Single Solution Stochastic Metaheuristics

Deterministic single-solution metaheuristic

- Deterministic Algorithms look into the neighbourhood of a solution and follow the "gradient" to converge to the closest local minimum.
- These algorithms are very efficient to quickly reach the bottom of the basin of attraction.
- In the case of highly multi-modal function these algorithms are in general not so efficient.

For these reasons some of them are also referred to as Local Searchers (LS)

Some algorithms are better for global search, others for local, it also depends on how their parameters are tuned.

Local VS Global

- Most deterministic algorithms are likely to converge to a near local optimum that is uninteresting in many real world applications.
- It is better a sub-optimal solution with a non-null gradient but a high quality rather that a null gradient solution with a low quality.
- *It is better to be approximately right than exactly wrong!*

Stochastic search

- In order to overcome this problem already in 50's and 60's some Stochastic Searchers were designed.
- The main idea is to reach the global minimum by searching not in the neighbourhood (gradient) but in all the decision space and just making pairwise comparisons.

"Trivial" stochastic searches

from 50s.

Algorithm 1 (random point)

$\mathbf{x_{cb}} \leftarrow$ initial guess $\quad \triangleright$ $\mathbf{x_{cb}}$: current best
while condition on budget **do**
  $\mathbf{x_{new}} \leftarrow$random point $\in D$
  if $f(\mathbf{x_{new}}) < f(\mathbf{x_{cb}})$ **then**
    $\mathbf{x_{cb}} \leftarrow \mathbf{x_{new}}$
  end if
end while
Output $\mathbf{x_{cb}}$

- Global search ✓
- Local search ✗

- Subsequently modified into:
- Makes use of a perturbation vector to move the current solution within D.

Algorithm 2 (perturbation)

$\mathbf{x_{cb}} \leftarrow$ initial guess
while condition on budget **do**
  $\mathbf{x_{new}} \leftarrow \mathbf{x_{cb}} + \mathbf{x_{random}} \in \mathbb{R}$
  if $f(\mathbf{x_{new}}) < f(\mathbf{x_{cb}})$ **then**
    $\mathbf{x_{cb}} \leftarrow \mathbf{x_{new}}$
  end if
end while
Output $\mathbf{x_{cb}}$

Simulated annealing
[Kirkpatrick et al., 1983]

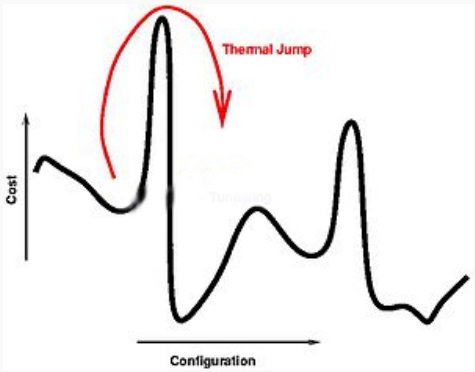
- By means of an analogy with a physical phenomenon, the stochastic search is slightly modified (a modification of the Metropolis Algorithm).
- Originally designed for combinatorial problems, can be adapted to real-valued optimisation by using the search logic of Algorithm 1 and 2.
- Improves upon trivial stochastic searches by embedding a mechanism to avoid getting caught in local minima (*jump!*).
- Although, not very robust, this algorithm can be rather efficient for some applications in detecting the global optimum.

Simulated annealing
Implementation details:

Simulated Annealing pseudo-code

$\mathbf{x_{cb}} \leftarrow$ initial guess$\in D$
$T \leftarrow$initial temperature ▷ e.g. equal to 1
while condition on budget **do** ▷ or T_{min} has been reached!
  $\mathbf{x_{new}} \leftarrow neighborSolution(\mathbf{x_{cb}})$ ▷ e.g. as in Algorithm 2
  if $\mathcal{U}(0,1) < e^{\frac{f(\mathbf{x_{cb}})-f(\mathbf{x_{new}})}{T}}$ **then** ▷ $\mathcal{U}(0,1)$: uniform random number $\in [0,1)$
    $\mathbf{x_{cb}} \leftarrow \mathbf{x_{new}}$
  end if
  reduce[1] T ▷ Typically $T \leftarrow \alpha T$, with $\alpha = 0.8, 0.99$
end while
Output $\mathbf{x_{cb}}$

- $e^{\frac{f(\mathbf{x_{cb}})-f(\mathbf{x_{new}})}{T}} \in [1,\infty] \Rightarrow$ *move to the better solution* $\mathbf{x_{new}}$*!*
- $f(\mathbf{x_{new}}) > f(\mathbf{x_{cb}}) \Rightarrow e^{\frac{f(\mathbf{x_{cb}})-f(\mathbf{x_{new}})}{T}} \in (0,1) =$*probability of accepting a worse solution!*$\Rightarrow$ *jump to* $\mathbf{x_{new}}$*!*



[1] Do not allow $t < t_{min}$: you have a finite amount of numerical places and could end up with $T = 0$!

Acceptance probability function $e^{\frac{f(\mathbf{x_{cb}})-f(\mathbf{x_{new}})}{T}}$:

working principle.

This function "behaves" as follows:

- is always ≥ 1 when the new point is better than the old one! ✓
- Gets small if the new solution is worse and too far from the old one! ✓

the algorithm is more likely to accept not too long jumps

we need to jump out from a local minimum, not to go too far!

- Decreases together with the temperature! ✓

Exploration (global search) and exploitation (local search)

We need to jump early on (high T), in order to explore D, and stay close to the potential global optimum towards the end of this process (T is low), to refine it

Simulated annealing:
THE METAPHOR.

Just out of curiosity...

... the working principle of simulated annealing draws its inspiration from metalworking. Throw in a constant in the Acceptance Probability function and it describes the embodied energy of metal particles as they are cooled slowly, after being subjected to high heat. This process allows the electrons to move from a random configuration to one with a very low embodied energy. Computer scientists borrow the annealing equation to help them move from a random solution to one with a very low (global minimum) functional value.

Solis-Wets algorithm
[Solis and Wets, 1981]

- Randomised greedy hill-climber;
- it generates a new solution by stochastic perturbation;
- the perturbation vector $\mathbf{b} + \mathcal{N}(0, \rho\mathbf{I})$ is used to perform:
  - a step forward from $\mathbf{x_{sw}}$;
    $(\mathbf{x_{new}} \leftarrow \mathbf{x_{sw}} + \mathbf{b} + \mathcal{N}(0, \rho\mathbf{I}))$
  - if $f(\mathbf{x_{new}}) \geq f(\mathbf{x_{sw}})$, a step backward;
    $(\mathbf{x_{new}} \leftarrow \mathbf{x_{sw}} - (\mathbf{b} + \mathcal{N}(0, \rho\mathbf{I})))$
- consecutive successes and failures are registered to:
  - expand the exploratory radius ρ;
  - contract the exploratory radius ρ.

It can be seen as a random search within a neighbourhood of $\mathbf{x_{sw}}$: $\mathcal{N}(0, \rho\mathbf{I})$ is a Gaussian distribution centred in a slightly shifted version of $\mathbf{x_{sw}}$ and variance ρ!

-local search ✓-can avoid local minima $\sim$ - scalability ✗

Solis-Wets algorithm

Implementation details:

Algorithm 3 Solis-Wets Method

$\mathbf{x_{sw}} \leftarrow$initial guess $\in D$
$\rho \leftarrow 1$
$\#S \leftarrow 0$ ▷ Number of consecutive successes
$\#F \leftarrow 0$ ▷ Number of consecutive failures
$\mathbf{b} \leftarrow \emptyset$
while *condition on budget* **do** ▷ or ρ smaller than a threshold (accuracy)
  $\mathbf{x_{new}} \leftarrow \mathbf{x_{sw}} + \mathbf{b} + \mathcal{N}(0, \rho \mathbf{I})$ ▷ Step forward
  if $f(\mathbf{x_{new}}) < f(\mathbf{x_{sw}})$ **then**
    $\mathbf{b} \leftarrow 0.4(\mathbf{x_{new}} - \mathbf{x_{sw}}) + 0.2\mathbf{b}$ ▷ Update bias
    $\mathbf{x_{sw}} \leftarrow \mathbf{x_{new}}$
    $\#S \leftarrow \#S + 1$ ▷ Register a success. . .
    $\#F \leftarrow 0$ ▷ . . . and reset failures
  else if $f(2\mathbf{x_{sw}} - \mathbf{x_{new}}) < f(\mathbf{x_{sw}})$ **then** ▷ Step backward
    $\mathbf{b} \leftarrow \mathbf{b} - 0.4(\mathbf{x_{new}} - \mathbf{x_{sw}})$ ▷ Update bias
    $\mathbf{x_{sw}} \leftarrow 2\mathbf{x_{sw}} - \xi$
    $\#S \leftarrow \#S + 1$ ▷ Register a success. . .
    $\#F \leftarrow 0$ ▷ . . . and reset failures
  else
    $\#S \leftarrow 0$ ▷ Reset successes. . .
    $\#F \leftarrow \#F + 1$ ▷ . . . and register a failures
  end if
  if $\#S > S_{max}$ **then** ▷ Threshold S_{max}: 5 in [Solis and Wets, 1981]
    $\rho \leftarrow \alpha_e \cdot \rho$ ▷ Expantion factor α_e: 2 in [Solis and Wets, 1981]
  else if $\#F > F_{max}$ **then** ▷ Threshold F_{max}: 3 in [Solis and Wets, 1981]
    $\rho \leftarrow \alpha_c \cdot \rho$ ▷ Contraction factor α_c: 0.5 in [Solis and Wets, 1981]
  end if
end while
Output $\mathbf{x_{sw}}$

Considerations

- Solis-Wets performs a move along all the axes (diagonal across the search space).
- ρ is the same $\forall$ the design variables $\Rightarrow$ in $\mathbb{R}^2$ the Gaussian distribution has a circular section, and does not change shape, only size!
- The covariance matrix remains diagonal $\Rightarrow$ time and space complexity is linear: Solis-Wets id much "lighter" than Rosenbrock.
- The Simulated Annealing is characterised essentially by the selection of the new solution.
- Solis-Wets Algorithm is characterised by the search strategy (perturbation by means of a normal distribution).

Abstract interpretation of single solution algorithms

Single-Solution Optimisation Algorithms are all characterized by the same scheme:

- generate an initial base point and iteratively:
  - apply a logic to compute a trial solution;
  - apply a logic to select the new base point.

Algorithm design
(single solution)

- We should not consider algorithms as closed boxes;
- If we consider the algorithm as a tuple of the two aforementioned steps, an algorithm can be designed (e.g. a generation of trial according to Solis-Wets and a selection mechanism like in simulated annealing).

Algorithms and kitchen

- The process of designing an algorithm is similar to cooking, where the algorithmic designer is a chef;
- The designer selects the appropriate operators and combine them;
- Not all the choices of operators are meaningful (imagine a dish composed of only salt and sugar);
- The correct choice of "ingredient" is not enough to make a good algorithm as it is important how they are combined and their quantities (parameter setting);

An Introduction to Population Based Algorithms

Why a population?

- Population-based algorithms do not store only a solution but use a list of solutions for searching towards the optimum.
- The generation of a trial point is the generation of a trial solution. This operation can be performed by using a single population member to generate each trial solution or by using multiple population members.
- The selection of a new base point is the selection of a new population.

Advantages of a population

- The main argument for using a population is that the decision space is better covered, and different solutions may explore from different perspectives.
- The solutions interact and support each other.

According to [Prügel-Bennett, 2010]:

- the initial random sampling for generating the population performs an exploratory stochastic search of the space;
- the population acts as a filter which averages small spikes in the landscape (e.g. presence of noise);
- having multiple solutions improves the stability of the algorithm, i.e. the standard deviation of the final values is reasonably small.

Macro-families

Amongst a myriad of options, we can distinguish two main families:

- Evolutionary Algorithms.
- Swarm Intelligence Algorithms.

An Introduction to Population Based Algorithms

Evolutionary algorithms: historical notes.

The idea of exploiting evolution as inspiration for designing (optimisation) algorithms dates back to 1948 with Alan Turing.

However, the first implementations of "evolutionary approaches" appeared only later on in the '60s [Bremermann, 1962]. Below, the most important achievements.

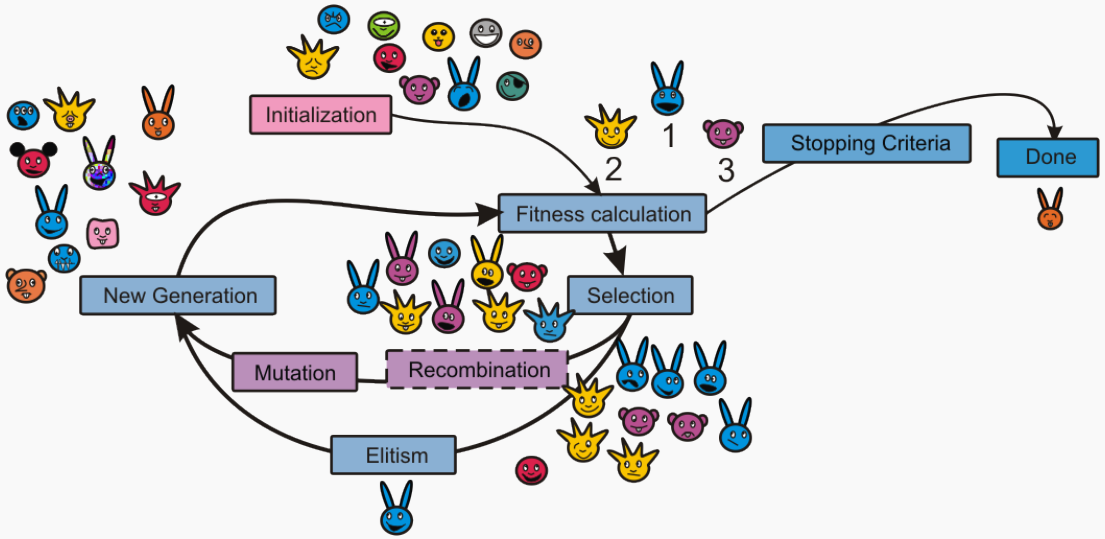
- **Fogel Owens (USA, 1964)**: *Evolutionary Programming (EP)*
  [Fogel et al., 1964], see also [Fogel et al., 1966]
- **Holland (USA, 1975)**: *Genetic Algorithm (GA)*
  [Holland, 1975]
- **Rechenberg Schwefel (Germany, 1971)**: *Evolution Strategies (ES)*
  [Rechenberg, 1971]
- Koza (USA, 1990): *Genetic Programming (GP)*
  [Koza, 1990], see also [Koza, 1992b] and [Koza, 1992a]
- **Moscato (Argentina-USA 1989)**: *Memetic Algorithms (MA)*
  [Moscato, 1989]

Darwinian evolution

- All environments have finite resources (eating),
- life forms have basic instinct/ life cycles geared towards reproduction;
- therefore some kind of selection is inevitable;
- those individuals that compete for the resources most effectively have increased chance of reproduction.

Darwinian evolution
in optimisation algorithms:

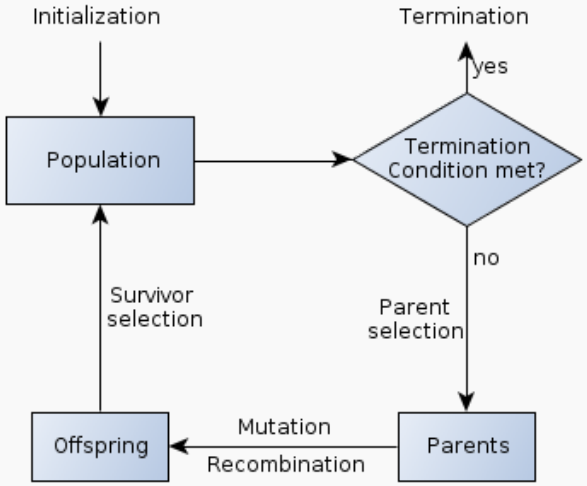
- Darwin thus states that at each individual (i.e. genotype) has an associated fitness which characterises the quality of the genotype and which let one individual winning the competition with the others;
- this logic encourages the best genotypes being transmitted to the offspring and thus over the generations to have fitter and fitter individuals;
- therefore this logic can be encoded in a computer program for solving optimisation problems.



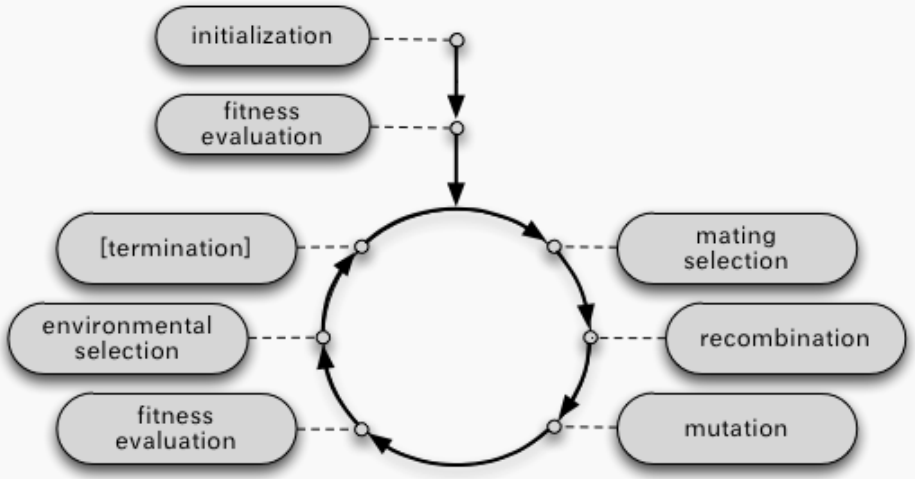
Evolutionary algorithms: general scheme.

Evolutionary Algorithms pseudo-code

INTITIALISE population with random candidate solutions
EVALUATE each solution
while *condition on budget* **do** ▷ or fitness value!
  SELECT parents ▷ 2 or multiple solutions
  RECOMBINE parents ▷ 2 or some of them
  MUTATE the resulting offspring
  EVALUATE new candidates
  SELECT individual for next generation
end while
Output best individual of the final population



Solutions cycle



Selection and variation operators cycle

Representation

- A solution is an individual and the corresponding vector (array) is referred to as chromosome (e.g. a bridge is an individual and the set of design parameters encoded in a vector compose the chromosome):
  - the chromosome can be a binary sequence,
  - a binary sequence encoding a real number,
  - (we deal only with) a floating point sequence of real numbers.
- The objective function is also named fitness of the problem.
- An individual that outperforms another is a solution characterised by a lower value of the objective function.

Parent selection mechanism

- Assigns probabilities to individuals acting as parents depending on their fitnesses;
- It thus selects the individuals undergoing recombination by means of a (***usually***) probabilistic logic:
  - high quality solutions more likely to become parents than low quality,
  - even worst in current population usually has non-zero probability of becoming a parent.
- This stochastic nature can aid escape from local optima (as for Simulated Annealing).

Variation operators

- Role is to generate new candidate (trial) solutions.
- Usually divided into two types according to their *arity* (number of inputs):
  - Arity 1 : *mutation* operators;
  - Arity >1 : recombination operators;
  - Arity = 2 typically called *crossover*.

Recombination

- It selects the individuals which are going to survive for the subsequent generation i.e. *it selects the individuals for the new population*.
- It is often deterministic e.g.:
  - **Age based**: make as many offspring as parents and delete all parents;
  - **Fitness based**: e.g. the so called "plus" strategy that ranks parents+offspring and take best (elitism).
  - **Mixed** (age and fitness): the so called "comma" strategy, that replaces the old population with the best individual previously generated.
- Stochastic methods can be used, e.g.:
  - round robin;
  - roulette wheel tournament, etc.

Mutation

- Acts on one genotype and delivers another.
- Contains some randomness which is essential and differentiates it from other unary heuristic operators.
- The importance and the efficiency of the mutation strictly depends on the problem.
- Has the role of "giving a further chance to the evolution" to detect new promising search directions.

Exploration and exploitation

No rigorous terms but very important ideas!

Properties of algorithmic components

- *Exploration*: capability of detecting new solutions in untested areas of the decision space.
- *Exploitation*: capability of using the available solutions (and search directions) and thus finishing testing already detected areas.

Balance between exploration and exploitation is key

Mutation can be considered explorative, Recombination exploitative. Algorithmic components are never entirely explorative or exploitative. The success of an optimisation algorithm strictly depends on the balance between exploration and exploitation!

Laboratory and participation work:

- Implement a Simulated Annealing and run it over the four problems (De Jong, Rastrigin, Schwefel, Michalewicz) in 10D, 30D and 50D, with toroidal handling of the bounds. Performs at least 30 runs, and make use of a computational budget equal to $5000 * D$ fitness functional calls.
- Try a different implementations of *neighborSolution*($\mathbf{x_{cb}}$) and different temperature update rules. (e.g. see some options HERE), in order to improve upon your previous results!

Discussion board

In the next slides you can see a detailed pseudo-code for two popular optimisers: Newton Method and Simultaneous perturbation stochastic approximation (SPSA) [Spall, 1987]. Beside the papers, plenty of material can be easily found on-line, and a concise but complete description is provided in [Caraffini, 2014].

- What kind of single solution optimiser are Newton and SPSA?
- What do the two algorithms have in common? What are the similarities?
- What are the most discriminant differences between the two?
- which one is, in your opinion, preferable over noisy fitness functions?
- Do you reckon they perform netter over multi-modal or mono-modal problems?

share your answers in the discussion boards.

Discussion board:
psudo-codes

Newton's Method

$k \leftarrow 0$
$\mathbf{x_0} \leftarrow$initial guess
while $\| \nabla f(\mathbf{x_k}) \|$ or $\|\mathbf{x_k} - \mathbf{x_{k-1}}\| \backsimeq 0$ **do**
  $\mathbf{N_k} \leftarrow \nabla f(\mathbf{x_k}) \mathbf{H_f}^{-1}(\mathbf{x_k})$
  $\mathbf{x_{k+1}} \leftarrow -\alpha \mathbf{N_k} + \mathbf{x_k}$
  $k \leftarrow k + 1$
end while
Output $\mathbf{x_k}$

SPSA

$k \leftarrow 0$
$\boldsymbol{\theta_k} \leftarrow \mathbf{x_0}$
while *stop condition not met* **do**
  $c_k \leftarrow \frac{c}{(k+\mathbf{1})^{\gamma}}$
  $a_k \leftarrow \frac{a}{(k+A+\mathbf{1})^{\alpha}}$
  for i=1:n **do**
    $\boldsymbol{\Delta_k}[i] \leftarrow \mathcal{B}^{\pm \mathbf{1}}(0.5)$ ▷ i.e. $Prob(\pm 1) = 0.5$
  end for
  $\boldsymbol{\theta}_{\boldsymbol{k}}^{+} \leftarrow \boldsymbol{\theta_k} + c_k \boldsymbol{\Delta_k}$
  $\boldsymbol{\theta}_{\boldsymbol{k}}^{-} \leftarrow \boldsymbol{\theta_k} - c_k \boldsymbol{\Delta_k}$
  $\hat{\mathbf{g}}_{\mathbf{k}} \leftarrow \frac{f\left(\boldsymbol{\theta}_{\boldsymbol{k}}^{+}\right) - f\left(\boldsymbol{\theta}_{\boldsymbol{k}}^{-}\right)}{\mathbf{2}c_k} \begin{bmatrix} 1/\boldsymbol{\Delta_k}[1] \\ 1/\boldsymbol{\Delta_k}[2] \\ \vdots \\ 1/\boldsymbol{\Delta_k}[n] \end{bmatrix}$
  $k \leftarrow k + 1$
  $\boldsymbol{\theta_k} \leftarrow \boldsymbol{\theta_{k-1}} - a_k \hat{\mathbf{g}}_{\mathbf{k}}$
end while
Output $\boldsymbol{\theta_k}$

References I

Bremermann, H. J. (1962).
Optimization through evolution and recombination.
In Yovits, M. C., Jacobi, G. T., and Golstine, G. D., editors, *Proceedings of the Conference on Self-Organizing Systems – 1962*, pages 93–106, Washington, DC. Spartan Books.

Caraffini, F. (2014).
Novel Memetic Computing Structures for Continuous Optimisation.
PhD thesis, De Montfort University, Leicetser, United Kingdom.
https://www.dora.dmu.ac.uk/xmlui/handle/2086/10629.

Fogel, L. J., Owens, A., and Walsh, M. (1964).
On the evolution of artificial intelligence(artificial intelligence generated by natural evolution process).
In *National Symposium on Human Factors in Electronics, 5th, San Diego, California*, pages 63–76.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966).
Artificial Intelligence through Simulated Evolution.
John Wiley.

References II

Holland, J. (1975).
Adaptation in Natural and Artificial Systems.
University of Michigan Press.

Kirkpatrick, S., Gelatt, C. D. J., and Vecchi, M. P. (1983).
Optimization by simulated annealing.
Science, (220):671–680.

Koza, J. R. (1990).
Concept formation and decision tree induction using the genetic programming paradigm.
In *Parallel Problem Solving from Nature*, pages 124–128. Springer.

Koza, J. R. (1992a).
Genetic programming: on the programming of computers by means of natural selection, volume 1.
MIT press.

References III

Koza, J. R. (1992b).
Genetic Programming: vol. 1, On the programming of computers by means of natural selection, volume 1.
MIT press.

Moscato, P. (1989).
On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms.
Technical Report 826.

Prügel-Bennett, A. (2010).
Benefits of a population: Five mechanisms that advantage population-based algorithms.
IEEE Transactions on Evolutionary Computation, 14(4):500–517.

Rechenberg, I. (1971).
Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.
PhD thesis, Technical University of Berlin.

References IV

Solis, F. J. and Wets, R. J.-B. (1981).
Minimization by random search techniques.
Mathematics of Operations Research, 6(1):19–30.

Spall, J. (1987).
A stochastic approximation technique for generating maximum likelihood parameter estimates.
In *American Control Conference, 1987*, pages 1161–1167.