - Estimated cost of parameter tuning in terms of number of FEs
    - Actual parameter values used

Liang, J. J. et al. Problem definitions and evaluation criteria for the CEC 2013. Nanyang TU, Singapore, Technical Report.

How does the performance depend on the parameters?

- What is a parameter? Sometimes the dynamical variables of the algorithm are called parameters, namely the parameters that estimate the solution, then the "parameters" of the algorithm are called hyperparameters
- Good algorithms have lower parameter sensitivity, but scaling of the problem usually increases parameter sensitivity
- Parameter values can often be interred by theory (but note that theories make assumptions that may not be valid in practical cases)
- Perform parameter scans
  - there are usually few main parameters
  - help to understand the algorithm
  - may be replaced by higher-order search algorithm (hyperheuristics)

Presentation of MHO results

- Use commonly used benchmarks
- Show execution times (on a specific machine) and number of executions
- Show convergence behaviour
- Include comparisons with other methods
- Show, both for objective function and runtime, best and worst results, the average and the standard deviation, and the median
- Include a statistical test that the main hypotheses are valid (or that the hypothesis can be rejected that your algorithm is indistinguishable from a competitor)
- Make your source code available

Osaba et al. (2016) Good Practice Proposal for the Implementation, Presentation, and Comparison of Metaheuristics for Solving Routing Problems

Termination

External criteria:

- Prescribed termination criterion, e.g. if
  - error acceptable in application
  - significantly better than competing method
- Limited runtime on a given machine, e.g. results needed overnight
- Limited number of fitness evaluations, e.g. if cost involved
- Limited number of generations or iteration step

Intrinsic criteria: continuation, restart or parameter adaptation can be advisable:

- Saturation (fitness stopped to improve (maybe plateau?))
- Stagnation (collapse, degeneration, loss of diversity)
- Divergence (numerical, all particles outside search space, or, for individuals of variable complexity, by “bloat”)

How to estimate required runtime?

- Many algorithms find the globally optimal solution with probability 1 in exponential time. E.g. SA with $t > 0$, ACO with $\tau_{\min} > 0$ or GA with $p_m > 0$.
- There is no general non-exponential upper bound known for the run-time for NP-complete problems
- For lower bounds on the runtime, check whether there are enough iterations that every bit can be expected to be mutated at least a few times or that the particles are able to move to every place in the search space
- Practically, check whether
  - *innovations* (fitness improvements) still occur
  - restarts lead to similar results for the same runtime
  - balance between exploration and exploitation is kept (or suitably shifted from exploration to exploitation)

How to evaluate the results statistically?

Possible outcomes (assuming global optimum is known)

- finds global optimum
    - how often within multiple runs
    - after what run time / FE / iterations
- finds fitness value off global optimum
    - average, std. dev.
    - metrics
    - dealing with errors (median)
- no result (fails, diverges, inadmissible result, ...)

Mean or best result on a number of runs?

- *mean* $\pm$ *standard deviation* is the representation of choice
- *mean* $-$ *standard deviation* can be negative even for a positive random variable: use one-sided std. dev.
- best result over a number of runs is
    - overly optimistic
    - probably not robust
    - may be an outlier
- *variance* or even *mean* may not exist (e.g. for certain probability distributions) or does not make sense, e.g., if for some random initialisations the algorithm diverges while it performs well for others). Often the median can be used instead.
- Best result can be useful in applications, if sufficiently robust

Birattari & Dorigo (2007) How to assess and report the performance of a stochastic algorithm on a benchmark problem: *mean* or *best* result on a number of runs? *Optimization Letters* **1**:309–311.

Hypothesis testing

- E.g. your hypothesis is
  Algorithm A_1 is better than algorithm A_2 on problem set $\{P_i\}$ under the condition that both algorithms are optimally adapted to the problem set.
- Perform many runs with both algorithms, calculate standard deviation and check by how many standard deviations the mean performances differ
- Consider also to use statistical tests (p values, F test)

Scaling analysis

Distinguish between different forms of scaling

- Scaling of performance with problems size or dimensionality (complexity)
- Scaling w.r.t. to termination criterion (precision)
- Scaling of population (populations often quite small)

Warning: MHO algorithms sometimes scale irregularly, i.e. they may scale well for medium problem sizes, but are exponential at larger sizes (for exponential or NP-complete problems)

Parallel implementation

- Cooperative multiple search threads lead to more robust and more effective implementations
- Population-based MHO algorithms are usually easily parallelis able. Almost linear speed-up possible if fitness evaluations are time-consuming, if algorithms are often restarted, parameters are scanned, or obviously for statistical evaluation. In some cases, the algorithms have to be modified to speed-up almost linearly, e.g.:
  - Interestingly, first attempt on parallelisation were made on SA, where either partitions (e.g. in a TSP) were considered or a **multiple-walk** strategy was introduced: after an number of parallel steps a winner is taken from which all processes then continue
  - Also ACO is not trivially parallelisable because only one pheromone trail exists, but also here delay global updates of the pheromone can be delayed for a few steps.
- Recently, GPU-based algorithms have been tested.
- Cung et al (2001) Strategies for the parallel implementation of metaheuristics.

Parallel Models for Metaheuristics

Modes of parallelisation

- Solution-based: parallelisation w.r.t. dimensions of the search space
- Iteration-Level: delayed synchronisation, island algorithm
- Algorithm-based: Hyper-heuristic that chooses among several metaheuristics of among several versions of one metaheuristic

Hyperheuristics/Memetic algorithms

- Cooperation and coevolution of different metaheuristics, e.g.
  - exploration (e.g. PSO)
  - exploitation (e.g. SA)
  - guiding, monitoring (e.g. Taboo search)
  - controlling (parameter adaptation by higher-order MHO)
  - adaptation (coordination of the components)
- Requires some effort and includes many design decisions which will be more efficient if based on domain-knowledge.

Conclusion

- Also in MHO, best results can be expected if domain knowledge or problem-specific knowledge is available
- In other cases, it is still possible to obtain good results with some effort on
    - efficient representation
    - algorithm selection
    - parameter selection
    - statistical evaluation
- There are attempts in current MH research to construct algorithms that adapt or select algorithms for given problems
- Later in this course, we will return to many of the aspects mentioned in this unit

Final question ;-)

Why do MHO algorithms have funny animal names?

- Mathematicians try *not to leave traces*, i.e. their results tend to be exactly correct, but it is rarely clear how one would come up with the idea. Now, if a method is not exactly exact, then the inspiration appears like a good excuse
- As the animal behaviour seems obvious, the end-users may fall prey to the illusion that they understand essentially (although actually merely metaphorically) the workings of the algorithm
- Historical reason: It started all with evolutionary and genetic algorithms, which set an early connection to the animal kingdom
- Animal names are mnemotechnically preferable to complex acronyms: e.g. "$(\lambda^k, \mu^k)^n - \mathrm{ACO}/\mathrm{DE} - \mathrm{BFGS}$" vs. *kitten algorithm* (inspired by kitten playing with balls of wool)